Pic Pack Library

Embedded Adventures

www.embeddedadventures.com

Version 3.09
8/19/2011 9:04:00 AM

Table of Contents

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

File Index

File List

Here is a list of all files with brief descriptions:

Data Structure Documentation

buffer_descriptor Struct Reference

Data Fields

- uns16 addr
- uns8 count
- uns8 stat

Detailed Description

Describes the USB endpoint buffer descriptor

Field Documentation

uns16 buffer_descriptor::addr

uns8 buffer_descriptor::count

uns8 buffer_descriptor::stat

The documentation for this struct was generated from the following file:

- pic_usb.h

CDC_ACM_functional_descriptor Struct Reference

Data Fields

- uns8 capabilities
- uns8 descriptor_subtype
- uns8 descriptor_type
- uns8 length

Field Documentation

uns8 CDC_ACM_functional_descriptor::capabilities

uns8 CDC_ACM_functional_descriptor::descriptor_subtype

uns8 CDC_ACM_functional_descriptor::descriptor_type

uns8 CDC_ACM_functional_descriptor::length

The documentation for this struct was generated from the following file:

- pic_usb.h

CDC_call_mgt_functional_descriptor Struct Reference

Data Fields

- uns8 capabilities
- uns8 data_interface
- uns8 descriptor_subtype
- uns8 descriptor_type
- uns8 length

Field Documentation

uns8 CDC_call_mgt_functional_descriptor::capabilities

uns8 CDC_call_mgt_functional_descriptor::data_interface

uns8 CDC_call_mgt_functional_descriptor::descriptor_subtype

uns8 CDC_call_mgt_functional_descriptor::descriptor_type

uns8 CDC_call_mgt_functional_descriptor::length

The documentation for this struct was generated from the following file:

- pic_usb.h

CDC_header_functional_descriptor Struct Reference

Data Fields

- uns16 CDC_version
- uns8 descriptor_subtype
- uns8 descriptor_type
- uns8 length

Field Documentation

uns16 CDC_header_functional_descriptor::CDC_version

uns8 CDC_header_functional_descriptor::descriptor_subtype

uns8 CDC_header_functional_descriptor::descriptor_type

uns8 CDC_header_functional_descriptor::length

The documentation for this struct was generated from the following file:

- pic_usb.h

CDC_union_functional_descriptor Struct Reference

Data Fields

- uns8 descriptor_subtype
- uns8 descriptor_type
- uns8 length
- uns8 master_interface
- uns8 slave_interface

Field Documentation

uns8 CDC_union_functional_descriptor::descriptor_subtype

uns8 CDC_union_functional_descriptor::descriptor_type

uns8 CDC_union_functional_descriptor::length

uns8 CDC_union_functional_descriptor::master_interface

uns8 CDC_union_functional_descriptor::slave_interface

The documentation for this struct was generated from the following file:

- pic_usb.h

---

configuration_descriptor Struct Reference

Data Fields

- uns8 attributes
- uns8 configuration_string_id
- uns8 configuration_value
- uns8 descriptor_type
- uns8 length
- uns8 max_power
- uns8 num_interfaces
- uns16 total_length

Detailed Description

Configuration descriptor

Field Documentation

uns8 configuration_descriptor::attributes

uns8 configuration_descriptor::configuration_string_id

uns8 configuration_descriptor::configuration_value

uns8 configuration_descriptor::descriptor_type

uns8 configuration_descriptor::length

uns8 configuration_descriptor::max_power

uns8 configuration_descriptor::num_interfaces

uns16 configuration_descriptor::total_length

The documentation for this struct was generated from the following file:

- pic_usb.h

---

device_descriptor Struct Reference

Data Fields

- uns8 descriptor_type
- uns8 device_class
- uns8 device_protocol
- uns16 device_release
- uns8 device_subclass
- uns8 length
- uns8 manufacturer_string_id
- uns8 max_packet_size_ep0
- uns8 num_configurations
- uns16 product_id
- uns8 product_string_id
- uns8 serial_string_id
- uns16 usb_version
- uns16 vendor_id

Detailed Description

Device descriptor

Field Documentation

uns8 device_descriptor::descriptor_type

uns8 device_descriptor::device_class

uns8 device_descriptor::device_protocol

uns16 device_descriptor::device_release

uns8 device_descriptor::device_subclass

uns8 device_descriptor::length

uns8 device_descriptor::manufacturer_string_id

uns8 device_descriptor::max_packet_size_ep0

uns8 device_descriptor::num_configurations

uns16 device_descriptor::product_id

uns8 device_descriptor::product_string_id

uns8 device_descriptor::serial_string_id

uns16 device_descriptor::usb_version

uns16 device_descriptor::vendor_id

The documentation for this struct was generated from the following file:

- pic_usb.h

---

endpoint_descriptor Struct Reference

Data Fields

- uns8 attributes
- uns8 descriptor_type
- uns8 endpoint_address
- uns8 interval
- uns8 length
- uns16 max_packet_size

Detailed Description

Endpoint descriptor

Field Documentation

uns8 endpoint_descriptor::attributes

uns8 endpoint_descriptor::descriptor_type

uns8 endpoint_descriptor::endpoint_address

uns8 endpoint_descriptor::interval

uns8 endpoint_descriptor::length

uns16 endpoint_descriptor::max_packet_size

The documentation for this struct was generated from the following file:

- pic_usb.h

---

hid_descriptor Struct Reference

Data Fields

- uns16 class_descriptor_length
- uns8 class_descriptor_type
- uns8 country_code
- uns8 descriptor_type
- uns16 hid_spec
- uns8 length
- uns8 num_class_descriptors

Detailed Description

Human Interface Device descriptor

Field Documentation

uns16 hid_descriptor::class_descriptor_length

uns8 hid_descriptor::class_descriptor_type

uns8 hid_descriptor::country_code

uns8 hid_descriptor::descriptor_type

uns16 hid_descriptor::hid_spec

uns8 hid_descriptor::length

uns8 hid_descriptor::num_class_descriptors

The documentation for this struct was generated from the following file:

- pic_usb.h

---

interface_descriptor Struct Reference

Data Fields

- uns8 alternate_setting
- uns8 descriptor_type
- uns8 interface_class
- uns8 interface_number
- uns8 interface_protocol
- uns8 interface_string_id
- uns8 interface_subclass
- uns8 length
- uns8 num_endpoints

Detailed Description

Interface descriptor

Field Documentation

uns8 interface_descriptor::alternate_setting

uns8 interface_descriptor::descriptor_type

uns8 interface_descriptor::interface_class

uns8 interface_descriptor::interface_number

uns8 interface_descriptor::interface_protocol

uns8 interface_descriptor::interface_string_id

uns8 interface_descriptor::interface_subclass

uns8 interface_descriptor::length

uns8 interface_descriptor::num_endpoints

The documentation for this struct was generated from the following file:

- pic_usb.h

---

its2_packet Struct Reference

Data Fields

- uns8 hop_count
- uns16 its_dest_id
- uns16 its_network_id
- uns16 its_source_id
- uns8 max_hop_count
- uns8 num_routes
- uns8 packet_type
- uns16 routers [ITS2_MAX_HOP_COUNT]
- uns8 sequence

Detailed Description

ITS2 Packet definition

Field Documentation

uns8 its2_packet::hop_count

Number of routes (hops) specified for this packet to take

uns16 its2_packet::its_dest_id

ITS device ID of sender

uns16 its2_packet::its_network_id

Sequence of packet (incremented for each packet)

uns16 its2_packet::its_source_id

Network ID of packet

uns8 its2_packet::max_hop_count

ITS device ID of destination

uns8 its2_packet::num_routes

Maximum number of permitted routes

uns8 its2_packet::packet_type

uns16 its2_packet::routers[ITS2_MAX_HOP_COUNT]

Number of routes (hops) already made by this packet

uns8 its2_packet::sequence

Type of packet (see its_common.h)

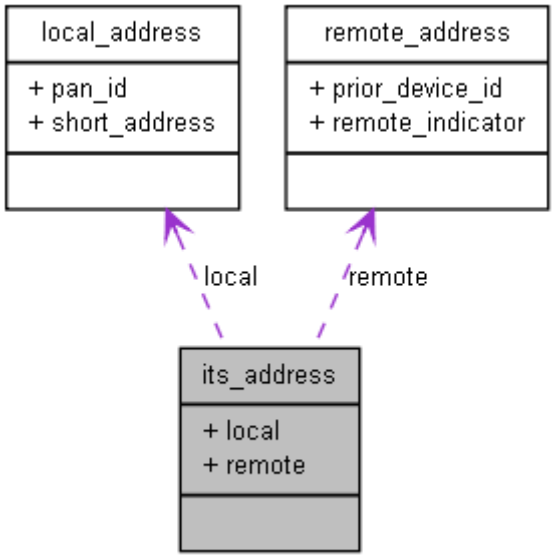
The documentation for this struct was generated from the following file:

- its_mode2.h

---

its_address Union Reference

Collaboration diagram for its_address:



Data Fields

- local_address local
- remote_address remote

Detailed Description

Union of local and remote addresses

Field Documentation

local_address its_address::local

remote_address its_address::remote

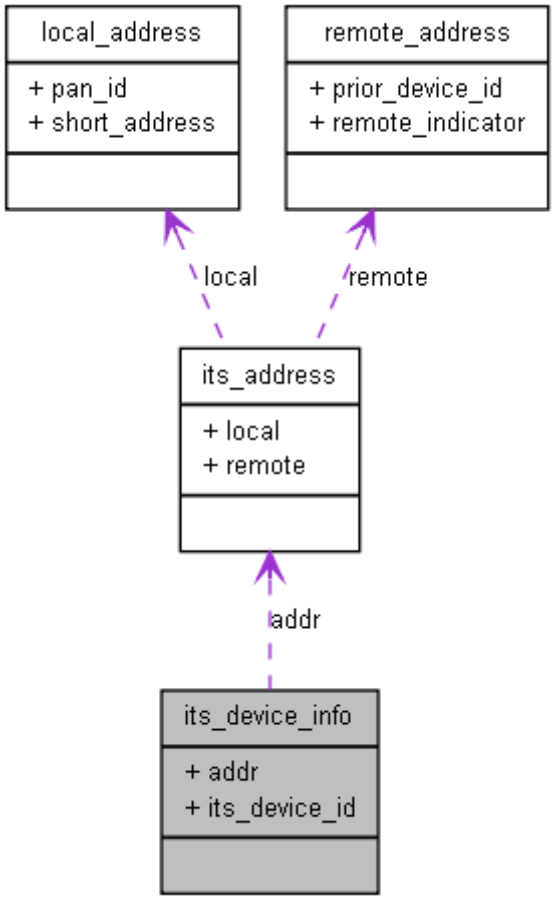
The documentation for this union was generated from the following file:

- its_common.h

---

its_device_info Struct Reference

Collaboration diagram for its_device_info:



Data Fields

- its_address addr
- uns16 its_device_id

Detailed Description

Device info - its_device_id and address (local or remote)

Field Documentation

its_address its_device_info::addr

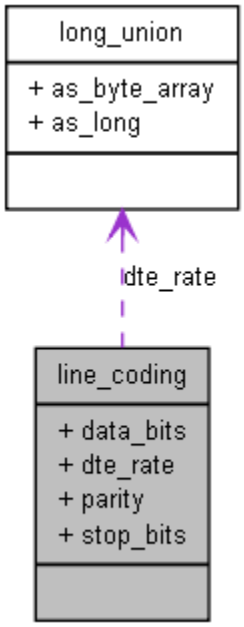
uns16 its_device_info::its_device_id

The documentation for this struct was generated from the following file:

- its_common.h

line_coding Struct Reference

Collaboration diagram for line_coding:



Data Fields

- uns8 data_bits
- long_union dte_rate
- uns8 parity
- uns8 stop_bits

Detailed Description

Line coding struction that defines what (should) happen if we were actually USB to RS232 converter

---

Field Documentation

uns8 line_coding::data_bits

long_union line_coding::dte_rate

uns8 line_coding::parity

uns8 line_coding::stop_bits

The documentation for this struct was generated from the following file:

- usb_cdc_class.c

local_address Struct Reference

Data Fields

- uns16 pan_id
- uns16 short_address

Detailed Description

Local address definition

Field Documentation

uns16 local_address::pan_id

uns16 local_address::short_address

The documentation for this struct was generated from the following file:

- its_common.h

long_union Union Reference

Data Fields

- uns8 as_byte_array [4]
- long as_long

Field Documentation

uns8 long_union::as_byte_array[4]

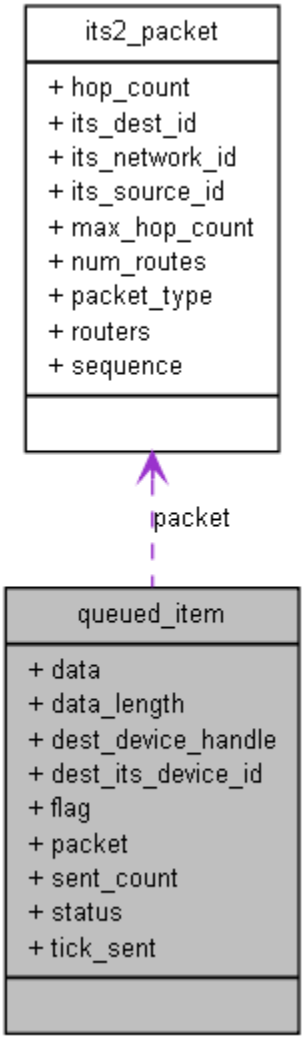
long long_union::as_long

The documentation for this union was generated from the following file:

- usb_cdc_class.c

queued_item Struct Reference

Collaboration diagram for queued_item:



Data Fields

- uns8 * data
- uns8 data_length
- uns8 dest_device_handle
- uns16 dest_its_device_id
- uns8 flag
- its2_packet packet
- uns8 sent_count
- uns8 status
- uns16 tick_sent

Field Documentation

uns8* queued_item::data

uns8 queued_item::data_length

uns8 queued_item::dest_device_handle

uns16 queued_item::dest_its_device_id

uns8 queued_item::flag

its2_packet queued_item::packet

uns8 queued_item::sent_count

uns8 queued_item::status

uns16 queued_item::tick_sent

The documentation for this struct was generated from the following file:

- its_mode2.h

---

remote_address Struct Reference

Data Fields

- uns16 prior_device_id
- uns16 remote_indicator

Detailed Description

Remote address definition

Field Documentation

uns16 remote_address::prior_device_id

uns16 remote_address::remote_indicator

The documentation for this struct was generated from the following file:

- its_common.h

rf_config Struct Reference

Data Fields

- uns8 address_ch1 [5]
- uns8 address_ch2 [5]
- uns8 address_width
- uns8 channel
- uns8 crystal
- uns8 options
- uns8 output_power
- uns8 payload_width_ch1
- uns8 payload_width_ch2

Detailed Description

RF configuration structure

Field Documentation

uns8 rf_config::address_ch1

Address of channel 1

uns8 rf_config::address_ch2

Address of channel 2

uns8 rf_config::address_width

Address width in bits

uns8 rf_config::channel

Channel to operate on, 7 bits valid

uns8 rf_config::crystal

Crystal frequency look up, 3 bits valid

uns8 rf_config::options

Options (see option bits)

uns8 rf_config::output_power

Output power, 2 bits valid

uns8 rf_config::payload_width_ch1

Payload width of channel 1 in bits

uns8 rf_config::payload_width_ch2

Payload width of channel 2 in bits

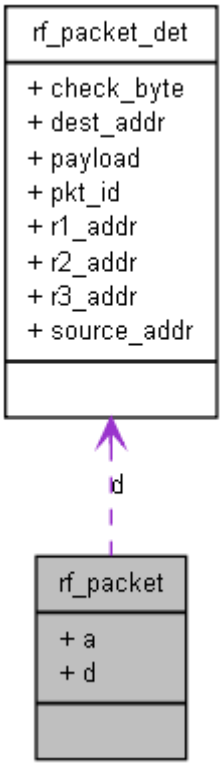
The documentation for this struct was generated from the following files:

- pic_rf_2401a.h
- pic_rf_24l01.h

---

rf_packet Union Reference

Collaboration diagram for rf_packet:



Data Fields

- char a [PKT_PACKET_SIZE]
- rf_packet_det d

Field Documentation

char rf_packet::a[PKT_PACKET_SIZE]

rf_packet_det rf_packet::d

The documentation for this union was generated from the following file:

- pic_packet.c

---

rf_packet_det Struct Reference

Data Fields

- uns8 check_byte
- uns16 dest_addr
- uns8 payload [PKT_PAYLOAD_SIZE]
- uns16 pkt_id
- uns16 r1_addr
- uns16 r2_addr
- uns16 r3_addr
- uns16 source_addr

Detailed Description

Internal packet structure. You shouldn't need to worry about this generally

---

Field Documentation

uns8 rf_packet_det::check_byte

uns16 rf_packet_det::dest_addr

uns8 rf_packet_det::payload[PKT_PAYLOAD_SIZE]

uns16 rf_packet_det::pkt_id

uns16 rf_packet_det::r1_addr

uns16 rf_packet_det::r2_addr

uns16 rf_packet_det::r3_addr

uns16 rf_packet_det::source_addr

The documentation for this struct was generated from the following file:

- pic_packet.h

---

seen_packet Struct Reference

Data Fields

- uns16 its_source_id
- uns16 pkt_id
- uns8 sequence
- uns16 source_addr

Field Documentation

uns16 seen_packet::its_source_id

uns16 seen_packet::pkt_id

uns8 seen_packet::sequence

uns16 seen_packet::source_addr

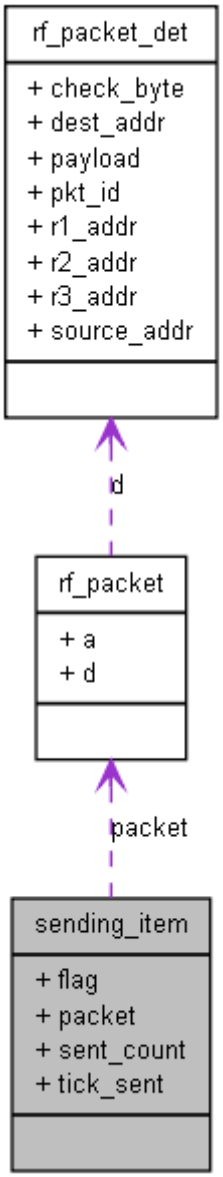
The documentation for this struct was generated from the following files:

- its_mode2.h
- pic_packet.c

---

sending_item Struct Reference

Collaboration diagram for sending_item:



Data Fields

- uns8 flag
- rf_packet packet
- uns8 sent_count
- uns16 tick_sent

Field Documentation

uns8 sending_item::flag

rf_packet sending_item::packet

uns8 sending_item::sent_count

uns16 sending_item::tick_sent

The documentation for this struct was generated from the following file:

- pic_packet.c

---

setup_data_packet Struct Reference

Data Fields

- uns8 bmRequestType
- uns8 bRequest
- uns16 wIndex
- uns16 wLength
- uns16 wValue

Detailed Description

Describes a set up data packet, part of the control transfer

Field Documentation

uns8 setup_data_packet::bmRequestType

uns8 setup_data_packet::bRequest

uns16 setup_data_packet::wIndex

uns16 setup_data_packet::wLength

uns16 setup_data_packet::wValue

The documentation for this struct was generated from the following file:

- pic_usb.h

---

wpan_address Struct Reference

Data Fields

- uns8 dest_address_type
- uns8 dest_ea [8]
- uns16 dest_pan_id
- uns16 dest_sa
- uns8 source_address_type
- uns8 source_ea [8]
- uns16 source_pan_id
- uns16 source_sa

Field Documentation

uns8 wpan_address::dest_address_type

uns8 wpan_address::dest_ea[8]

uns16 wpan_address::dest_pan_id

uns16 wpan_address::dest_sa

uns8 wpan_address::source_address_type

uns8 wpan_address::source_ea[8]

uns16 wpan_address::source_pan_id

uns16 wpan_address::source_sa

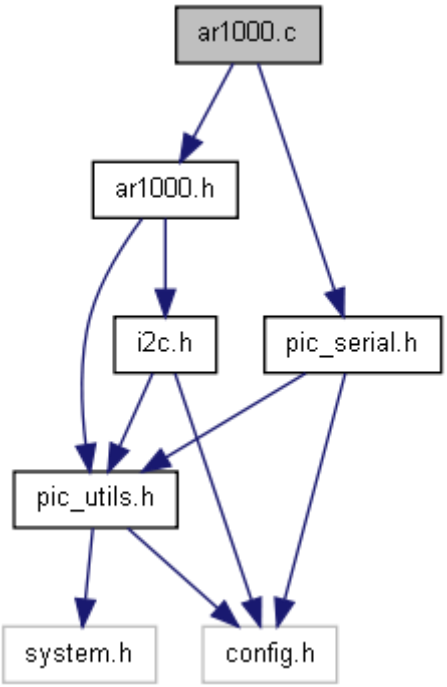
The documentation for this struct was generated from the following file:

- wpan.h

File Documentation

ar1000.c File Reference

Include dependency graph for ar1000.c:



Functions

- uns8 ar1000_get_register (uns8 reg)
- void ar1000_init ()
- uns16 ar1000_read_register (uns8 reg)
- void ar1000_read_registers ()
- void ar1000_seek (uns16 frequency, bit seek_up)
- void ar1000_seek2 ()
- void ar1000_seek_more ()
- void ar1000_set_register (uns8 reg, uns8 data)
- void ar1000_set_seek_threshold (uns8 new_seek_threshold)
- void ar1000_set_volume (uns8 volume)
- void ar1000_setup_io ()
- *Setup AR1000 ports and pins.* void ar1000_test ()
- void ar1000_tune (uns16 frequency)
- void ar1000_write_register (uns8 reg, uns16 data)
- void ar1000_write_registers ()

Variables

- uns8 ar1000_seek_threshold
- uns16 regs [18]
- rom uns8 vol_lookup []

Function Documentation

uns8 ar1000_get_register (uns8 *reg*)

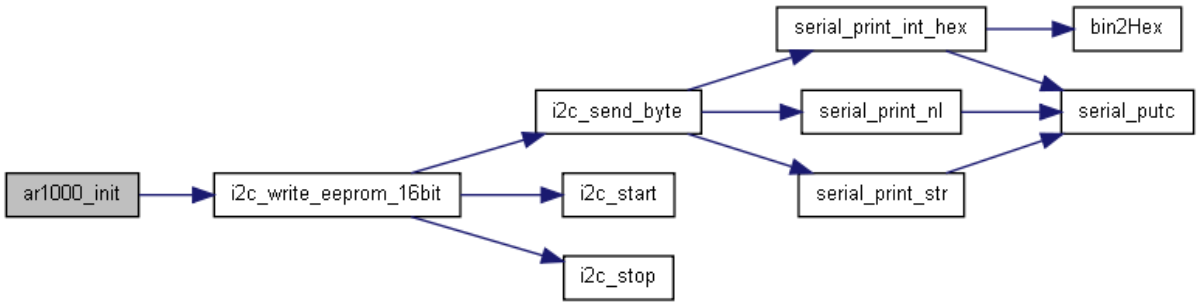
```
56 {
```

```
57     return regs[reg];
58 }
```

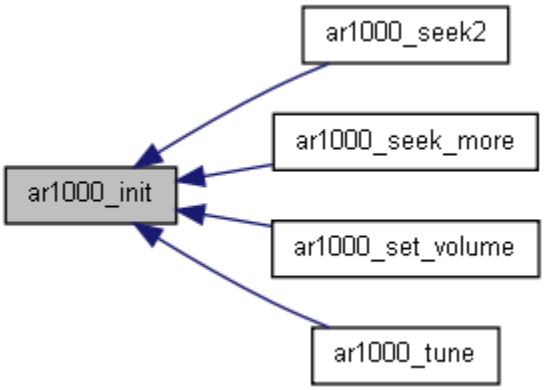
void ar1000_init ()

```
224                     {
225     ar1000_seek_threshold = 0;
226     for (uns8 count = 1; count < 18; count ++) {
227         i2c_write_eeprom_16bit(AR1000_DEV_ADDR, count, regs[count]);
228     }
229     i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 0, regs[0]);
230 
231 }
```

Here is the call graph for this function:



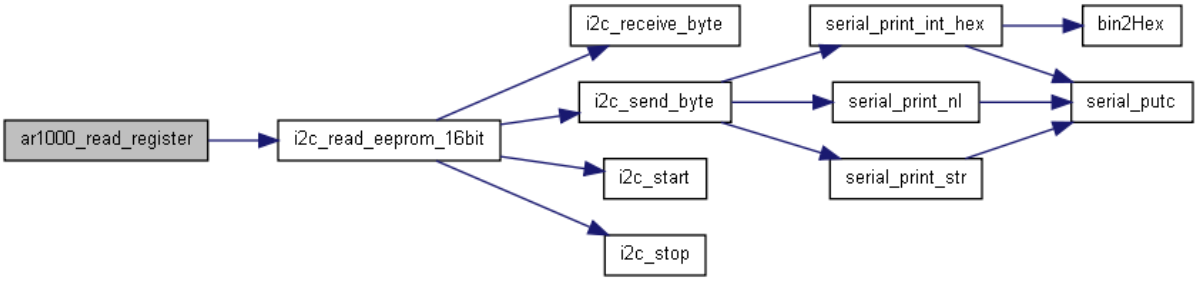
Here is the caller graph for this function:



uns16 ar1000_read_register (uns8 *reg*)

```
71 {
72 
73     return i2c_read_eeprom_16bit(AR1000_DEV_ADDR, reg);
74 
75 }
```

Here is the call graph for this function:



void ar1000_read_registers ()

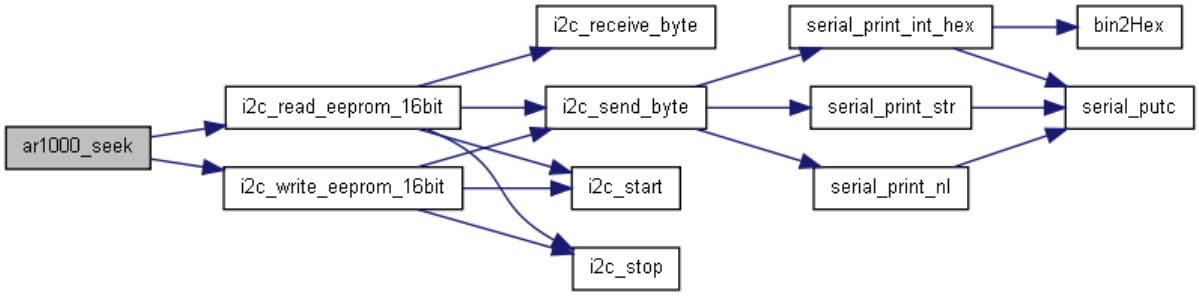
```
66                                  {
67 }
```

void ar1000_seek (uns16 *frequency*, bit *seek_up*)

```
83                                                  {
84
85 uns16 r1, r2, r3;
86
87 r1 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 1);
88 r2 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 2);
89 r3 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 3);
90
91 // Set hmute bit
92 set_bit(r1, R1_HARD_MUTE_ENABLE);
93
94 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, r1);
95
96 // clear tune bit
97 clear_bit(r2, R2_TUNE_ENABLE);
98
99 // set chan bits r2 to 87.5Mhz
100 // Means setting the bits to 185
101 r2 = r2 & 0b1000000000000000;   // Mask out tune bits
102 //r2 = r2 + (frequency - 690);  // eg 875 - 690 = 185 for 87.5Mhz
103 r2 = r2 & ((frequency - 690) << 7); // eg 875 - 690 = 185 for 87.5Mhz
104
105 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 2, r2);
106
107 // clear seek bit
108 clear_bit(r3, R3_SEEK_ENABLE);
109 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, r3);
110
111 // Set SEEK UP/DOWN / SPACE / BAND / SEEKTH
112 if (seek_up) {
113     set_bit(r3, R3_SEEK_UP);
114 } else {
115     clear_bit(r3, R3_SEEK_UP);
116 }
117 set_bit(r3, R3_SEEK_CHANNEL_SPACING);   // 100k spacing
118 clear_bit(r3, R3_BAND_1);   // US / EUROPE
119
120 r3 = r3 & 0b1111111110000000 + ar1000_seek_threshold;
121 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, r3);
122
123 // Enable SEEK Bit
124 set_bit(r3, R3_SEEK_ENABLE);
125 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, r3);
126
127 // Wait STC flag (Seek/Tune Complete, in “Status” register)
```

```
128 // Not done yet! not sure which register it is...
129
130 // Clear hmute Bit
131 clear_bit(r1, R1_HARD_MUTE_ENABLE);
132 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, r1);
133 //  register_values[02] = 0xB480;  //set tune to 900kHz
134 //  register_values[03] = 0xA001;  //turn off seek, seek up, set threshold to 1
135
136 //  ar1000calibration(register_values);
137
138 //  register values[03] = 0xE001; //turns on seek
139
140 //  ar1000calibration(register_values);
141
142
143 }
```

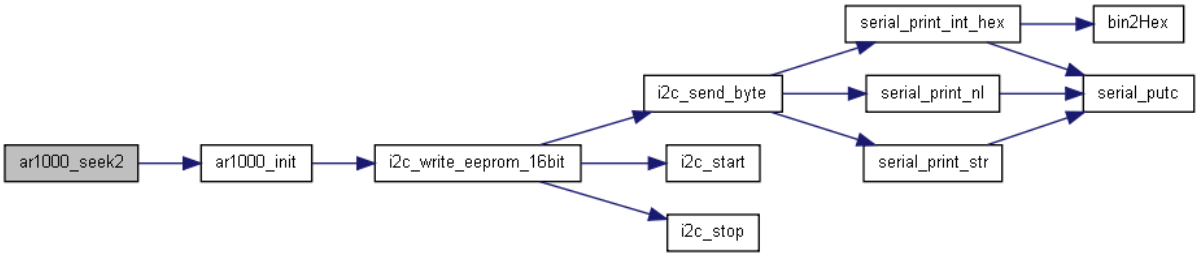
Here is the call graph for this function:



void ar1000_seek2 ()

```
234                     {
235 //    i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 2, 0xb480);
236  //   i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, 0xa001);
237   //  i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, 0xe001);
238   //0xF4B9
239     regs[2] = 0xcd80; //0xB480;//  0xf4b9; //0b1111111000000000;
240     //ar1000_init();
241     regs[3] = 0xa001;
242     ar1000_init();
243     regs[3] = 0xe001;
244     ar1000_init();
245     //  register_values[02] = 0xB480;  //set tune to 900kHz
246 //  register_values[03] = 0xA001;  //turn off seek, seek up, set threshold to 1
247
248 //  ar1000calibration(register_values);
249
250 //  register_values[03] = 0xE001; //turns on seek
251
252 //  ar1000calibration(register_values);
253 }
```

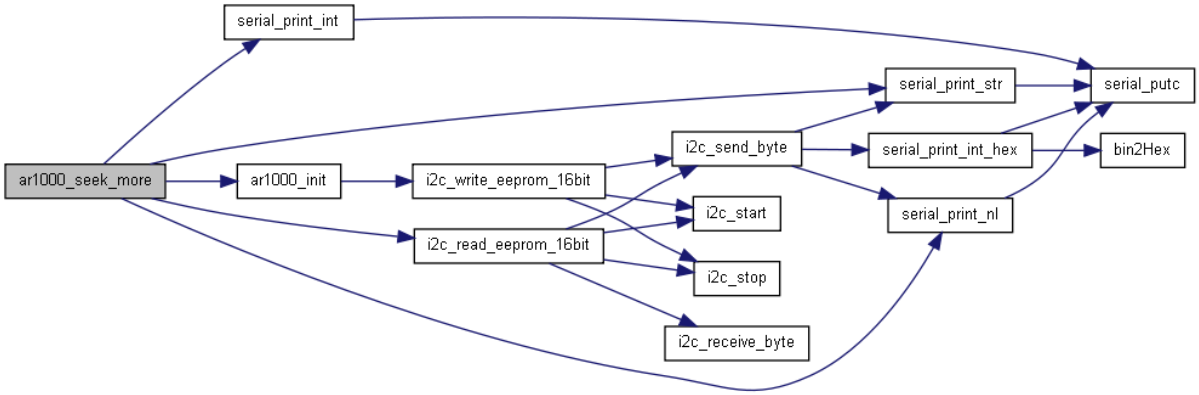
Here is the call graph for this function:



void ar1000_seek_more ()

```
273                          {
274 //    i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 2, 0xb480);
275  //   i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, 0xa001);
276   //  i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, 0xe001);
277   //0xF4B9
278 //    regs[2] = 0; //0xB480;//  0xf4b9; //0b1111111000000000;
279     set_bit(regs[2], R2_TUNE_ENABLE);
280     uns16 channel = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, AR1000_STATUS);
281     channel = channel >> 7;
282     serial_print_str(" fr=");
283     serial_print_int(channel+690);
284     serial_print_nl();
285     //channel = channel << 6;
286     regs[2] = channel;
287     set_bit(regs[2], R2_TUNE_ENABLE);
288     //ar1000_init();
289     regs[3] = 0xa005;
290     ar1000_init();
291     regs[3] = 0xe005;
292     ar1000_init();
293 }
```

Here is the call graph for this function:



void ar1000_set_register (uns8 *reg*, uns8 *data*)

```
51 {
52     regs[reg] = data;
53 }
```

void ar1000_set_seek_threshold (uns8 *new_seek_threshold*)

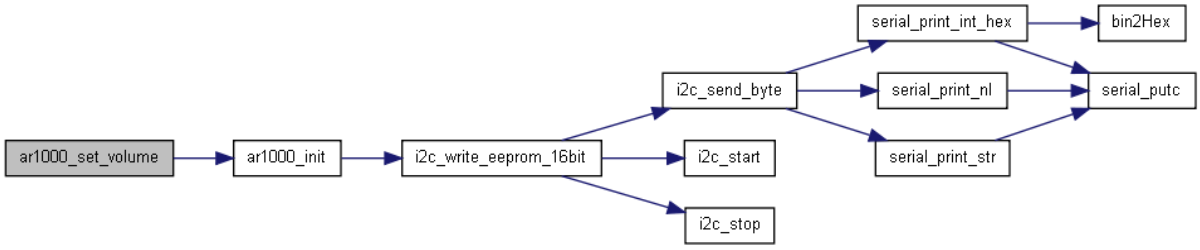
```
220 {
221     ar1000_seek_threshold = new_seek_threshold;
222 }
```

void ar1000_set_volume (uns8 *volume*)

```
321                                  {
```

```
322
323 uns16 reg, temp;
324 uns8  vol;
325 uns16 vol1, vol2;
326
327     if (volume > 21) { return; }
328     vol = vol_lookup[volume];
329     vol2 = vol >> 4;
330     vol1 = vol & 0x0f;
331
332
333     regs[3] = (regs[3] & ~0x0780) | (vol1 << 7);
334     //write(3, register_values[3]);
335
336     regs[14] = (regs[14] & ~0xF000) | (vol2 << 12);
337     //write(14, register_values[14]);
338
339     //serial_print_str(" AR3=");
340     //serial_print_int_hex_16bit(reg);
341     //regs[3] = reg;
342     ar1000_init();
343 }
```

Here is the call graph for this function:



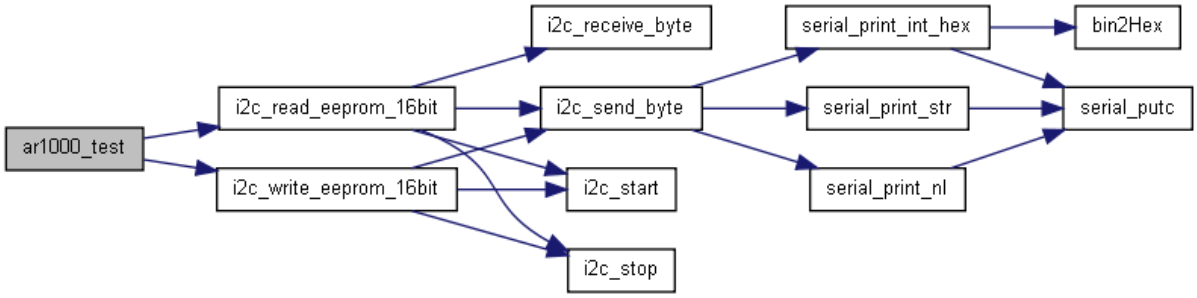
void ar1000_setup_io ()

```
46                          {
47     i2c_setup();
48 }
```

void ar1000_test ()

```
255                         {
256 /*  serial_print_str("Read 01 = ");
257     uns16 r1 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 1);
258     serial_print_int_hex_16bit(r1);
259     serial_print_nl();
260     serial_print_str("Write  ");
261     i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, 0xabcd);
262     serial_print_str("Read 01 = ");
263     r1 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 1);
264     serial_print_int_hex_16bit(r1);
265     serial_print_nl();
266 */
267     uns16 r1 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 1);
268     toggle_bit(r1, R1_HARD_MUTE_ENABLE);
269     i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, r1);
270
271 }
```

Here is the call graph for this function:

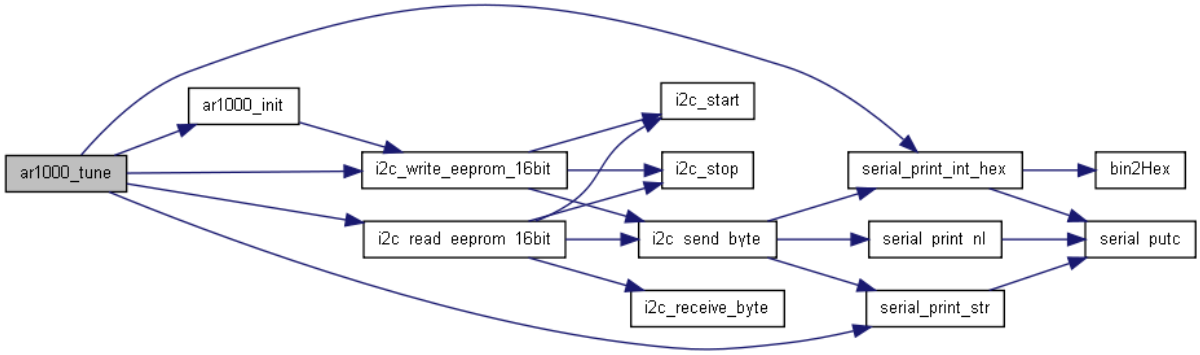


void ar1000_tune (uns16 *frequency*)

```
145                                     {
146 
147 uns16 r1, r2, r3;
148 set_bit(regs[3], R3_SEEK_CHANNEL_SPACING);
149 regs[2] = 0b0000000100111111;
150 ar1000_init();
151 regs[2] = 0b0000001100111111;
152 
153 ar1000_init();  //a7e0
154 return;
155 r1 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 1);
156 r2 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 2);
157 serial_print_str("r2o=");
158 serial_print_int_hex(r2);
159 r3 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 3);
160 //1 100111111 000000
161 // Set hmute bit
162 set_bit(r1, R1_HARD_MUTE_ENABLE);
163 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, r1);
164 
165 
166 // clear tune bit
167 clear_bit(r2, R2_TUNE_ENABLE);
168 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 2, r2);
169 
170 // clear seek bit
171 clear_bit(r3, R3_SEEK_ENABLE);
172 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, r3);
173 
174 // Set  SPACE / BAND / CHAN
175 set_bit(r3, R3_SEEK_CHANNEL_SPACING);   // 100k spacing
176 clear_bit(r3, R3_BAND_1);   // US / EUROPE
177 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, r3);
178 
179 // set chan bits r2 to 87.5Mhz
180 // Means setting the bits to 185
181 //serial_print_str("f=");
182 //serial_print_int(frequency);
183 //serial_print_nl();
184 r2 = frequency - 690;
185 //serial_print_str(" f-690=");
186 //serial_print_int(r2);
187 r2 = r2 << 6;
188 //serial_print_str("tuning to=");
189 //serial_print_int_hex_16bit(r2);
190 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 2, r2);
191 // 100111100 0000000
192 // Enable TUNE Bit
193 set_bit(r2, R2_TUNE_ENABLE);
194 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 2, r2);
195 //1 001111000 00000
196 //serial_print_str(" r2en=");
197 //serial_print_int_hex_16bit(r2);
```

```
198
199 // Wait STC flag (Seek/Tune Complete, in "Status" register)
200 // Not done yet! not sure which register it is...
201 // Clear hmute Bit
202 clear_bit(r1, R1_HARD_MUTE_ENABLE);
203 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, r1);
204 //  register_values[02] = 0xB480;  //set tune to 900kHz
205 //  register_values[03] = 0xA001;  //turn off seek, seek up, set threshold to 1
206
207 //  ar1000calibration(register_values);
208
209 //  register_values[03] = 0xE001; //turns on seek
210
211 //  ar1000calibration(register_values);
212
213
214
215
216 }
```

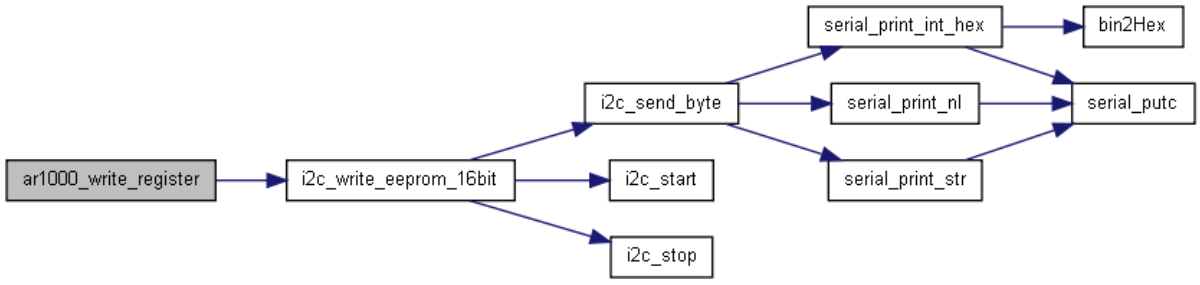
Here is the call graph for this function:



void ar1000_write_register (uns8 *reg*, uns16 *data*)

```
77                                                  {
78
79      i2c_write_eeprom_16bit(AR1000_DEV_ADDR, reg, data);
80
81 }
```

Here is the call graph for this function:



void ar1000_write_registers ()

```
60                                  {
61 uns8 count;
```

```
62
63 }
```

Variable Documentation

uns8 ar1000_seek_threshold

uns16 regs[18]

```
Initial value: {
    0xffff,0x5b15,0xF4B9,0x8012,0x0400,0x28aa,0x4400,0x1ee7,
    0x7141,0x007d,0x82ce,0x4f55,0x970c,0xb845,0xfc2d,0x8097,
    0x04a1,0xdf6a}
```

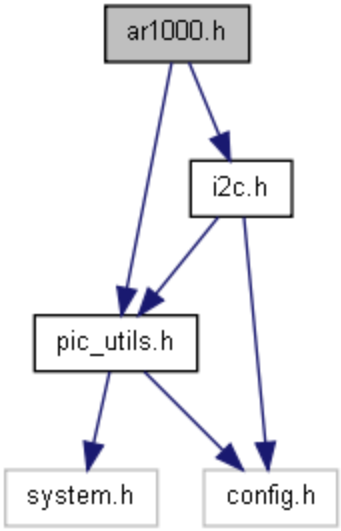
rom uns8 vol_lookup[]

```
Initial value: {
0x0F,
0xCF,
0xDF,
0xEF,
0xFF,
0xEE,
0xFE,
0xED,
0xFD,
0xFB,
0xFA,
0xF9,
0xF7,
0xE6,
0xF6,
0xE5,
0xF5,
0xE3,
0xF3,
0xF2,
0xF1,
0xF0}
```

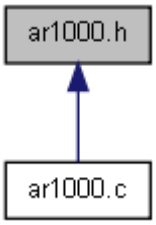
---

ar1000.h File Reference

Routines to access the AR1000 FM radio chip.
Include dependency graph for ar1000.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define AR1000_CHIP_ID 28
- #define AR1000_DEV_ADDR 0b00100000
- #define AR1000_DEV_ID 27
- #define AR1000_R0 0
- #define AR1000_R1 1
- #define AR1000_R10 10
- #define AR1000_R11 11
- #define AR1000_R13 13
- #define AR1000_R14 14
- #define AR1000_R15 15
- #define AR1000_R2 2
- #define AR1000_R3 3
- #define AR1000_RBS 20
- #define AR1000_RDS_1 21
- #define AR1000_RDS_2 22
- #define AR1000_RDS_3 23
- #define AR1000_RDS_4 24
- #define AR1000_RDS_5 25
- #define AR1000_RDS_6 26
- #define AR1000_RSSI 18
- #define ar1000_setup() ar1000_setup_io()
- *Setup AR1000 ports and pins.* #define AR1000_STATUS 19
- #define DEV_ID_MFID_0 0

- #define DEV_ID_MFID_1 1
- #define DEV_ID_MFID_10 10
- #define DEV_ID_MFID_11 11
- #define DEV_ID_MFID_2 2
- #define DEV_ID_MFID_3 3
- #define DEV_ID_MFID_4 4
- #define DEV_ID_MFID_5 5
- #define DEV_ID_MFID_6 6
- #define DEV_ID_MFID_7 7
- #define DEV_ID_MFID_8 8
- #define DEV_ID_MFID_9 9
- #define DEV_ID_VERSION_0 12
- #define DEV_ID_VERSION_1 13
- #define DEV_ID_VERSION_2 14
- #define DEV_ID_VERSION_3 15
- #define R0_ENABLE 0
- #define R0_INT_OSC_EN 15
- #define R10_SEEK_WRAP_ENABLE 3
- #define R11_AFC_HIGH_SIDE_b1 2
- #define R11_AFC_HIGH_SIDE_b2 0
- #define R11_AFC_INJECTION_CONTROL 15
- #define R11_HILO_SIDE 15
- #define R13_GPIO1_0 0
- #define R13_GPIO1_1 1
- #define R13_GPIO2_0 2
- #define R13_GPIO2_1 3
- #define R13_GPIO3_0 4
- #define R13_GPIO3_1 5
- #define R14_VOL2_0 12
- #define R14_VOL2_1 13
- #define R14_VOL2_2 14
- #define R14_VOL2_3 15
- #define R15_RDS_CTRL 0
- #define R15_RDS_MECC_0 3
- #define R15_RDS_MECC_1 4
- #define R15_RDS_STA_EN 5
- #define R1_DEEMP_SETTING 4
- #define R1_FORCE_MONO 3
- #define R1_HARD_MUTE_ENABLE 1
- #define R1_RDS_ENABLE 13
- #define R1_RDS_INT_ENABLE 6
- #define R1_SOFT_MUTE_ENABLE 2
- #define R1_STC_INT_ENABLE 5
- #define R2_CHAN_0 0
- #define R2_CHAN_1 1
- #define R2_CHAN_2 2
- #define R2_CHAN_3 3
- #define R2_CHAN_4 4
- #define R2_CHAN_5 5
- #define R2_CHAN_6 6
- #define R2_CHAN_7 7
- #define R2_CHAN_8 8
- #define R2_TUNE_ENABLE 9

- #define R3_BAND_0 11
- #define R3_BAND_1 12
- #define R3_SEEK_CHANNEL_SPACING 13
- #define R3_SEEK_ENABLE 14
- #define R3_SEEK_UP 15
- #define R3_SEEKTH_0 0
- #define R3_SEEKTH_1 1
- #define R3_SEEKTH_2 2
- #define R3_SEEKTH_3 3
- #define R3_SEEKTH_4 4
- #define R3_SEEKTH_5 5
- #define R3_SEEKTH_6 6
- #define R3_VOL_0 7
- #define R3_VOL_1 8
- #define R3_VOL_2 9
- #define R3_VOL_3 10
- #define STATUS_BIT_2 2
- #define STATUS_CHAN_0 7
- #define STATUS_CHAN_1 8
- #define STATUS_CHAN_2 9
- #define STATUS_CHAN_3 10
- #define STATUS_CHAN_4 11
- #define STATUS_CHAN_5 12
- #define STATUS_CHAN_6 13
- #define STATUS_CHAN_7 14
- #define STATUS_CHAN_8 15
- #define STATUS_RDS_DATA_READY 6
- #define STATUS_SEEK_FAIL 4
- #define STATUS_SEEK_TUNE_COMPLETE 5
- #define STATUS_STEREO 3

Functions

- uns8 ar1000_get_register (uns8 reg)
- void ar1000_init ()
- uns16 ar1000_read_register (uns8 reg)
- void ar1000_read_registers ()
- void ar1000_seek (uns16 frequency, bit seek_up)
- void ar1000_seek2 ()
- void ar1000_seek_more ()
- void ar1000_set_register (uns8 reg, uns8 data)
- void ar1000_set_seek_threshold (uns8 new_seek_threshold)
- void ar1000_set_volume (uns8 volume)
- void ar1000_setup_io ()
- *Setup AR1000 ports and pins.* void ar1000_test ()
- void ar1000_tune (uns16 frequency)
- void ar1000_write_register (uns8 reg, uns16 data)
- void ar1000_write_registers ()

Detailed Description

ITS networking routines - common to mode 1 and 2.

Define Documentation

#define AR1000_CHIP_ID 28

Chip id - 0x1000 for RDS version, 0x1010 for non-rds version

#define AR1000_DEV_ADDR 0b00100000

#define AR1000_DEV_ID 27

#define AR1000_R0 0

#define AR1000_R1 1

#define AR1000_R10 10

#define AR1000_R11 11

#define AR1000_R13 13

GPIO controls in this register

#define AR1000_R14 14

#define AR1000_R15 15

#define AR1000_R2 2

#define AR1000_R3 3

#define AR1000_RBS 20

#define AR1000_RDS_1 21

#define AR1000_RDS_2 22

#define AR1000_RDS_3 23

#define AR1000_RDS_4 24

#define AR1000_RDS_5 25

#define AR1000_RDS_6 26

#define AR1000_RSSI 18

#define ar1000_setup() ar1000_setup_io()

define so as not to break existing code with new naming standard

#define AR1000_STATUS 19

#define DEV_ID_MFID_0 0

MFID (12 bits 5B1) bit 0

#define DEV_ID_MFID_1 1
MFID (12 bits 5B1) bit 1

#define DEV_ID_MFID_10 10
MFID (12 bits 5B1) bit 10

#define DEV_ID_MFID_11 11
MFID (12 bits 5B1) bit 11

#define DEV_ID_MFID_2 2
MFID (12 bits 5B1) bit 2

#define DEV_ID_MFID_3 3
MFID (12 bits 5B1) bit 3

#define DEV_ID_MFID_4 4
MFID (12 bits 5B1) bit 4

#define DEV_ID_MFID_5 5
MFID (12 bits 5B1) bit 5

#define DEV_ID_MFID_6 6
MFID (12 bits 5B1) bit 6

#define DEV_ID_MFID_7 7
MFID (12 bits 5B1) bit 7

#define DEV_ID_MFID_8 8
MFID (12 bits 5B1) bit 8

#define DEV_ID_MFID_9 9
MFID (12 bits 5B1) bit 9

#define DEV_ID_VERSION_0 12
Version of FM radio bit 0

#define DEV_ID_VERSION_1 13
Version of FM radio bit 1

#define DEV_ID_VERSION_2 14
Version of FM radio bit 2

#define DEV_ID_VERSION_3 15
Version of FM radio bit 3

#define R0_ENABLE 0

#define R0_INT_OSC_EN 15

#define R10_SEEK_WRAP_ENABLE 3

Seek wrap enable

#define R11_AFC_HIGH_SIDE_b1 2

High side control bits

#define R11_AFC_HIGH_SIDE_b2 0

#define R11_AFC_INJECTION_CONTROL 15

AFC injection control

#define R11_HILO_SIDE 15

#define R13_GPIO1_0 0

#define R13_GPIO1_1 1

GPIO1- 00-Disable 01-Reserved 10-Output logic 0 11-Output logic 1

#define R13_GPIO2_0 2

#define R13_GPIO2_1 3

GPIO2- 00-Disable 01-STC or RDS interrupt 10-Output logic 0 11-Output logic 1

#define R13_GPIO3_0 4

#define R13_GPIO3_1 5

GPIO3- 00-Disable 01-Stereo indication 10-Output logic 0 11-Output logic 1

#define R14_VOL2_0 12

Second volume settings (bit 0)

#define R14_VOL2_1 13

Second volume settings (bit 1)

#define R14_VOL2_2 14

Second volume settings (bit 2)

#define R14_VOL2_3 15

Second volume settings (bit 3)

#define R15_RDS_CTRL 0

RDS Control mode 0=block mode (RDSR asserted after 4 blocks received, data in blocks may not be correct. RBS1-4 will present the status of each block) 1=group mode (RDSR is asserted after 4 blocks received without any error)

#define R15_RDS_MECC_0 3

#define R15_RDS_MECC_1 4

RDS Error correction 0x=disable erorr correction 10=2 bit error correction 11=5 bit error correction

#define R15_RDS_STA_EN 5

RDS statistic data enable signal

#define R1_DEEMP_SETTING 4

De-emphasis 1=75us 0=50us

#define R1_FORCE_MONO 3

Force mono

#define R1_HARD_MUTE_ENABLE 1

Hard mute enable

#define R1_RDS_ENABLE 13

Enable RDS signal

#define R1_RDS_INT_ENABLE 6

Enable RDS interrupt

#define R1_SOFT_MUTE_ENABLE 2

Soft mute enable

#define R1_STC_INT_ENABLE 5

Seek Tune Complete (STC) interrupt enable

#define R2_CHAN_0 0

#define R2_CHAN_1 1

#define R2_CHAN_2 2

#define R2_CHAN_3 3

#define R2_CHAN_4 4

#define R2_CHAN_5 5

#define R2_CHAN_6 6

#define R2_CHAN_7 7

#define R2_CHAN_8 8

#define R2_TUNE_ENABLE 9

Tune channel enable

#define R3_BAND_0 11

Band control 0=Japan narrow band 76Mhz - 90Mhz 1=Japan wide band 76Mhz - 108Mhz

#define R3_BAND_1 12

Band control 0=US/EUROPE 1=Japan

#define R3_SEEK_CHANNEL_SPACING 13

Seek channel spacing 1=100k 0=200k

#define R3_SEEK_ENABLE 14

Seek enable

#define R3_SEEK_UP 15

Seek direction 1=up 0=down

#define R3_SEEKTH_0 0

#define R3_SEEKTH_1 1

#define R3_SEEKTH_2 2

#define R3_SEEKTH_3 3

#define R3_SEEKTH_4 4

#define R3_SEEKTH_5 5

#define R3_SEEKTH_6 6

#define R3_VOL_0 7

#define R3_VOL_1 8

#define R3_VOL_2 9

#define R3_VOL_3 10

#define STATUS_BIT_2 2

Used but unknown function

#define STATUS_CHAN_0 7

Current tuned channel bit 0

#define STATUS_CHAN_1 8

Current tuned channel bit 1

#define STATUS_CHAN_2 9

Current tuned channel bit 2

#define STATUS_CHAN_3 10

Current tuned channel bit 3

#define STATUS_CHAN_4 11

Current tuned channel bit 4

#define STATUS_CHAN_5 12

Current tuned channel bit 5

#define STATUS_CHAN_6 13

Current tuned channel bit 6

#define STATUS_CHAN_7 14

Current tuned channel bit 7

#define STATUS_CHAN_8 15

Current tuned channel bit 8

#define STATUS_RDS_DATA_READY 6

RDS data recevied

#define STATUS_SEEK_FAIL 4

Seek has failed

#define STATUS_SEEK_TUNE_COMPLETE 5

Seek or Tune has completed

#define STATUS_STEREO 3

Stereo flag 1=stereo 0=mono

Function Documentation

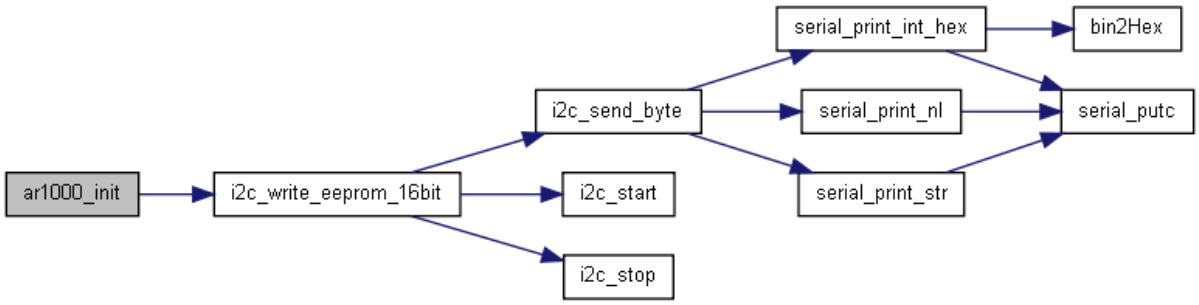
uns8 ar1000_get_register (uns8 *reg*)

```
56 {
57     return regs[reg];
58 }
```

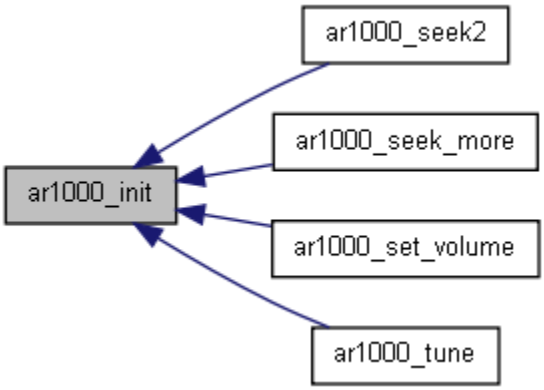
void ar1000_init ()

```
224                     {
225     ar1000_seek_threshold = 0;
226     for (uns8 count = 1; count < 18; count ++) {
227         i2c_write_eeprom_16bit(AR1000_DEV_ADDR, count, regs[count]);
228     }
229     i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 0, regs[0]);
230 
231 }
```

Here is the call graph for this function:



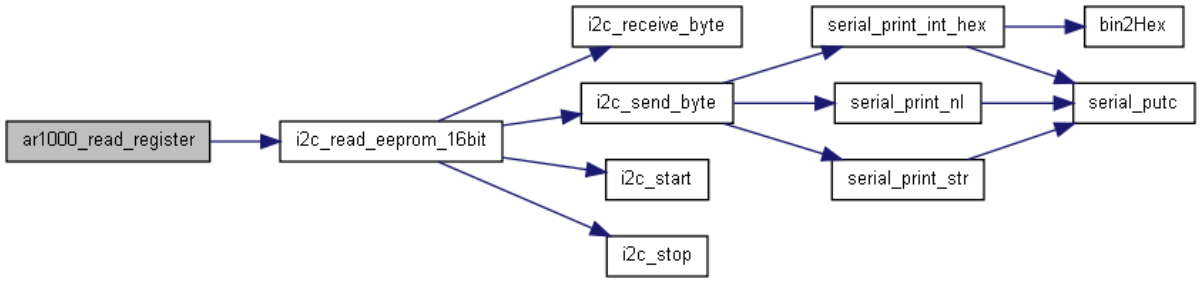
Here is the caller graph for this function:



uns16 ar1000_read_register (uns8 *reg*)

```
71 {
72
73     return i2c_read_eeprom_16bit(AR1000_DEV_ADDR, reg);
74
75 }
```

Here is the call graph for this function:



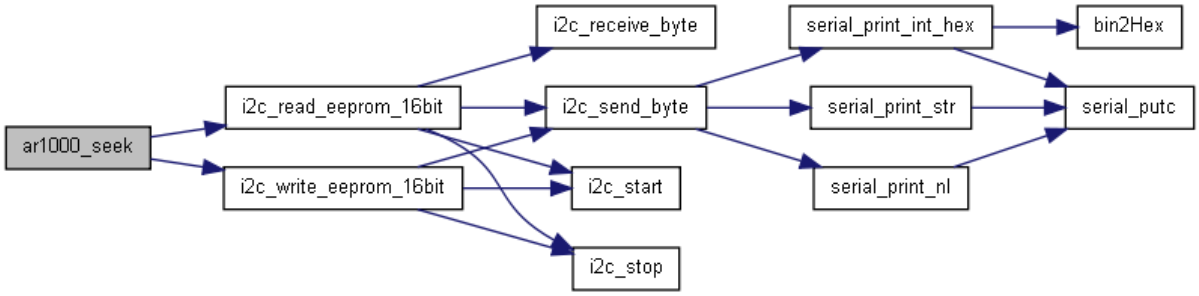
void ar1000_read_registers ()

```
66                                  {
67 }
```

void ar1000_seek (uns16 *frequency*, bit *seek_up*)

```
83                                                  {
84
85 uns16 r1, r2, r3;
86
87 r1 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 1);
88 r2 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 2);
89 r3 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 3);
90
91 // Set hmute bit
92 set_bit(r1, R1_HARD_MUTE_ENABLE);
93
94 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, r1);
95
96 // clear tune bit
97 clear_bit(r2, R2_TUNE_ENABLE);
98
99 // set chan bits r2 to 87.5Mhz
100 // Means setting the bits to 185
101 r2 = r2 & 0b1000000000000000;   // Mask out tune bits
102 //r2 = r2 + (frequency - 690);  // eg 875 - 690 = 185 for 87.5Mhz
103 r2 = r2 & ((frequency - 690) << 7); // eg 875 - 690 = 185 for 87.5Mhz
104
105 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 2, r2);
106
107 // clear seek bit
108 clear_bit(r3, R3_SEEK_ENABLE);
109 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, r3);
110
111 // Set SEEK UP/DOWN / SPACE / BAND / SEEKTH
112 if (seek_up) {
113     set_bit(r3, R3_SEEK_UP);
114 } else {
115     clear_bit(r3, R3_SEEK_UP);
116 }
117 set_bit(r3, R3_SEEK_CHANNEL_SPACING);   // 100k spacing
118 clear_bit(r3, R3_BAND_1);   // US / EUROPE
119
120 r3 = r3 & 0b1111111110000000 + ar1000_seek_threshold;
121 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, r3);
122
123 // Enable SEEK Bit
124 set_bit(r3, R3_SEEK_ENABLE);
125 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, r3);
126
127 // Wait STC flag (Seek/Tune Complete, in "Status" register)
128 // Not done yet! not sure which register it is...
129
130 // Clear hmute Bit
131 clear_bit(r1, R1_HARD_MUTE_ENABLE);
132 i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, r1);
133 //  register_values[02] = 0xB480;  //set tune to 900kHz
134 //  register_values[03] = 0xA001;  //turn off seek, seek up, set threshold to 1
135
136 //  ar1000calibration(register_values);
137
138 //  register_values[03] = 0xE001; //turns on seek
139
140 //  ar1000calibration(register_values);
141
142
143 }
```

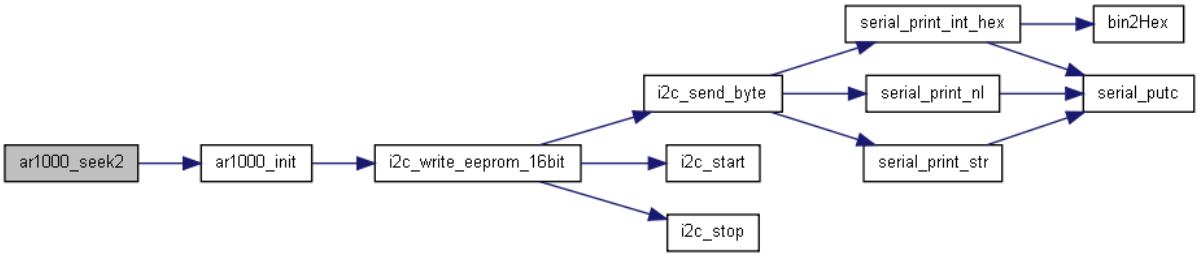
Here is the call graph for this function:



void ar1000_seek2 ()

```
234                     {
235 //    i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 2, 0xb480);
236  //   i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, 0xa001);
237   //  i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, 0xe001);
238   //0xF4B9
239     regs[2] = 0xcd80; //0xB480;//  0xf4b9; //0b1111111000000000;
240     //ar1000_init();
241     regs[3] = 0xa001;
242     ar1000_init();
243     regs[3] = 0xe001;
244     ar1000_init();
245     //  register_values[02] = 0xB480;  //set tune to 900kHz
246 //  register_values[03] = 0xA001;  //turn off seek, seek up, set threshold to 1
247
248 //  ar1000calibration(register_values);
249
250 //  register_values[03] = 0xE001; //turns on seek
251
252 //  ar1000calibration(register_values);
253 }
```

Here is the call graph for this function:

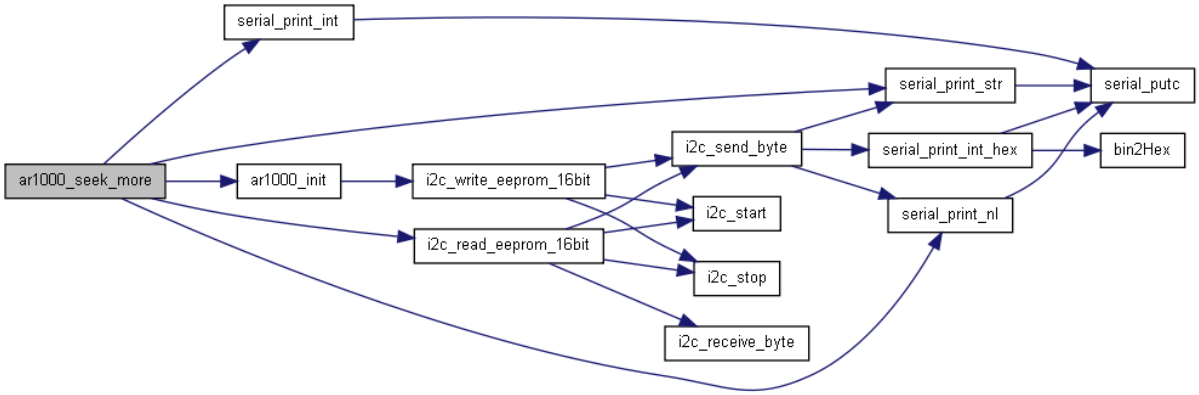


void ar1000_seek_more ()

```
273                         {
274 //    i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 2, 0xb480);
275  //   i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, 0xa001);
276   //  i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 3, 0xe001);
277   //0xF4B9
278 //    regs[2] = 0; //0xB480;//  0xf4b9; //0b1111111000000000;
279     set_bit(regs[2], R2_TUNE_ENABLE);
280     uns16 channel = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, AR1000_STATUS);
281     channel = channel >> 7;
282     serial_print_str(" fr=");
283     serial_print_int(channel+690);
284     serial_print_nl();
285     //channel = channel << 6;
```

```
286     regs[2] = channel;
287     set_bit(regs[2], R2_TUNE_ENABLE);
288     //ar1000_init();
289     regs[3] = 0xa005;
290     ar1000_init();
291     regs[3] = 0xe005;
292     ar1000_init();
293 }
```

Here is the call graph for this function:



void ar1000_set_register (uns8 *reg*, uns8 *data*)

```
51 {
52     regs[reg] = data;
53 }
```

void ar1000_set_seek_threshold (uns8 *new_seek_threshold*)

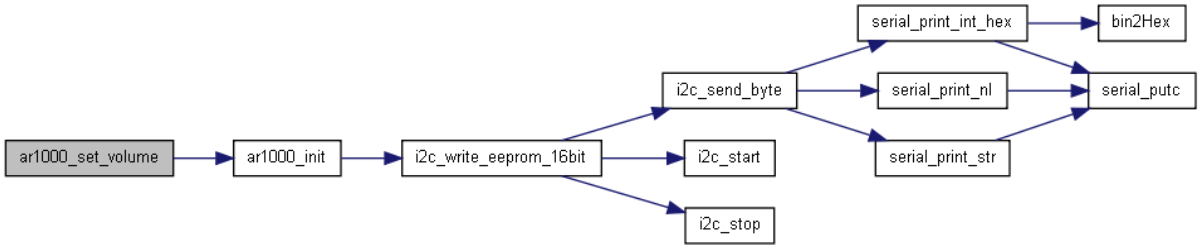
```
220 {
221     ar1000_seek_threshold = new_seek_threshold;
222 }
```

void ar1000_set_volume (uns8 *volume*)

```
321                                     {
322 
323 uns16 reg, temp;
324 uns8  vol;
325 uns16 vol1, vol2;
326 
327     if (volume > 21) { return; }
328     vol = vol_lookup[volume];
329     vol2 = vol >> 4;
330     vol1 = vol & 0x0f;
331 
332 
333     regs[3] = (regs[3] & ~0x0780) | (vol1 << 7);
334     //write(3, register_values[3]);
335 
336     regs[14] = (regs[14] & ~0xF000) | (vol2 << 12);
337     //write(14, register_values[14]);
338 
339     //serial_print_str(" AR3=");
340     //serial_print_int_hex_16bit(reg);
```

```
341     //regs[3] = reg;
342     ar1000_init();
343 }
```

Here is the call graph for this function:



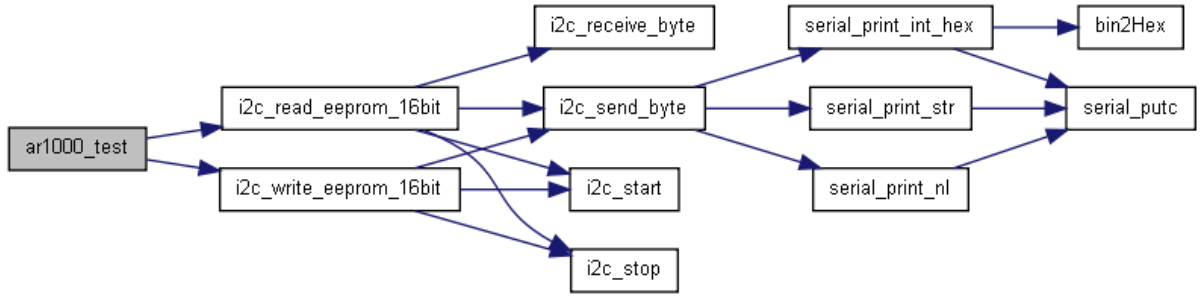
void ar1000_setup_io ()

```
46                          {
47     i2c_setup();
48 }
```

void ar1000_test ()

```
255                     {
256 /*  serial_print_str("Read 01 = ");
257     uns16 r1 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 1);
258     serial_print_int_hex_16bit(r1);
259     serial_print_nl();
260     serial_print_str("Write  ");
261     i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, 0xabcd);
262     serial_print_str("Read 01 = ");
263     r1 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 1);
264     serial_print_int_hex_16bit(r1);
265     serial_print_nl();
266 */
267     uns16 r1 = i2c_read_eeprom_16bit(AR1000_DEV_ADDR, 1);
268     toggle_bit(r1, R1_HARD_MUTE_ENABLE);
269     i2c_write_eeprom_16bit(AR1000_DEV_ADDR, 1, r1);
270
271 }
```

Here is the call graph for this function:

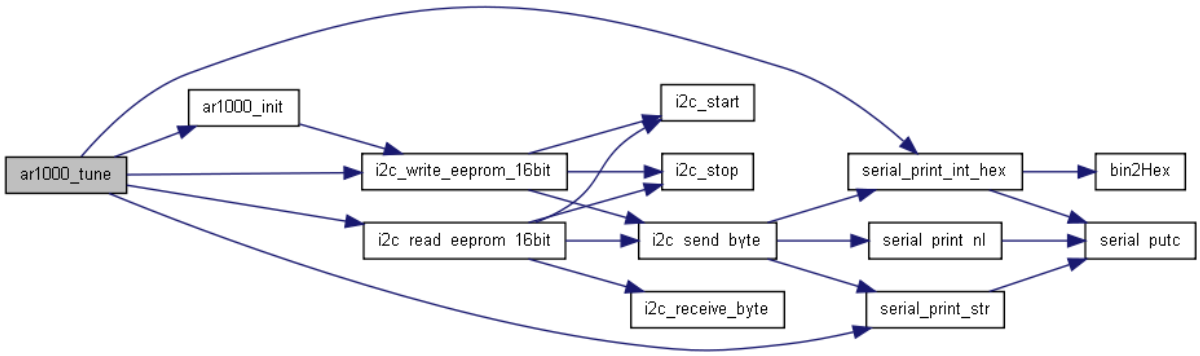


void ar1000_tune (uns16 *frequency*)

```
145                                 {
```

```
146
147 uns16 r1, r2, r3;
148 set bit(regs[3], R3 SEEK CHANNEL SPACING);
149 regs[2] = 0b0000000100111111;
150 ar1000 init();
151 regs[2] = 0b0000001100111111;
152
153 ar1000 init();  //a7e0
154 return;
155 r1 = i2c read eeprom 16bit(AR1000 DEV ADDR, 1);
156 r2 = i2c read eeprom 16bit(AR1000 DEV ADDR, 2);
157 serial print str("r2o=");
158 serial print int hex(r2);
159 r3 = i2c read eeprom 16bit(AR1000 DEV ADDR, 3);
160 //1 100111111 000000
161 // Set hmute bit
162 set_bit(r1, R1 HARD MUTE ENABLE);
163 i2c write eeprom 16bit(AR1000 DEV ADDR, 1, r1);
164
165
166 // clear tune bit
167 clear_bit(r2, R2 TUNE ENABLE);
168 i2c write eeprom 16bit(AR1000 DEV ADDR, 2, r2);
169
170 // clear seek bit
171 clear_bit(r3, R3 SEEK ENABLE);
172 i2c write eeprom 16bit(AR1000 DEV ADDR, 3, r3);
173
174 // Set  SPACE / BAND / CHAN
175 set_bit(r3, R3 SEEK CHANNEL SPACING);   // 100k spacing
176 clear_bit(r3, R3 BAND 1);   // US / EUROPE
177 i2c write eeprom 16bit(AR1000 DEV ADDR, 3, r3);
178
179 // set chan bits r2 to 87.5Mhz
180 // Means setting the bits to 185
181 //serial_print_str("f=");
182 //serial_print_int(frequency);
183 //serial_print_nl();
184 r2 = frequency - 690;
185 //serial print str(" f-690=");
186 //serial_print_int(r2);
187 r2 = r2 << 6;
188 //serial_print_str("tuning to=");
189 //serial print int hex 16bit(r2);
190 i2c write eeprom 16bit(AR1000 DEV ADDR, 2, r2);
191 // 100111100 0000000
192 // Enable TUNE Bit
193 set_bit(r2, R2 TUNE ENABLE);
194 i2c write eeprom 16bit(AR1000 DEV ADDR, 2, r2);
195 //1 001111000 00000
196 //serial_print_str(" r2en=");
197 //serial_print_int_hex_16bit(r2);
198
199 // Wait STC flag (Seek/Tune Complete, in “Status” register)
200 // Not done yet! not sure which register it is...
201 // Clear hmute Bit
202 clear bit(r1, R1 HARD MUTE ENABLE);
203 i2c write eeprom 16bit(AR1000 DEV ADDR, 1, r1);
204 //  register_values[02] = 0xB480;  //set tune to 900kHz
205 //  register_values[03] = 0xA001;  //turn off seek, seek up, set threshold to 1
206
207 //  ar1000calibration(register_values);
208
209 //  register_values[03] = 0xE001; //turns on seek
210
211 //  ar1000calibration(register_values);
212
213
214
215
216 }
```

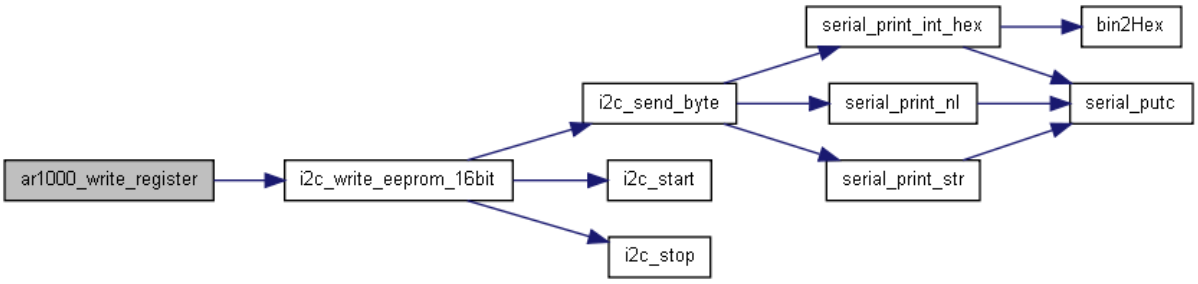
Here is the call graph for this function:



void ar1000_write_register (uns8 *reg*, uns16 *data*)

```
77                                                  {
78
79      i2c_write_eeprom_16bit(AR1000_DEV_ADDR, reg, data);
80
81  }
```

Here is the call graph for this function:



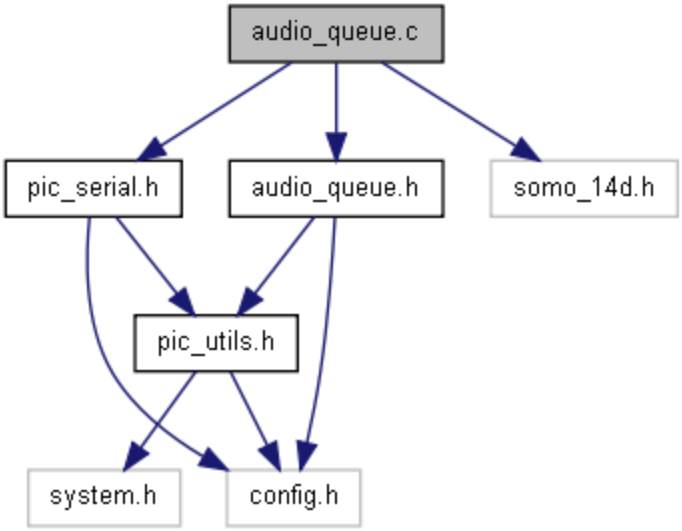
void ar1000_write_registers ()

```
60                                  {
61  uns8 count;
62
63  }
```

---

audio_queue.c File Reference

Include dependency graph for audio_queue.c:



Functions

- void audio_queue_add (uns8 phrase)
- void audio_queue_clear ()
- uns8 audio_queue_empty ()
- void audio_queue_process ()

Variables

- uns8 aq_end = 0
- uns8 aq_start = 0
- bit audio_playing = 0
- uns8 audio_queue_fifo [AUDIO_QUEUE_FIFO_SIZE]

Function Documentation

void audio_queue_add (uns8 *phrase*)

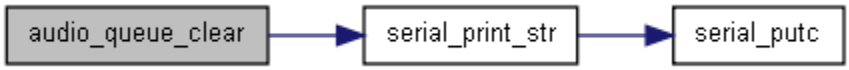
```
51                                  {
52
53 uns8 aq_next;
54
55     if ((aq_end == aq_start) &&  // Nothing in the fifo
56         (audio_playing == 0)) {  // And txreg is empty
57
58         somo_14d_set_file_id(phrase);
59         audio_playing =  1;
60     } else {    // Put it in the buffer
61         aq_next = aq_end + 1;   // Get next buffer position
62         if (aq_next == AUDIO_QUEUE_FIFO_SIZE) { // If we're at the end
63             aq_next = 0;    // wrap to the beginning
64         }
65         if (aq_next == aq_start) {
66             // we're full!
67             return; // just forget it for now
68         }
69
70         audio_queue_fifo[aq_end] = phrase; // put it in
71         aq_end = aq_next;  // move pointer along
```

```
72     }
73 }
```

void audio_queue_clear ()

```
105                             {
106     // crit sec
107     start_crit_sec();
108     aq_end = aq_start;
109     if (audio_playing || somo_14d_is_busy()) {
110         serial_print_str("<<BUSY>>");
111         delay_ms(10);
112         somo_14d_stop();
113         while (somo_14d_is_busy()) {};
114         delay_ms(10);
115     }
116     audio_playing = 0;
117     end_crit_sec();
118     // crit sec end
119 }
```

Here is the call graph for this function:



uns8 audio_queue_empty ()

```
121                             {
122     return (audio_playing == 0);
```

123 }

void audio_queue_process ()

Call when audio file completes.

This routine will pluck the next file off the queue and start playing it. Assumes it is in an interrupt otherwise will need wrapping in critsec

```
82 {
83 uns8 aq_next;
84
85     if (aq_end == aq_start) { // anything in the fifo?
86         //serial_print_str("End of queue\n");
87         audio_playing = 0;
88        return; // nope
89     }
90     aq_next = aq_start + 1; // get next position
91     if (aq_next == AUDIO_QUEUE_FIFO_SIZE) { // if we're at the end of the buffer
92         aq_next = 0;    // wrap to the beginning
93     }
94     if (aq_end == aq_next) {    // if we've only got one character to send
95         //?????clear_bit(pie1, TXIE);   // then turn off interrupts
96     }
97     audio_playing = 1;
98
99     somo_14d_set_file_id(audio_queue_fifo[aq_start]);
100     aq_start = aq_next; // move start position of fifo
101
102 }
```

Variable Documentation

uns8 aq_end = 0

Audio queue fifo end point

uns8 aq_start = 0

Audio queue fifo start point

bit audio_playing = 0

Audio playing at present

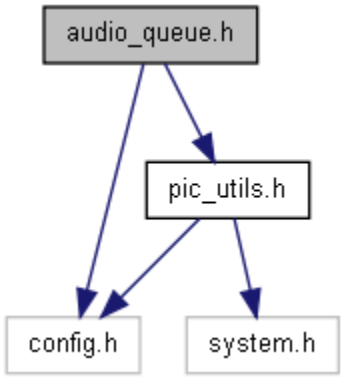
uns8 audio_queue_fifo[AUDIO_QUEUE_FIFO_SIZE]

Audio queue fifo

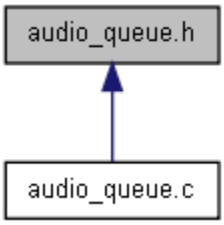
---

audio_queue.h File Reference

Queue audio files for the somo-14d.
Include dependency graph for audio_queue.h:



This graph shows which files directly or indirectly include this file:



Functions

- void audio_queue_add (uns8 phrase)
- void audio_queue_clear ()
- uns8 audio_queue_empty ()
- void audio_queue_process ()

Detailed Description

Put the following into your config.h

- ---------- audio_queue defines
- ----------

define AUDIO_QUEUE_FIFO_SIZE 5

Function Documentation

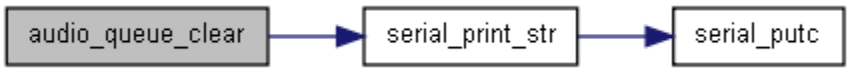
void audio_queue_add (uns8 *phrase*)

```
51                                  {
52
53 uns8 aq_next;
54
55     if ((aq_end == aq_start) &&  // Nothing in the fifo
56         (audio_playing == 0)) {  // And txreg is empty
57
58         somo_14d_set_file_id(phrase);
59         audio_playing =  1;
60     } else {    // Put it in the buffer
61         aq_next = aq_end + 1;   // Get next buffer position
62         if (aq_next == AUDIO_QUEUE_FIFO_SIZE) { // If we're at the end
63             aq_next = 0;    // wrap to the beginning
64         }
65         if (aq_next == aq_start) {
66             // we're full!
67             return; // just forget it for now
68         }
69
70         audio_queue_fifo[aq_end] = phrase; // put it in
71         aq_end = aq_next;  // move pointer along
72     }
73 }
```

void audio_queue_clear ()

```
105                             {
106     // crit_sec
107     start_crit_sec();
108     aq_end = aq_start;
109     if (audio_playing || somo_14d_is_busy()) {
110         serial_print_str("<<BUSY>>");
111         delay_ms(10);
112         somo_14d_stop();
113         while (somo_14d_is_busy()) {};
114         delay_ms(10);
115     }
116     audio_playing = 0;
117     end_crit_sec();
118     // crit_sec_end
119 }
```

Here is the call graph for this function:



uns8 audio_queue_empty ()

```
121                             {
122     return (audio_playing == 0);
123 }
```

void audio_queue_process ()

Call when audio file completes.

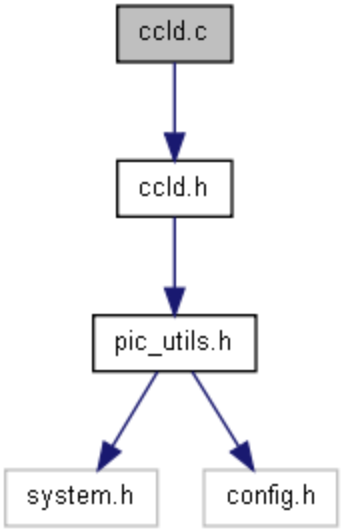
This routine will pluck the next file off the queue and start playing it. Assumes it is in an interrupt otherwise will need wrapping in critsec

```
82 {
83 uns8 aq_next;
84 
85     if (aq_end == aq_start) { // anything in the fifo?
86         //serial_print_str("End of queue\n");
87         audio_playing = 0;
88        return; // nope
89     }
90     aq_next = aq_start + 1; // get next position
91     if (aq_next == AUDIO_QUEUE_FIFO_SIZE) { // if we're at the end of the buffer
92         aq_next = 0;    // wrap to the beginning
93     }
94     if (aq_end == aq_next) {    // if we've only got one character to send
95         //?????clear_bit(pie1, TXIE);   // then turn off interrupts
96     }
97     audio_playing = 1;
98 
99     somo_14d_set_file_id(audio_queue_fifo[aq_start]);
100     aq_start = aq_next; // move start position of fifo
101 
102 }
```

---

ccld.c File Reference

Include dependency graph for ccld.c:



Functions

- void ccld_enable_display (uns8 on)
- *Enable display on ccld.* void ccld_latch_data ()
- void ccld_setup_io ()
- *Setup IO to communicate with ccld chip.* void ccld_tlc_normal_mode (void)
- void ccld_tlc_special_mode (void)
- void ccld_write_data (uns16 d)
- *Send 16 bits to ccld.* void ccld_write_data_byte (uns8 d)

Send 8 bits of data to ccld chip.

Function Documentation

void ccld_enable_display (uns8 *on*)

Enable the display

Parameters:

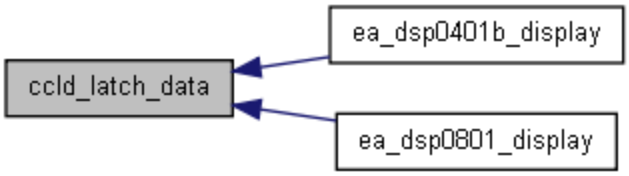
on 0 - display off, 1 - display 1

```
103                                     {
104
105 #ifdef ccld_blk_port
106
107     if (on){
108         clear_pin(ccld_blk_port, ccld_blk_pin);
109     } else {
110         set_pin(ccld_blk_port, ccld_blk_pin);
111     }
112 #endif
113 }
```

void ccld_latch_data (void)

```
50                          {
51
52     set_pin(ccld_lat_port, ccld_lat_pin);
53     clear_pin(ccld_lat_port, ccld_lat_pin);
54
55 }
```

Here is the caller graph for this function:



void ccld_setup_io ()

Setup IO to communicate with ccld chips

```
57                      {
58
59      clear pin(ccld clk port, ccld clk pin);
60      clear pin(ccld_lat_port, ccld_lat_pin);
61  #ifdef ccld_blk_port
62      set pin(ccld_blk_port, ccld_blk_pin);
63  #endif
64      make output(ccld_sin_port, ccld_sin_pin);
65      make output(ccld_clk_port, ccld_clk_pin);
66  #ifdef ccld_blk_port
67      make output(ccld blk port, ccld blk pin);
68  #endif
69      make output(ccld_lat_port, ccld_lat_pin);
70
71  }
```

Here is the caller graph for this function:

void ccld_tlc_normal_mode (void)

```
159                                 {
160     #ifdef ccld_blk_port
161         set pin(ccld_blk_port, ccld_blk_pin);
162     #endif
163     // pulse clk
164     set pin(ccld clk port, ccld clk pin);
165     clear pin(ccld_clk_port, ccld_clk_pin);
166
167     // set OE low
168     #ifdef ccld_blk_port
169         clear pin(ccld_blk_port, ccld_blk_pin);
170     #endif
171     // pulse clk
172     set pin(ccld clk port, ccld clk pin);
173     clear pin(ccld_clk_port, ccld_clk_pin);
174
175     // set OE high
176     #ifdef ccld blk port
177         set pin(ccld_blk_port, ccld_blk_pin);
178     #endif
179
180     // pulse clk
181     set pin(ccld_clk_port, ccld_clk_pin);
182     clear pin(ccld_clk_port, ccld_clk_pin);
183
184     // pulse clk
185     set pin(ccld_clk_port, ccld_clk_pin);
```

```
186     clear_pin(ccld_clk_port, ccld_clk_pin);
187
188     // pulse clk
189     set_pin(ccld_clk_port, ccld_clk_pin);
190     clear_pin(ccld_clk_port, ccld_clk_pin);
191
192
193 }
```

void ccld_tlc_special_mode (void)

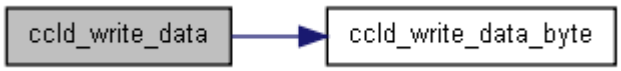
```
118                                 {
119
120     #ifdef ccld_blk_port
121         set_pin(ccld_blk_port, ccld_blk_pin);
122     #endif
123     // pulse clk
124     set_pin(ccld_clk_port, ccld_clk_pin);
125     clear_pin(ccld_clk_port, ccld_clk_pin);
126
127     // set OE low
128     #ifdef ccld_blk_port
129         clear_pin(ccld_blk_port, ccld_blk_pin);
130     #endif
131     // pulse clk
132     set_pin(ccld_clk_port, ccld_clk_pin);
133     clear_pin(ccld_clk_port, ccld_clk_pin);
134
135     // set OE high
136     #ifdef ccld_blk_port
137         set_pin(ccld_blk_port, ccld_blk_pin);
138     #endif
139     // pulse clk
140     set_pin(ccld_clk_port, ccld_clk_pin);
141     clear_pin(ccld_clk_port, ccld_clk_pin);
142
143     // now set latch high
144     set_pin(ccld_lat_port, ccld_lat_pin);
145
146     // pulse clk
147     set_pin(ccld_clk_port, ccld_clk_pin);
148     clear_pin(ccld_clk_port, ccld_clk_pin);
149
150     // latch low
151     clear_pin(ccld_lat_port, ccld_lat_pin);
152
153     // pulse clk
154     set_pin(ccld_clk_port, ccld_clk_pin);
155     clear_pin(ccld_clk_port, ccld_clk_pin);
156 }
```

void ccld_write_data (uns16 *d*)

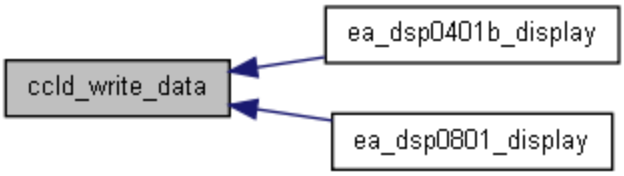
Send d to ccld

```
74                                  {
75
76     ccld_write_data_byte(d >> 8);
77     ccld_write_data_byte(d & 0xff);
78
79 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

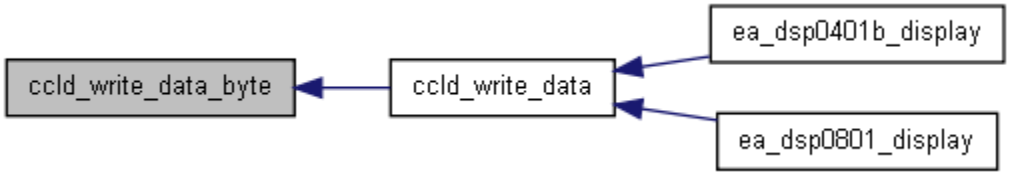


void ccld_write_data_byte (uns8 *d*)

Send d1 to ccld chip

```
82                                      {
83
84 uns8 count;
85
86     for (count=0; count<8; count++) {
87
88         // set data
89         change pin var(ccld_sin_port, ccld_sin_pin, d.7);
90
91         // pulse clk
92
93         set pin(ccld_clk_port, ccld_clk_pin);
94         d <<= 1;
95
96
97         clear pin(ccld clk port, ccld clk pin);
98
99     }
100     delay_us(100);
101 }
```

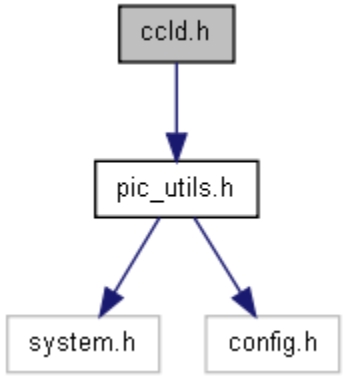
Here is the caller graph for this function:



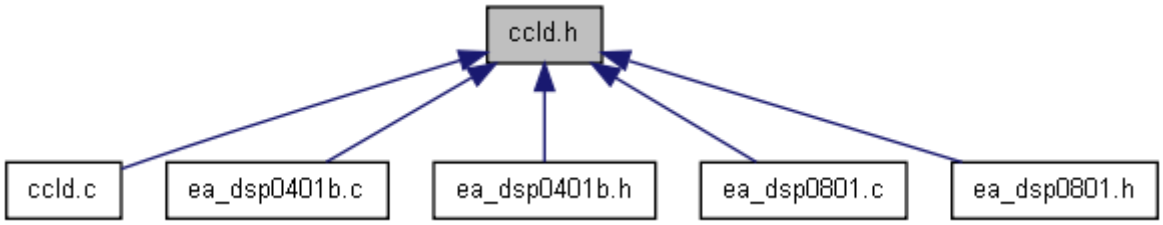
---

ccld.h File Reference

Include dependency graph for ccld.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define __ccld_H

Functions

- void ccld_enable_display (uns8 on)
- *Enable display on ccld.* void ccld_latch_data (void)
- void ccld_setup_io ()
- *Setup IO to communicate with ccld chip.* void ccld_tlc_normal_mode (void)
- void ccld_tlc_special_mode (void)
- void ccld_write_data (uns16 d)
- *Send 16 bits to ccld.* void ccld_write_data_byte (uns8 d)

Send 8 bits of data to ccld chip.

Define Documentation

#define __ccld_H

Function Documentation

void ccld_enable_display (uns8 *on*)

Enable the display

Parameters:

on 0 - display off, 1 - display 1

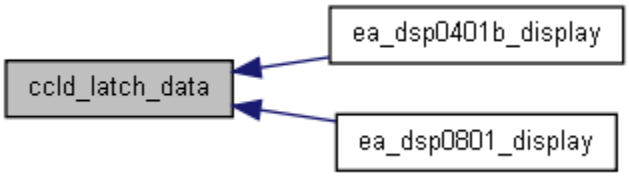
```
103                                 {
104
105 #ifdef ccld_blk_port
106
```

```
107     if (on){
108         clear pin(ccld_blk_port, ccld_blk_pin);
109     } else {
110         set pin(ccld_blk_port, ccld_blk_pin);
111     }
112 #endif
113 }
```

void ccld_latch_data (void)

```
50                          {
51
52     set pin(ccld_lat_port, ccld_lat_pin);
53     clear pin(ccld lat port, ccld lat pin);
54
55 }
```

Here is the caller graph for this function:



void ccld_setup_io ()

Setup IO to communicate with ccld chips

```
57                      {
58
59     clear pin(ccld_clk_port, ccld_clk_pin);
60     clear pin(ccld_lat_port, ccld_lat_pin);
61 #ifdef ccld_blk_port
62     set pin(ccld_blk_port, ccld_blk_pin);
63 #endif
64     make output(ccld sin port, ccld sin pin);
65     make output(ccld_clk_port, ccld_clk_pin);
66 #ifdef ccld_blk_port
67     make output(ccld_blk_port, ccld_blk_pin);
68 #endif
69     make output(ccld_lat_port, ccld_lat_pin);
70
71 }
```

Here is the caller graph for this function:

void ccld_tlc_normal_mode (void)

```
159                                 {
160     #ifdef ccld_blk_port
161         set pin(ccld_blk_port, ccld_blk_pin);
162     #endif
163     // pulse clk
164     set pin(ccld clk port, ccld clk pin);
165     clear pin(ccld_clk_port, ccld_clk_pin);
166
```

```
167     // set OE low
168     #ifdef ccld_blk_port
169         clear pin(ccld blk port, ccld blk pin);
170     #endif
171     // pulse clk
172     set pin(ccld_clk_port, ccld_clk_pin);
173     clear pin(ccld_clk_port, ccld_clk_pin);
174
175     // set OE high
176     #ifdef ccld_blk_port
177         set pin(ccld blk port, ccld blk pin);
178     #endif
179
180     // pulse clk
181     set pin(ccld clk port, ccld clk pin);
182     clear pin(ccld_clk_port, ccld_clk_pin);
183
184     // pulse clk
185     set pin(ccld_clk_port, ccld_clk_pin);
186     clear pin(ccld_clk_port, ccld_clk_pin);
187
188     // pulse clk
189     set pin(ccld clk port, ccld clk pin);
190     clear pin(ccld_clk_port, ccld_clk_pin);
191
192
193 }
```

void ccld_tlc_special_mode (void)

```
118                                 {
119
120     #ifdef ccld_blk_port
121         set pin(ccld_blk_port, ccld_blk_pin);
122     #endif
123     // pulse clk
124     set pin(ccld_clk_port, ccld_clk_pin);
125     clear pin(ccld_clk_port, ccld_clk_pin);
126
127     // set OE low
128     #ifdef ccld_blk_port
129         clear pin(ccld_blk_port, ccld_blk_pin);
130     #endif
131     // pulse clk
132     set pin(ccld_clk_port, ccld_clk_pin);
133     clear pin(ccld_clk_port, ccld_clk_pin);
134
135     // set OE high
136     #ifdef ccld_blk_port
137         set pin(ccld_blk_port, ccld_blk_pin);
138     #endif
139     // pulse clk
140     set pin(ccld_clk_port, ccld_clk_pin);
141     clear pin(ccld_clk_port, ccld_clk_pin);
142
143     // now set latch high
144     set pin(ccld_lat_port, ccld_lat_pin);
145
146     // pulse clk
147     set pin(ccld clk port, ccld clk pin);
148     clear pin(ccld_clk_port, ccld_clk_pin);
149
150     // latch low
151     clear pin(ccld lat port, ccld lat pin);
152
153     // pulse clk
154     set pin(ccld_clk_port, ccld_clk_pin);
155     clear pin(ccld_clk_port, ccld_clk_pin);
```

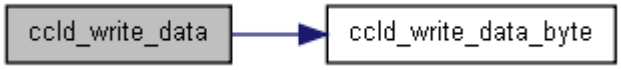
```
156 }
```

void ccld_write_data (uns16 *d*)

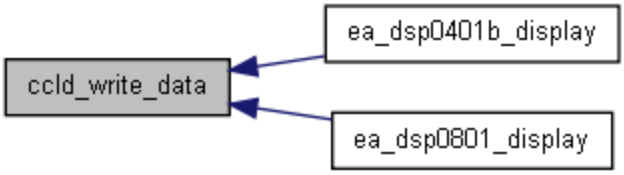
Send d to ccld

```
74                                  {
75
76      ccld_write_data_byte(d >> 8);
77      ccld_write_data_byte(d & 0xff);
78
79  }
```

Here is the call graph for this function:



Here is the caller graph for this function:

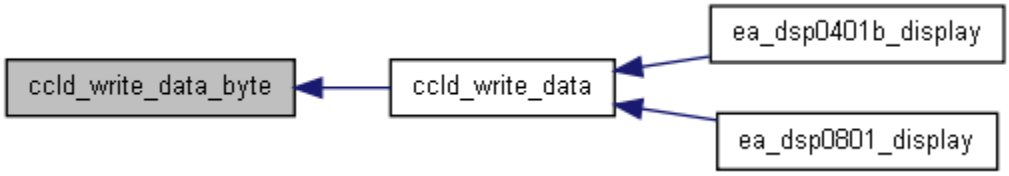


void ccld_write_data_byte (uns8 *d*)

Send d1 to ccld chip

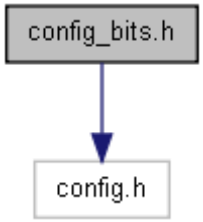
```
82                                      {
83
84  uns8 count;
85
86      for (count=0; count<8; count++) {
87
88          // set data
89          change_pin_var(ccld_sin_port, ccld_sin_pin, d.7);
90
91          // pulse clk
92
93          set_pin(ccld_clk_port, ccld_clk_pin);
94          d <<= 1;
95
96
97          clear_pin(ccld_clk_port, ccld_clk_pin);
98
99      }
100      delay_us(100);
101 }
```

Here is the caller graph for this function:



config_bits.h File Reference

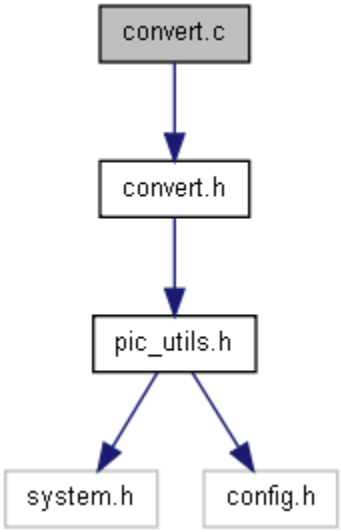
Trying the impossible task of creating some commonality around the config settings.
Include dependency graph for config_bits.h:



Detailed Description

convert.c File Reference

Include dependency graph for convert.c:



Functions

- uns8 convert_to_dec1 (uns8 data)
- uns8 convert_to_dec2 (uns8 data)
- uns8 convert_to_dec2b (uns8 data)
- char * temp_to_str (uns8 int_part, uns8 fract_part, char *buffer)

Function Documentation

uns8 convert_to_dec1 (uns8 *data*)

```
80                                  {
81
82      switch (data) {
83          case 0: return 0;
84          case 1: return 1;
85          case 2: return 1;
86          case 3: return 2;
87          case 4: return 3;
88          case 5: return 3;
89          case 6: return 4;
90          case 7: return 4;
91          case 8: return 5;
92          case 9: return 6;
93          case 10: return 6;
94          case 11: return 7;
95          case 12: return 8;
96          case 13: return 8;
97          case 14: return 9;
98          case 15: return 9;
99      }
100 }
```

uns8 convert_to_dec2 (uns8 *data*)

```
101                                  {
102
103     switch (data) {
104         case 0: return 00;
105         case 1: return 06;
106         case 2: return 13;
107         case 3: return 19;
108         case 4: return 25;
109         case 5: return 31;
110         case 6: return 38;
111         case 7: return 44;
112         case 8: return 50;
113         case 9: return 56;
114         case 10: return 63;
115         case 11: return 69;
116         case 12: return 75;
117         case 13: return 81;
118         case 14: return 88;
119         case 15: return 94;
120     }
121 }
```

uns8 convert_to_dec2b (uns8 *data*)

```
58                                  {
59
60      switch (data) {
61          case 0: return 00;
62          case 1: return 00;
63          case 2: return 00;
64          case 3: return 25;
65          case 4: return 25;
66          case 5: return 25;
67          case 6: return 50;
68          case 7: return 50;
```

```
69          case 8: return 50;
70          case 9: return 50;
71          case 10: return 50;
72          case 11: return 75;
73          case 12: return 75;
74          case 13: return 75;
75          case 14: return 99;
76          case 15: return 99;
77      }
78 }
```

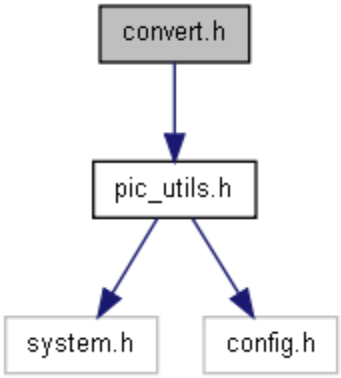
char* temp_to_str (uns8 *int_part*, uns8 *fract_part*, char * *buffer*)

```
124                                                                 {
125
126     buffer[5] = 0;
127     if (int_part>9) {
128         buffer[0] = '0' + int part / 10;
129         buffer[1] = '0' + int_part % 10;
130     } else {
131         buffer[0] = ' ';
132         buffer[1] = '0' + int_part;
133     }
134     buffer[2] = '.';
135     if (fract_part>9) {
136         buffer[3] = '0' + fract part / 10;
137         buffer[4] = '0' + fract_part % 10;
138     } else {
139         buffer[3] = '0';
140         buffer[4] = '0' + fract_part;
141     }
142     return buffer;
143 }
```

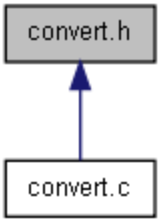
---

convert.h File Reference

Convert pseudo-decimal to strings (typically temperature conversion).
Include dependency graph for convert.h:



This graph shows which files directly or indirectly include this file:



Functions

- uns8 convert_to_dec1 (uns8 data)
- uns8 convert_to_dec2 (uns8 data)
- uns8 convert_to_dec2b (uns8 data)
- char * temp_to_str (uns8 int_part, uns8 fract_part, char *buffer)

Detailed Description

Function Documentation

uns8 convert_to_dec1 (uns8 *data*)

```
80                                  {
81
82      switch (data) {
83          case 0: return 0;
84          case 1: return 1;
85          case 2: return 1;
86          case 3: return 2;
87          case 4: return 3;
88          case 5: return 3;
89          case 6: return 4;
90          case 7: return 4;
91          case 8: return 5;
92          case 9: return 6;
93          case 10: return 6;
94          case 11: return 7;
95          case 12: return 8;
96          case 13: return 8;
97          case 14: return 9;
98          case 15: return 9;
99      }
100 }
```

uns8 convert_to_dec2 (uns8 *data*)

```
101                                  {
102
103     switch (data) {
104         case 0: return 00;
105         case 1: return 06;
106         case 2: return 13;
107         case 3: return 19;
108         case 4: return 25;
109         case 5: return 31;
```

```
110         case 6: return 38;
111         case 7: return 44;
112         case 8: return 50;
113         case 9: return 56;
114         case 10: return 63;
115         case 11: return 69;
116         case 12: return 75;
117         case 13: return 81;
118         case 14: return 88;
119         case 15: return 94;
120     }
121 }
```

uns8 convert_to_dec2b (uns8 *data*)

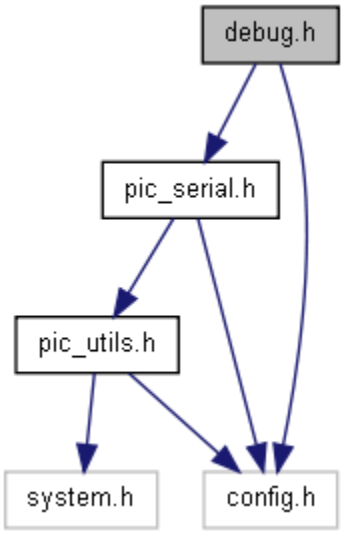
```
58                                  {
59
60     switch (data) {
61         case 0: return 00;
62         case 1: return 00;
63         case 2: return 00;
64         case 3: return 25;
65         case 4: return 25;
66         case 5: return 25;
67         case 6: return 50;
68         case 7: return 50;
69         case 8: return 50;
70         case 9: return 50;
71         case 10: return 50;
72         case 11: return 75;
73         case 12: return 75;
74         case 13: return 75;
75         case 14: return 99;
76         case 15: return 99;
77     }
78 }
```

char* temp_to_str (uns8 *int_part*, uns8 *fract_part*, char * *buffer*)

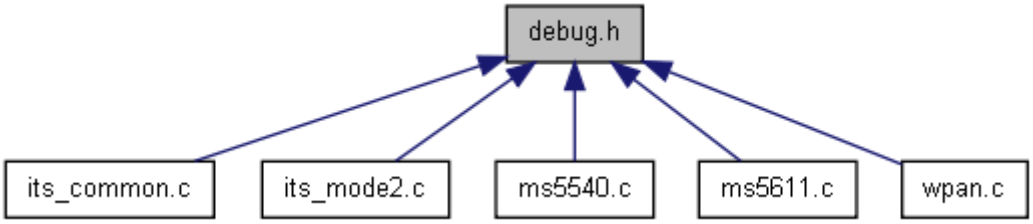
```
124                                                                 {
125
126     buffer[5] = 0;
127     if (int_part>9) {
128         buffer[0] = '0' + int_part / 10;
129         buffer[1] = '0' + int_part % 10;
130     } else {
131         buffer[0] = ' ';
132         buffer[1] = '0' + int_part;
133     }
134     buffer[2] = '.';
135     if (fract_part>9) {
136         buffer[3] = '0' + fract_part / 10;
137         buffer[4] = '0' + fract_part % 10;
138     } else {
139         buffer[3] = '0';
140         buffer[4] = '0' + fract_part;
141     }
142     return buffer;
143 }
```

debug.h File Reference

A nice way of printing debug, allowing it to be compiled out for production.
Include dependency graph for debug.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define debug_int(i)
- #define debug_int_hex(i)
- #define debug_int_hex_16bit(i)
- #define debug_nl()
- #define debug_putc(c)
- #define debug_spc()
- #define debug_str(str)
- #define debug_var(str, i)

Detailed Description

Define Documentation

#define debug_int(i)

#define debug_int_hex(i)

#define debug_int_hex_16bit(i)

#define debug_nl()

#define debug_putc(c)

#define debug_spc()

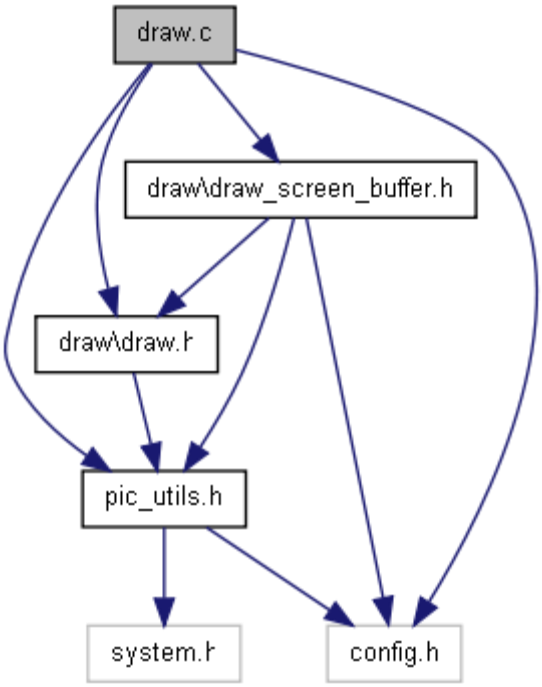
#define debug_str(str)

#define debug_var(str, i)

---

draw.c File Reference

Buffered graphics routines.
Include dependency graph for draw.c:



Defines

- #define FONT_FIRST_CHAR 32
- #define FONT_HEIGHT 7

- #define FONT_LAST_CHAR 127

Functions

- void draw_bitmap (uns8 x, uns8 y, uns8 colour, char *bitmap)
- void draw_circle (int x_centre, int y_centre, int r, uns8 colour)
- void draw_circle2 (int x_centre, int y_centre, int r, uns8 colour)
- void draw_circle_lines (int ctr_x, int ctr_y, int pt_x, int pt_y, uns8 colour)
- void draw_circle_points (int ctr_x, int ctr_y, int pt_x, int pt_y, uns8 colour)
- void draw_circle_points2 (int ctr_x, int ctr_y, int pt_x, int pt_y, uns8 colour)
- void draw_clear_screen ()
- void draw_filled_circle (int x_centre, int y_centre, int r, uns8 colour)
- uns8 draw_get_pixel (uns8 x, uns8 y)
- void draw_init ()
- uns16 draw_length_str (char *str)
- void draw_line (uns8 x0, uns8 y0, uns8 x1, uns8 y1, uns8 colour)
- void draw_print_buffer ()
- void draw_print_str (uns8 x, uns8 y, uns8 width, uns8 start_pixel, uns8 colour, char *str)
- void draw_rect (uns8 x, uns8 y, uns16 width, uns8 height, uns8 colour)
- void draw_set_pixel (uns8 x, uns8 y, uns8 colour)
- void draw_setup_io ()

Variables

- rom char PicPack5x7_bitmap_0 [1]
- rom char PicPack5x7_bitmap_1 [1]
- uns16 PicPack5x7_index [1]

Detailed Description

Define Documentation

#define FONT_FIRST_CHAR 32

#define FONT_HEIGHT 7

#define FONT_LAST_CHAR 127

Function Documentation

void draw_bitmap (uns8 *x*, uns8 *y*, uns8 *colour*, char * *bitmap*)

```
593                                                                 {
594
595
596 uns8 bitpos = 0;
597 uns8 bytepos = 0;
598 uns8 value;
599 uns8 bitmap_width = bitmap[0];
600 uns8 bitmap_height = bitmap[1];
```

```
601 uns8 bitmap_bpp = bitmap[2];
602 uns8 xbitmap, ybitmap;
603
604
605     bytepos = 3;     // first byte of data - 1
606     bitpos = 0b10000000;     // left most bit
607
608     for (xbitmap = 0; xbitmap < bitmap_width; xbitmap++) {
609         for (ybitmap = 0; ybitmap < bitmap_height; ybitmap++) {
610             if (bitpos == 0b10000000) {
611                 bitpos = 0b00000001;
612                 bytepos++;
613                 value = bitmap[bytepos];
614             } else {
615                 bitpos = bitpos << 1;
616             }
617             if (value & bitpos) {
618                 #if DRAW_HW_Y_ORIGIN == TOP_LEFT
619                     draw_set_pixel(x + xbitmap, y + ybitmap, colour);
620                 #else
621                     draw_set_pixel(x + xbitmap, y - ybitmap, colour);
622                 #endif
623             }
624         }
625     }
626 }
```

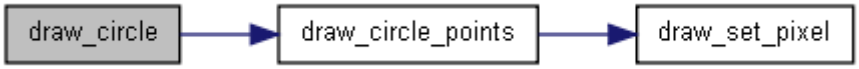
Here is the call graph for this function:

Here is the caller graph for this function:

void draw_circle (int *x_centre*, int *y_centre*, int *r*, uns8 *colour*)

```
392                                                                     {
393     int x,y;
394     int p = 1 - r;         // Initial value of decision parameter.
395
396     x = 0;
397     y = r;
398
399     draw_circle_points(x_centre, y_centre, x, y, 2);
400
401     while (x < y) {
402         x++;
403         if (p < 0)
404             p += 2 * x + 1;
405         else {
406             y--;
407             p += 2 * (x - y + 1);
408         }
409         draw_circle_points (x_centre, y_centre, x, y, colour);
410     }
411
412 }
```

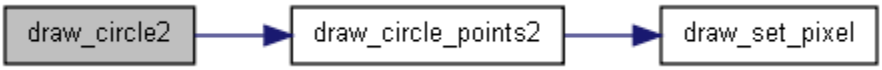
Here is the call graph for this function:



void draw_circle2 (int *x_centre*, int *y_centre*, int *r*, uns8 *colour*)

```
428                                                                     {
429     int x, y;
430     //int p = 1 - r; // orig
431     //int p = 3 - 2*r;         // (a) Initial value of decision parameter.
432     //int p = 1-r; // (b)
433     int p = 1 -r; // (c) 5//4 - r
434     x = 0;
435     y = r;
436 
437     draw_circle_points2(x_centre, y_centre, x, y, colour);
438 
439     while (x < y) {
440         x++;
441         if (p < 0)
442             //p += 2 * x + 2; // (orig)
443             //p = p + (4 * x) + 6; // (a)
444             //p = p + 2* x + 3; // (b)
445             p = p + 2* x + 1; // (c)
446         else {
447             y--;
448             //p += 2 * (x - y + 1); // (orig)
449             // p = p + 4 * (x - y) + 10; // (a)
450             //p = p + 2 * (x - y) + 5;
451             p = p + 2*(x-y) + 1;
452         }
453         draw_circle_points2 (x_centre, y_centre, x, y, colour);
454     }
455 
456 }
```

Here is the call graph for this function:

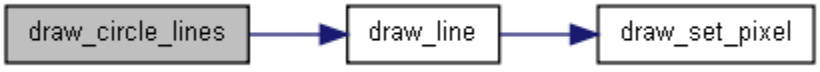


Here is the caller graph for this function:

void draw_circle_lines (int *ctr_x*, int *ctr_y*, int *pt_x*, int *pt_y*, uns8 *colour*)

```
347                                                                                 {
348     draw_line(ctr_x - pt_x, ctr_y + pt_y, ctr_x + pt_x, ctr_y + pt_y, colour);
349     draw_line(ctr_x - pt_x, ctr_y - pt_y, ctr_x + pt_x, ctr_y - pt_y, colour);
350     draw_line(ctr_x + pt_y, ctr_y + pt_x, ctr_x - pt_y, ctr_y + pt_x, colour);
351     draw_line(ctr_x + pt_y, ctr_y - pt_x, ctr_x - pt_y, ctr_y - pt_x, colour);
352 }
```

Here is the call graph for this function:



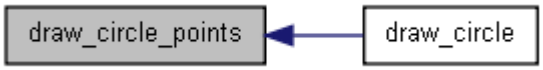
Here is the caller graph for this function:

void draw_circle_points (int *ctr_x*, int *ctr_y*, int *pt_x*, int *pt_y*, uns8 *colour*)

```
377                                                                                 {
378     // the eight symmetric points
379     draw_set_pixel (ctr_x + pt_x, ctr_y + pt_y, colour);
380     draw_set_pixel (ctr_x - pt_x, ctr_y + pt_y, colour);
381
382     draw_set_pixel (ctr_x + pt_x, ctr_y - pt_y, colour);
383     draw_set_pixel (ctr_x - pt_x, ctr_y - pt_y, colour);
384
385     draw_set_pixel (ctr_x + pt_y, ctr_y + pt_x, colour);
386     draw_set_pixel (ctr_x - pt_y, ctr_y + pt_x, colour);
387
388     draw_set_pixel (ctr_x + pt_y, ctr_y - pt_x, colour);
389     draw_set_pixel (ctr_x - pt_y, ctr_y - pt_x, colour);
390 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

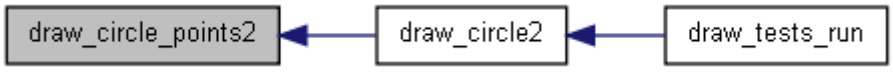


void draw_circle_points2 (int *ctr_x*, int *ctr_y*, int *pt_x*, int *pt_y*, uns8 *colour*)

```
413                                                                                 {
414     // the eight symmetric points
415     draw_set_pixel (ctr_x + pt_x + 1, ctr_y + pt_y +1, colour);
416     draw_set_pixel (ctr_x + pt_y + 1, ctr_y + pt_x +1, colour);
417
418     draw_set_pixel (ctr_x + pt_x + 1, ctr_y - pt_y, colour);
419     draw_set_pixel (ctr_x + pt_y + 1, ctr_y - pt_x, colour);
420
421     draw_set_pixel (ctr_x - pt_x, ctr_y - pt_y, colour);
422     draw_set_pixel (ctr_x - pt_y, ctr_y - pt_x, colour);
423
424     draw_set_pixel (ctr_x - pt_x, ctr_y + pt_y +1, colour);
425     draw_set_pixel (ctr_x - pt_y, ctr_y + pt_x +1, colour);
426
427 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

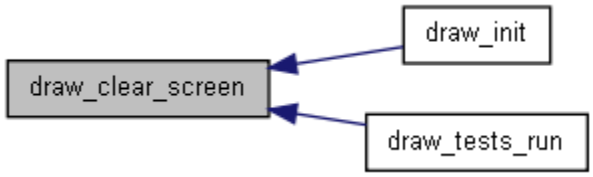


void draw_clear_screen ()

```
                        {

uns8 count;
    #if DRAW_TOTAL_BUFFER_SIZE == 1024 | DRAW_TOTAL_BUFFER_SIZE == 768 | DRAW_TOTAL_BUFFER_SIZE
== 512 | DRAW_TOTAL_BUFFER_SIZE == 256
        count = 0;
        do {
            draw_buffer0[count] = 0;
            #if DRAW_TOTAL_BUFFER_SIZE > 256
                draw_buffer1[count] = 0;
                #if DRAW_TOTAL_BUFFER_SIZE > 512
                    draw_buffer2[count] = 0;
                    #if DRAW_TOTAL_BUFFER_SIZE > 768
                        draw_buffer3[count] = 0;
                    #endif
                #endif
            #endif
            count++;
        } while (count != 0);
    #else
        #if DRAW_TOTAL_BUFFER_SIZE < 256
            count = 0;
            do {
                draw_buffer0[count] = 0;
                count++;
            } while (count < DRAW_TOTAL_BUFFER_SIZE);
        #else
            count = 0;
            do {
                draw_buffer0[count] = 0;
                count++;
            } while (count != 0);
            #if DRAW_TOTAL_BUFFER_SIZE < 512
                do {
                    draw_buffer1[count] = 0;
                    count++;
                } while (count < DRAW_TOTAL_BUFFER_SIZE - 256);
            #else
                do {
                    draw_buffer1[count] = 0;
                    count++;
                } while (count != 0);
                #if DRAW_TOTAL_BUFFER_SIZE > 512
                    #if DRAW_TOTAL_BUFFER_SIZE < 768
                        do {
                            draw_buffer2[count] = 0;
                            count++;
                        } while (count < DRAW_TOTAL_BUFFER_SIZE - 512);
                    #else
                        // > 768
                        do {
                            draw_buffer2[count] = 0;
                             count++;
                         } while (count != 0);

                         #if DRAW_TOTAL_BUFFER_SIZE < 1024
                            do {
                                draw_buffer3[count] = 0;
                                count++;
                            } while (count < DRAW_TOTAL_BUFFER_SIZE - 768);
                         #else
                            do {
                                draw_buffer3[count] = 0;
                                count++;
                            } while (count != 0);
```

```
113                         #endif
114                     #endif
115                 #endif
116
117             #endif
118
119         #endif
120     #endif
121
122 }
```

Here is the caller graph for this function:



void draw_filled_circle (int *x_centre*, int *y_centre*, int *r*, uns8 *colour*)

```
354                                                                         {
355     int x,y;
356     int p = 1 - r;          // Initial value of decision parameter.
357
358     x = 0;
359     y = r;
360
361     draw_circle_lines(x_centre, y_centre, x, y, colour);
362
363     while (x < y) {
364         x++;
365         if (p < 0)
366             p += 2 * x + 1;
367         else {
368             y--;
369             p += 2 * (x - y) + 1;
370         }
371         draw_circle_lines (x_centre, y_centre, x, y, colour);
372     }
373
374 }
```

Here is the call graph for this function:

uns8 draw_get_pixel (uns8 *x*, uns8 *y*)

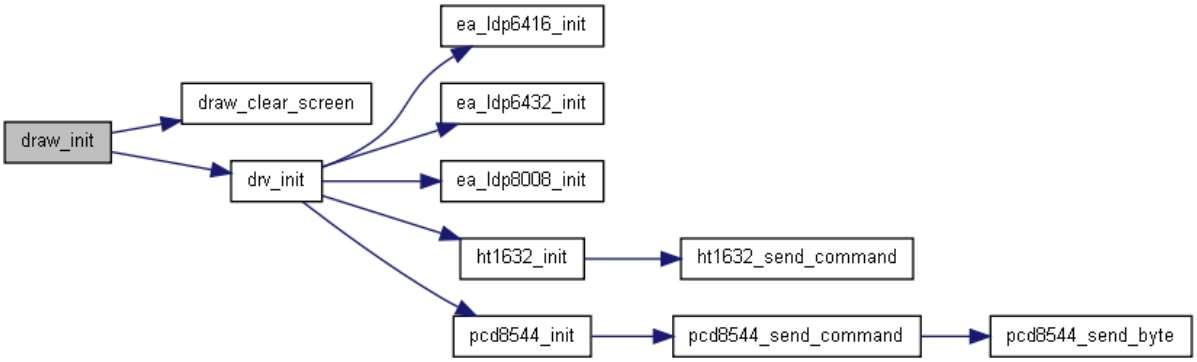
```
248                                     {
249     return 0;
250 }
```

void draw_init ()

```
128                     {
129     drv_init();
```

```
130     draw_clear_screen();
131 }
```

Here is the call graph for this function:



uns16 draw_length_str (char * *str*)

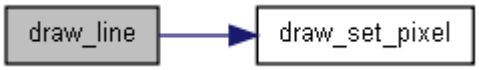
```
516                                 {
517 uns8 my_char;
518 uns16 length;
519 
520     length = 0;
521     while (*str != 0) {
522         my_char = *str;
523         my_char = my_char - 32;
524         length = length + (PicPack5x7_index[my_char + 1] - PicPack5x7_index[my_char]) + 1;
525         str++;
526     }
527     length = length - 1;    // no space at the end
528     return length;
529 }
```

void draw_line (uns8 *x0*, uns8 *y0*, uns8 *x1*, uns8 *y1*, uns8 *colour*)

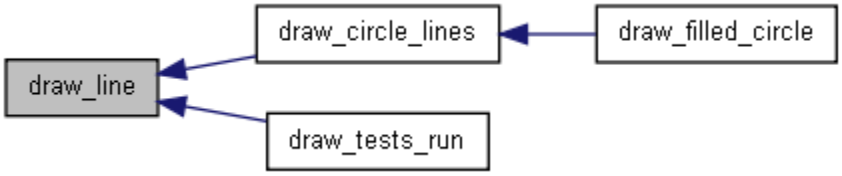
```
309                                                                 {
310 
311     int dy = y1 - y0;
312     int dx = x1 - x0;
313     int stepx, stepy;
314 
315     if (dy < 0) { dy = -dy;  stepy = -1; } else { stepy = 1; }
316     if (dx < 0) { dx = -dx;  stepx = -1; } else { stepx = 1; }
317     dy <<= 1;                                                   // dy is now 2*dy
318     dx <<= 1;                                                   // dx is now 2*dx
319 
320     draw_set_pixel(x0, y0, colour);
321     if (dx > dy) {
322         int fraction = dy - (dx >> 1);                          // same as 2*dy - dx
323         while (x0 != x1) {
324             if (fraction >= 0) {
325                 y0 += stepy;
326                 fraction -= dx;                                 // same as fraction -= 2*dx
327             }
328             x0 += stepx;
329             fraction += dy;                                     // same as fraction -= 2*dy
330             draw_set_pixel(x0, y0, colour);
331         }
332     } else {
333         int fraction = dx - (dy >> 1);
```

```
334         while (y0 != y1) {
335             if (fraction >= 0) {
336                 x0 += stepx;
337                 fraction -= dy;
338             }
339             y0 += stepy;
340             fraction += dx;
341             draw_set_pixel(x0, y0, colour);
342         }
343     }
344 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

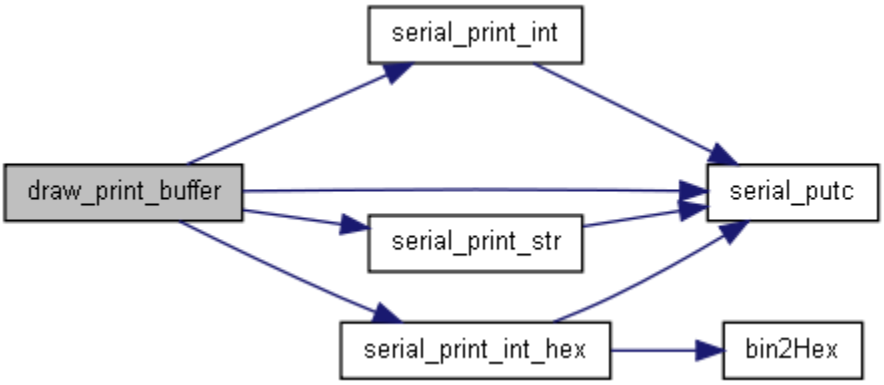


void draw_print_buffer ()

```
268                             {
269 #ifdef DRAW_DEBUG
270 uns8    inv_y, x , y,
271          byte_loc, bit_loc;
272 uns16    buffer_loc;
273 
274     for(y = 0 ; y < DRAW_PIXELS_HIGH ; y++) {
275         inv_y = DRAW_PIXELS_HIGH - 1 - y; // need to print out from the top
276         if (inv_y < 10) {
277             serial_putc('0');
278         }
279         serial_print_int(inv_y);
280         serial_putc(' ');
281         serial_print_int_hex(inv_y * DRAW_PIXELS_WIDE / DRAW_PIXELS_PER_BYTE);
282         serial_putc('|');
283         for(x = 0 ; x < DRAW_PIXELS_WIDE  ; x++) {
284             buffer_loc = inv_y * DRAW_PIXELS_WIDE + x;
285             byte_loc = buffer_loc / DRAW_PIXELS_PER_BYTE;
286             bit_loc = buffer_loc & (DRAW_PIXELS_PER_BYTE -1);
287 
288             //if (bit_loc == 0) {
289             //  serial_putc(' ');
290             //  serial_print_int_hex(byte_loc);
291             //  serial_putc(' ');
292             //}
293 
294             if (test_bit(draw_buffer0[byte_loc], bit_loc)) {
295 
296                     serial_putc('1');
297 
298             } else {
299                 serial_putc('0');
300             }
301         }
302 
303         serial_print_str("|\n");
304 
```

```
305     }
306 #endif
307 }
```

Here is the call graph for this function:



void draw_print_str (uns8 *x*, uns8 *y*, uns8 *width*, uns8 *start_pixel*, uns8 *colour*, char * *str*)

```
531
{
532
533 uns8 my_char;
534 uns16 index_pos;
535 uns16 index_pos_next;
536 uns16 count, s_count;
537 uns8 sliver, x origin, y origin, pixel;
538     y_origin = y;
539     x_origin = x;
540     pixel = 0;
541     while (*str != 0) {
542         // first look up character in index
543         my_char = *str;
544         my_char = my_char - 32;
545
546
547         index_pos = PicPack5x7 index[my_char];
548         index_pos_next = PicPack5x7 index[my_char + 1];
549             for(count = index_pos ; count < index_pos_next ; count++) {
550                 if (count < 256) {
551                     sliver = PicPack5x7 bitmap 0[count];
552                 } else {
553                     sliver = PicPack5x7 bitmap 1[count-256];
554                 }
555                 // Now iterate over sliver
556                 if (pixel >= start_pixel) {
557                     s_count = 0;
558                     while (s count < 7) {
559                         if (test_bit(sliver, 6)) {
560                             #if DRAW_HW_Y_ORIGIN == BOTTOM_LEFT
561                                 draw set pixel(x, y + s_count, colour);
562                             #else
563                             // DRAW_HW_Y_ORIGIN == TOP_LEFT
564                                 draw set pixel(x, y - s_count, colour);
565                             #endif
566                         }
567                         sliver <<= 1;
568                         s_count++;
569                     }
570                     x++;
```

```
571                 }
572                 if (x - x_origin == width) {
573                     return;
574                 }
575                 pixel++;
576                 // Need to add a bit here to only
577                 // increment to new line when we have got to height chars
578 
579             }
580         str++;
581         if (pixel >= start pixel) {
582             x++;
583         }
584         pixel++;
585 
586         if (x - x_origin == width) {
587             return;
588         }
589     }
590 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

void draw_rect (uns8 *x*, uns8 *y*, uns16 *width*, uns8 *height*, uns8 *colour*)

```
253                                                                     {
254 uns16 dx, dy;
255 
256     for(dy = y ; dy < y + height ; dy++) {
257         for(dx = x ; dx < x + width ; dx++) {
258             draw_set_pixel(dx, dy, colour);
259 /*          serial_print_str("P:");
260             serial_print_int(dx);
261             serial_print_str(",");
262             serial_print_int(dy);
263             serial_print_nl();
264 */      }
265     }
266 }
```

Here is the call graph for this function:

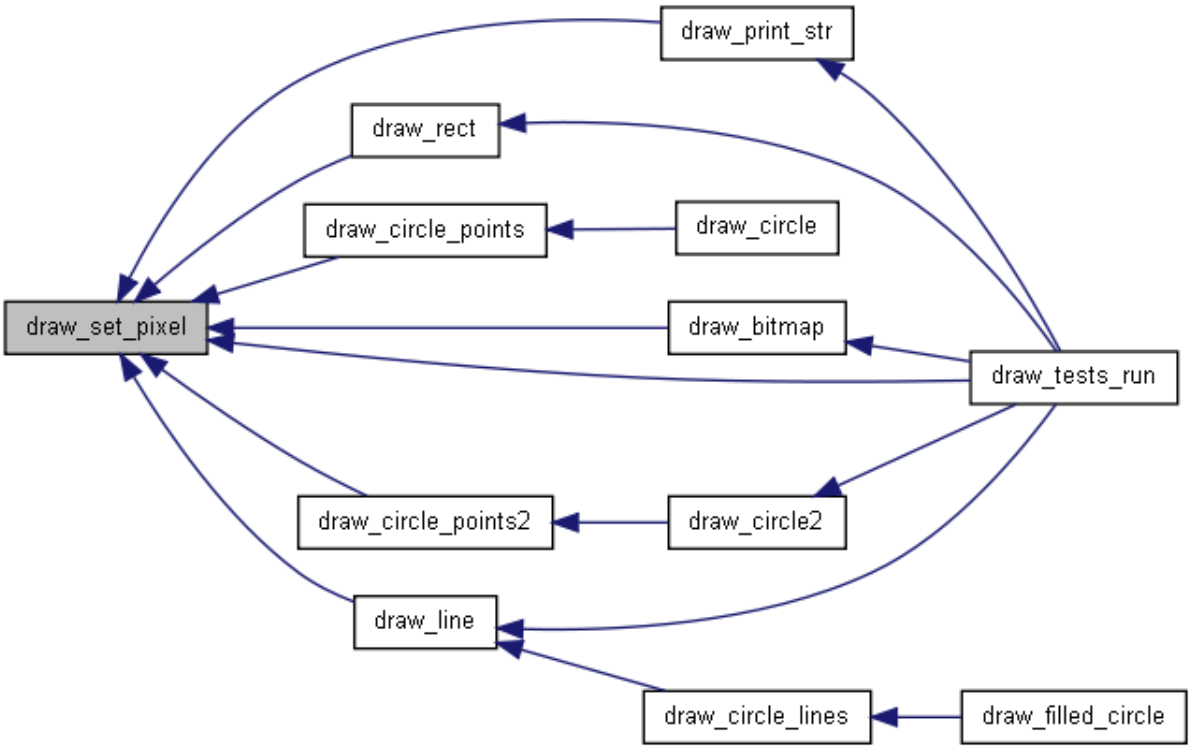
Here is the caller graph for this function:

void draw_set_pixel (uns8 *x*, uns8 *y*, uns8 *colour*)

```
135                                                 {
136
137 uns8 *buffer;
138 uns16 buffer_loc, loc_byte;
139 uns8  loc_bit, loc_in_buffer, buffer_num;
140 uns8  bit_count;
141     // inverse here
142     // y = DRAW_PIXELS_HIGH - 1 - y;
143
144     #if DRAW_HW_Y_ORIGIN == TOP_LEFT
145         #if DRAW_HW_BUFFER_ORIENTATION == HORIZONTAL
146             buffer_loc = y * DRAW_PIXELS_WIDE + x;
147         #else
148         // DRAW_HW_BUFFER_ORIENTATION == VERTICAL
149             buffer_loc = x * DRAW_PIXELS_HIGH + y;
150         #endif
151     #else
152     // DRAW_HW_Y_ORIENTATION == BOTTOM_LEFT
153         #if DRAW_HW_BUFFER_ORIENTATION == HORIZONTAL
154             buffer_loc = y * DRAW_PIXELS_WIDE + x;
155         #else
156         // DRAW_HW_BUFFER_ORIENTATION == VERTICAL
157             buffer_loc = x * DRAW_PIXELS_HIGH + y;
158         #endif
159     #endif
160 /*    serial_print_int(x);
161     serial_putc(' ');
162     serial_print_int(y);
163     serial_print_nl();
164
165     serial_print_str("->");
166     serial_print_int(buffer_loc);
167 */
168
169     buffer_loc = buffer_loc * DRAW_BITS_PER_PIXEL;
170 //  loc_byte = buffer_loc / DRAW_PIXELS_PER_BYTE;
171     loc_byte = buffer_loc / 8;
172     loc_bit = (buffer_loc & (0x07));
173
174 /*byte  0        1        2
175 bit     0 2 4 6  0 2 4 6  0 2 4 6
176 pix     0 1 2 3  4 5 6 7  8 9 a b
177 bit pos 0 2 4 6  8 a c e
178 */
179     loc_in_buffer = loc_byte & 0xff;
180     buffer_num = loc_byte >> 8;
181
182     /*
183     // For debugging
184     serial_print_str(" x=");
185     serial_print_int(x);
186     serial_print_str(" y=");
187     serial_print_int(y);
188     serial_print_str(" buffer_loc=");
189     serial_print_int(buffer_loc);
190     serial_print_str(" loc_byte=");
191     serial_print_int(loc_byte);
192     serial_print_str(" loc_bit=");
193     serial_print_int(loc_bit);
194     serial_print_str(" LIB=");
195     serial_print_int(loc_in_buffer);
196     serial_print_nl();
197     serial_print_str(" bnum=");
198     serial_print_int(buffer_num);
199     serial_print_str("\n");
200     */
201     if (buffer_num == 0) {
202         buffer = &draw_buffer0;
203     }
204     #if DRAW_TOTAL_BUFFER_SIZE > 256
205
```

```
206     else if (buffer_num == 1) {
207         buffer = &draw_buffer1;
208 
209     }
210         #if DRAW_TOTAL_BUFFER_SIZE > 512
211         else if (buffer_num == 2) {
212             buffer = &draw_buffer2;
213 
214         }
215             #if DRAW_TOTAL_BUFFER_SIZE > 768
216             else if (buffer num == 3) {
217             buffer = &draw_buffer3;
218 
219             }
220             #endif
221         #endif
222     #endif
223     #if DRAW_BITS_PER_PIXEL > 1
224 
225         bit_count = 0;
226 
227         while (bit_count < DRAW_BITS_PER_PIXEL) {
228             if (test bit(colour, bit count)) {
229                 set_bit(buffer[loc_in_buffer], loc_bit);
230             } else {
231                 clear_bit(buffer[loc_in_buffer], loc_bit);
232             }
233             bit count++;
234             loc_bit++;
235         }
236     #else
237 
238         if (colour) {
239             set_bit(buffer[loc_in_buffer], loc_bit);
240         } else {
241             clear_bit(buffer[loc_in_buffer], loc_bit);
242         }
243     #endif
244 
245 }
```

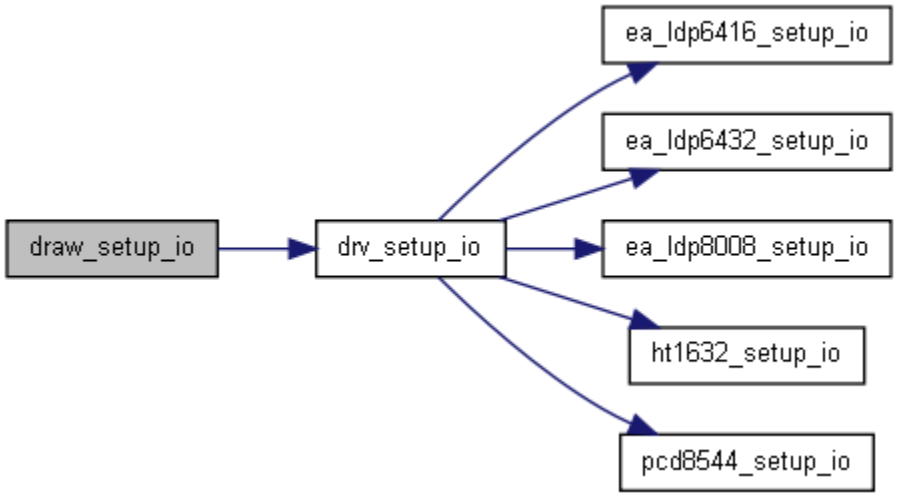
Here is the caller graph for this function:



void draw_setup_io ()

```
124                         {
125     drv_setup_io();
126 }
```

Here is the call graph for this function:



Variable Documentation

rom char PicPack5x7_bitmap_0[1]

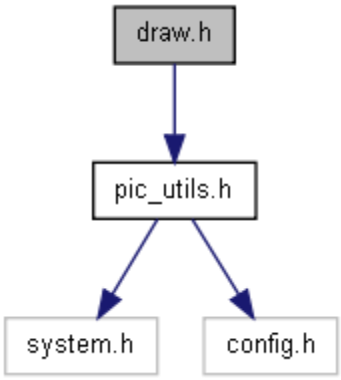
rom char PicPack5x7_bitmap_1[1]

uns16 PicPack5x7_index[1]

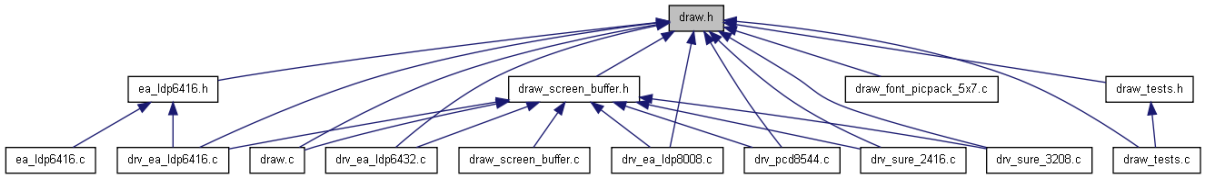
---

draw.h File Reference

Buffered graphics routines.
Include dependency graph for draw.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define BOTTOM_LEFT 1
- #define draw_paint() drv_paint()
- #define DRAW_PIXELS_PER_BYTE (8 / DRAW_BITS_PER_PIXEL)
- #define draw_set_display_brightness(brightness) drv_set_display_brightness(brightness)
- #define drv_setup() drv_setup_io()
- #define HORIZONTAL 0
- #define TOP_LEFT 0
- #define VERTICAL 1

Functions

- void draw_bitmap (uns8 x, uns8 y, uns8 colour, char *bitmap)
- void draw_circle (int x_centre, int y_centre, int r, uns8 colour)
- void draw_circle2 (int x_centre, int y_centre, int r, uns8 colour)

- void draw_clear_screen ()
- uns8 draw_get_pixel (uns8 x, uns8)
- void draw_init ()
- uns16 draw_length_str (char *str)
- void draw_line (uns8 x0, uns8 y0, uns8 x1, uns8 y1, uns8 colour)
- void draw_print_buffer ()
- void draw_print_str (uns8 x, uns8 y, uns8 width, uns8 start_pixel, uns8 colour, char *str)
- void draw_rect (uns8 x, uns8 y, uns16 width, uns8 height, uns8 colour)
- void draw_set_pixel (uns8 x, uns8 y, uns8 colour)
- void draw_setup_io ()
- void drv_init ()
- void drv_paint ()
- void drv_print_buffer ()
- void drv_refresh ()
- void drv_set_display_brightness (uns8 brightness)
- void drv_setup_io ()

Detailed Description

You will need to pick a hardware buffering mode for the draw routines.

Draw buffer addressing

4 | U V W X Y 3 | P Q R S T 2 | K L M N O 1 | F G H I J 0 | A B C D E --------- 0 1 2 3 4 DRAW_HW_Y_ORIGIN == BOTTOM_LEFT DRAW_HW_BUFFER_ORIENTATION == HORIZONTAL

0 | A B C D E 1 | F G H I J 2 | K L M N O 3 | P Q R S T 4 | U V W X Y --------- 0 1 2 3 4 DRAW_HW_Y_ORIGIN == TOP_LEFT DRAW_HW_BUFFER_ORIENTATION == HORIZONTAL

0 | E J O T Y 1 | D I N S X 2 | C H M R W 3 | B G L Q V 4 | A F K P U --------- 0 1 2 3 4 DRAW_HW_Y_ORIGIN == BOTTOM_LEFT DRAW_HW_BUFFER_ORIENTATION == VERTICAL

0 | A F K P U 1 | B G L Q V 2 | C H M R W 3 | D I N S X 4 | E J O T Y --------- 0 1 2 3 4 DRAW_HW_Y_ORIGIN == TOP_LEFT DRAW_HW_BUFFER_ORIENTATION == VERTICAL

Put the following in your config.h:

```
// - - - - - - - - - - - - - - - - - - - -
// Draw defines
// - - - - - - - - - - - - - - - - - - - -

#define DRAW_PIXELS_HIGH 24
#define DRAW_PIXELS_WIDE 16
#define DRAW_BITS_PER_PIXEL 1

#define DRAW_HW_Y_ORIGIN TOP_LEFT
// or BOTTOM_LEFT

#define DRAW_HW_BUFFER_ORIENTATION VERTICAL
// or HORIZONTAL

//Enable debug to see what's happening under the hood
//#define DRAW_DEBUG

// - - - - - - - - - - - - - - - - - - - -
```

Define Documentation

#define BOTTOM_LEFT 1

#define draw_paint() drv_paint()

#define DRAW_PIXELS_PER_BYTE (8 / DRAW_BITS_PER_PIXEL)

#define draw_set_display_brightness(brightness) drv_set_display_brightness(brightness)

#define drv_setup() drv_setup_io()

#define HORIZONTAL 0

#define TOP_LEFT 0

#define VERTICAL 1

Function Documentation

void draw_bitmap (uns8 *x*, uns8 *y*, uns8 *colour*, char * *bitmap*)

```
593                                                                 {
594
595
596 uns8 bitpos = 0;
597 uns8 bytepos = 0;
598 uns8 value;
599 uns8 bitmap_width = bitmap[0];
600 uns8 bitmap_height = bitmap[1];
601 uns8 bitmap_bpp = bitmap[2];
602 uns8 xbitmap, ybitmap;
603
604
605     bytepos = 3;    // first byte of data - 1
606     bitpos = 0b10000000;    // left most bit
607
608     for (xbitmap = 0; xbitmap < bitmap_width; xbitmap++) {
609         for (ybitmap = 0; ybitmap < bitmap_height; ybitmap++) {
610             if (bitpos == 0b10000000) {
611                 bitpos = 0b00000001;
612                 bytepos++;
613                 value = bitmap[bytepos];
614             } else {
615                 bitpos = bitpos << 1;
616             }
617             if (value & bitpos) {
618                 #if DRAW_HW_Y_ORIGIN == TOP_LEFT
619                     draw_set_pixel(x + xbitmap, y + ybitmap, colour);
620                 #else
621                     draw_set_pixel(x + xbitmap, y - ybitmap, colour);
622                 #endif
623             }
624         }
625     }
626 }
```

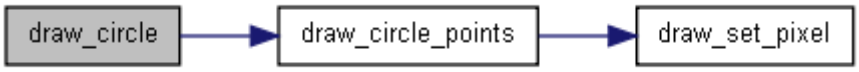
Here is the call graph for this function:

Here is the caller graph for this function:

void draw_circle (int *x_centre*, int *y_centre*, int *r*, uns8 *colour*)

```
392                                                                     {
393     int x,y;
394     int p = 1 - r;          // Initial value of decision parameter.
395
396     x = 0;
397     y = r;
398
399     draw_circle_points(x_centre, y_centre, x, y, 2);
400
401     while (x < y) {
402         x++;
403         if (p < 0)
404             p += 2 * x + 1;
405         else {
406             y--;
407             p += 2 * (x - y + 1);
408         }
409         draw_circle_points (x_centre, y_centre, x, y, colour);
410     }
411
412 }
```

Here is the call graph for this function:

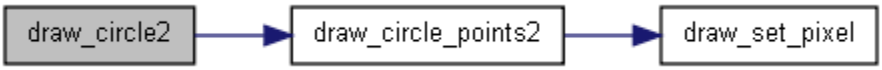


void draw_circle2 (int *x_centre*, int *y_centre*, int *r*, uns8 *colour*)

```
428                                                                     {
429     int x, y;
430     //int p = 1 - r; // orig
431     //int p = 3 - 2*r;          // (a) Initial value of decision parameter.
432     //int p = 1-r; // (b)
433     int p = 1 -r; // (c) 5//4 - r
434     x = 0;
435     y = r;
436
437     draw_circle_points2(x_centre, y_centre, x, y, colour);
438
439     while (x < y) {
440         x++;
441         if (p < 0)
442             //p += 2 * x + 2; // (orig)
443             //p = p + (4 * x) + 6; // (a)
444             //p = p + 2* x + 3; // (b)
445             p = p + 2* x + 1; // (c)
446         else {
447             y--;
448             //p += 2 * (x - y + 1); // (orig)
449             // p = p + 4 * (x - y) + 10; // (a)
```

```
450             //p = p + 2 * (x - y) + 5;
451             p = p + 2*(x-y) + 1;
452         }
453         draw_circle_points2 (x_centre, y_centre, x, y, colour);
454     }
455
456 }
```

Here is the call graph for this function:



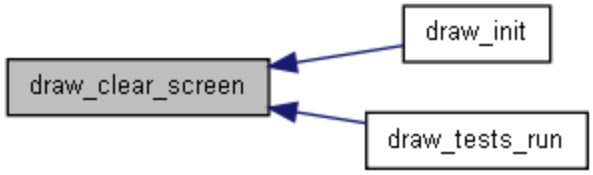
Here is the caller graph for this function:

void draw_clear_screen ()

```
49                          {
50
51 uns8 count;
52     #if DRAW_TOTAL_BUFFER_SIZE == 1024 | DRAW_TOTAL_BUFFER_SIZE == 768 | DRAW_TOTAL_BUFFER_SIZE
== 512 | DRAW_TOTAL_BUFFER_SIZE == 256
53         count = 0;
54         do {
55             draw_buffer0[count] = 0;
56             #if DRAW_TOTAL_BUFFER_SIZE > 256
57                 draw_buffer1[count] = 0;
58                 #if DRAW_TOTAL_BUFFER_SIZE > 512
59                     draw_buffer2[count] = 0;
60                     #if DRAW_TOTAL_BUFFER_SIZE > 768
61                         draw_buffer3[count] = 0;
62                     #endif
63                 #endif
64             #endif
65             count++;
66         } while (count != 0);
67     #else
68         #if DRAW_TOTAL_BUFFER_SIZE < 256
69             count = 0;
70             do {
71                 draw_buffer0[count] = 0;
72                 count++;
73             } while (count < DRAW_TOTAL_BUFFER_SIZE);
74         #else
75             count = 0;
76             do {
77                 draw_buffer0[count] = 0;
78                 count++;
79             } while (count != 0);
80             #if DRAW_TOTAL_BUFFER_SIZE < 512
81                 do {
82                     draw_buffer1[count] = 0;
83                     count++;
84                 } while (count < DRAW_TOTAL_BUFFER_SIZE - 256);
85             #else
86                 do {
87                     draw_buffer1[count] = 0;
88                     count++;
89                 } while (count != 0);
90                 #if DRAW_TOTAL_BUFFER_SIZE > 512
91                     #if DRAW_TOTAL_BUFFER_SIZE < 768
92                         do {
```

```
93                              draw_buffer2[count] = 0;
94                              count++;
95                          } while (count < DRAW_TOTAL_BUFFER_SIZE - 512);
96                      #else
97                          // > 768
98                          do {
99                              draw_buffer2[count] = 0;
100                             count++;
101                         } while (count != 0);
102
103                         #if DRAW_TOTAL_BUFFER_SIZE < 1024
104                             do {
105                                 draw_buffer3[count] = 0;
106                                 count++;
107                             } while (count < DRAW_TOTAL_BUFFER_SIZE - 768);
108                         #else
109                             do {
110                                 draw_buffer3[count] = 0;
111                                 count++;
112                             } while (count != 0);
113                         #endif
114                     #endif
115                 #endif
116
117             #endif
118
119         #endif
120     #endif
121
122 }
```

Here is the caller graph for this function:



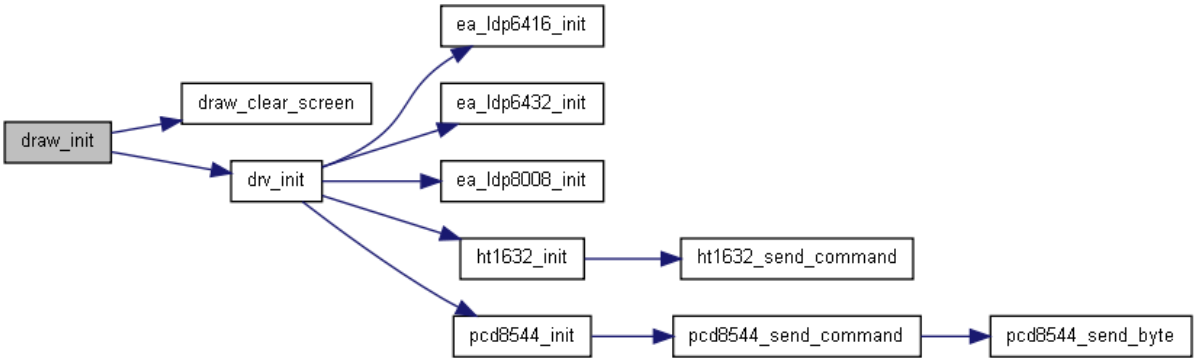
uns8 draw_get_pixel (uns8 *x*, uns8)

```
248                                     {
249     return 0;
250 }
```

void draw_init ()

```
128                 {
129     drv_init();
130     draw_clear_screen();
131 }
```

Here is the call graph for this function:



uns16 draw_length_str (char * *str*)

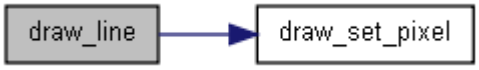
```
516                                 {
517 uns8 my_char;
518 uns16 length;
519
520     length = 0;
521     while (*str != 0) {
522         my_char = *str;
523         my_char = my_char - 32;
524         length = length + (PicPack5x7_index[my_char + 1] - PicPack5x7_index[my_char]) + 1;
525         str++;
526     }
527     length = length - 1;    // no space at the end
528     return length;
529 }
```

void draw_line (uns8 *x0*, uns8 *y0*, uns8 *x1*, uns8 *y1*, uns8 *colour*)

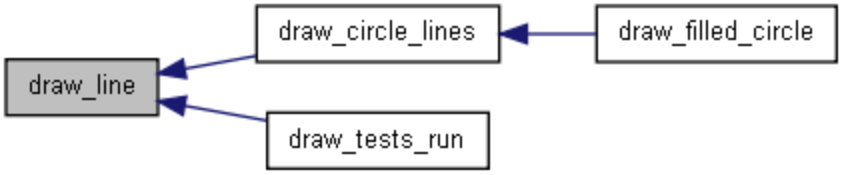
```
309                                                                 {
310
311     int dy = y1 - y0;
312     int dx = x1 - x0;
313     int stepx, stepy;
314
315     if (dy < 0) { dy = -dy;  stepy = -1; } else { stepy = 1; }
316     if (dx < 0) { dx = -dx;  stepx = -1; } else { stepx = 1; }
317     dy <<= 1;                                                   // dy is now 2*dy
318     dx <<= 1;                                                   // dx is now 2*dx
319
320     draw_set_pixel(x0, y0, colour);
321     if (dx > dy) {
322         int fraction = dy - (dx >> 1);                          // same as 2*dy - dx
323         while (x0 != x1) {
324             if (fraction >= 0) {
325                 y0 += stepy;
326                 fraction -= dx;                                 // same as fraction -= 2*dx
327             }
328             x0 += stepx;
329             fraction += dy;                                     // same as fraction -= 2*dy
330             draw_set_pixel(x0, y0, colour);
331         }
332     } else {
333         int fraction = dx - (dy >> 1);
334         while (y0 != y1) {
335             if (fraction >= 0) {
336                 x0 += stepx;
337                 fraction -= dy;
338             }
```

```
339             y0 += stepy;
340             fraction += dx;
341             draw_set_pixel(x0, y0, colour);
342         }
343     }
344 }
```

Here is the call graph for this function:



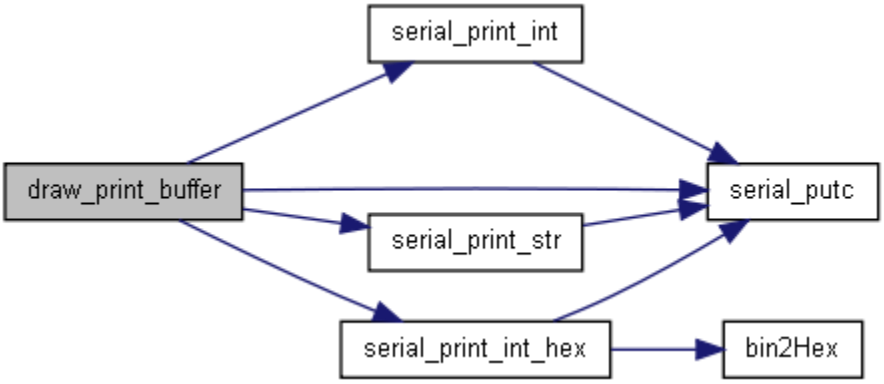
Here is the caller graph for this function:



void draw_print_buffer ()

```
268                         {
269 #ifdef DRAW_DEBUG
270 uns8    inv_y, x , y,
271          byte loc, bit loc;
272 uns16    buffer_loc;
273 
274     for(y = 0 ; y < DRAW_PIXELS_HIGH ; y++) {
275         inv_y = DRAW_PIXELS_HIGH - 1 - y; // need to print out from the top
276         if (inv_y < 10) {
277             serial_putc('0');
278         }
279         serial_print_int(inv_y);
280         serial_putc(' ');
281         serial_print_int_hex(inv_y * DRAW_PIXELS_WIDE / DRAW_PIXELS_PER_BYTE);
282         serial_putc('|');
283         for(x = 0 ; x < DRAW_PIXELS_WIDE  ; x++) {
284             buffer_loc = inv_y * DRAW_PIXELS_WIDE + x;
285             byte_loc = buffer_loc / DRAW_PIXELS_PER_BYTE;
286             bit_loc = buffer_loc & (DRAW_PIXELS_PER_BYTE -1);
287 
288             //if (bit_loc == 0) {
289             //  serial_putc(' ');
290             //  serial_print_int_hex(byte_loc);
291             //  serial_putc(' ');
292             //}
293 
294             if (test_bit(draw_buffer0[byte_loc], bit_loc)) {
295 
296                     serial_putc('1');
297 
298             } else {
299                 serial_putc('0');
300             }
301         }
302 
303         serial_print_str("|\n");
304 
305     }
306 #endif
307 }
```

Here is the call graph for this function:



void draw_print_str (uns8 *x*, uns8 *y*, uns8 *width*, uns8 *start_pixel*, uns8 *colour*, char * *str*)

```
531
{
532
533 uns8 my_char;
534 uns16 index_pos;
535 uns16 index_pos_next;
536 uns16 count, s_count;
537 uns8 sliver, x_origin, y_origin, pixel;
538     y_origin = y;
539     x_origin = x;
540     pixel = 0;
541     while (*str != 0) {
542         // first look up character in index
543         my_char = *str;
544         my_char = my_char - 32;
545
546
547         index_pos = PicPack5x7_index[my_char];
548         index_pos_next = PicPack5x7_index[my_char + 1];
549             for(count = index_pos ; count < index_pos_next ; count++) {
550                 if (count < 256) {
551                     sliver = PicPack5x7_bitmap_0[count];
552                 } else {
553                     sliver = PicPack5x7_bitmap_1[count-256];
554                 }
555                 // Now iterate over sliver
556                 if (pixel >= start_pixel) {
557                     s_count = 0;
558                     while (s_count < 7) {
559                         if (test_bit(sliver, 6)) {
560                             #if DRAW_HW_Y_ORIGIN == BOTTOM_LEFT
561                                 draw_set_pixel(x, y + s_count, colour);
562                             #else
563                             // DRAW_HW_Y_ORIGIN == TOP_LEFT
564                                 draw_set_pixel(x, y - s_count, colour);
565                             #endif
566                         }
567                         sliver <<= 1;
568                         s_count++;
569                     }
570                     x++;
571                 }
572                 if (x - x_origin == width) {
573                     return;
574                 }
575                 pixel++;
```

```
576                 // Need to add a bit here to only
577                 // increment to new line when we have got to height chars
578
579             }
580         str++;
581         if (pixel >= start_pixel) {
582             x++;
583         }
584         pixel++;
585
586         if (x - x origin == width) {
587             return;
588         }
589     }
590 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

void draw_rect (uns8 *x*, uns8 *y*, uns16 *width*, uns8 *height*, uns8 *colour*)

```
253                                                                     {
254 uns16 dx, dy;
255
256     for(dy = y ; dy < y + height ; dy++) {
257         for(dx = x ; dx < x + width ; dx++) {
258             draw_set_pixel(dx, dy, colour);
259 /*          serial_print_str("P:");
260             serial_print_int(dx);
261             serial_print_str(",");
262             serial_print_int(dy);
263             serial_print_nl();
264 */      }
265     }
266 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

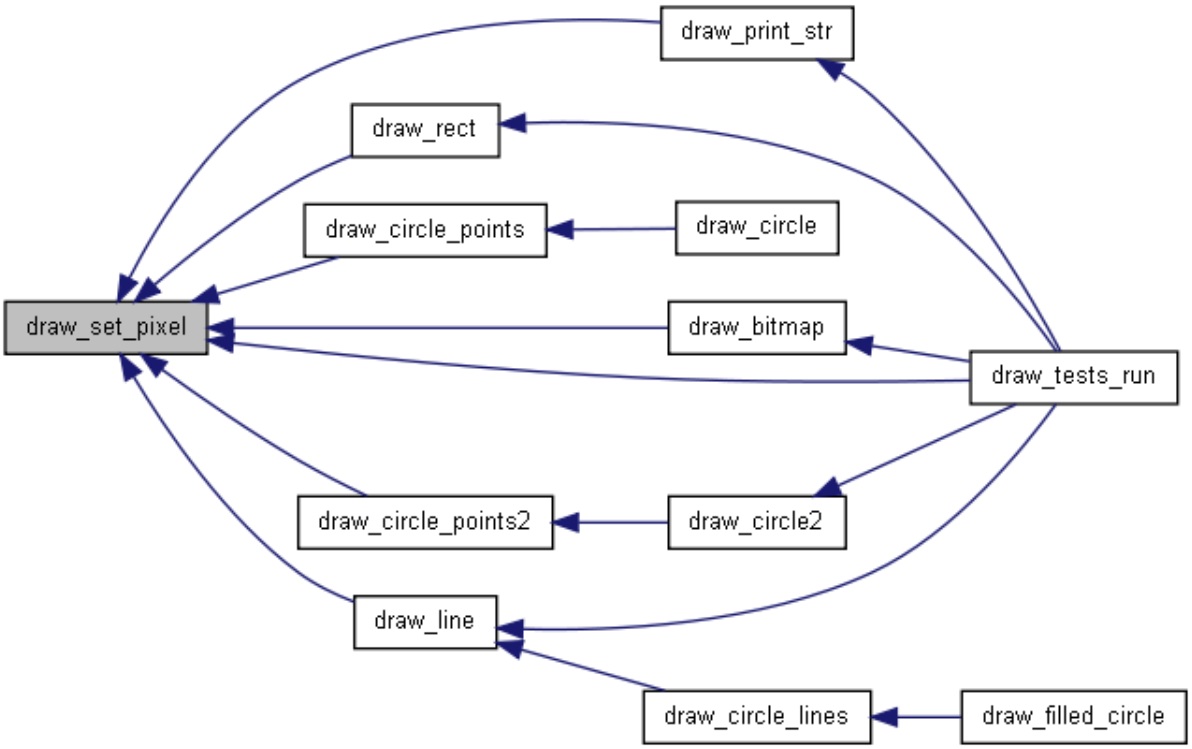
void draw_set_pixel (uns8 *x*, uns8 *y*, uns8 *colour*)

```
135                                                 {
136
137 uns8 *buffer;
138 uns16 buffer_loc, loc_byte;
139 uns8  loc_bit, loc_in_buffer, buffer_num;
```

```
140 uns8  bit_count;
141     // inverse here
142     // y = DRAW_PIXELS_HIGH - 1 - y;
143 
144     #if DRAW_HW_Y_ORIGIN == TOP_LEFT
145         #if DRAW_HW_BUFFER_ORIENTATION == HORIZONTAL
146             buffer_loc = y * DRAW_PIXELS_WIDE + x;
147         #else
148         // DRAW_HW_BUFFER_ORIENTATION == VERTICAL
149             buffer_loc = x * DRAW_PIXELS_HIGH + y;
150         #endif
151     #else
152     // DRAW_HW_Y_ORIENTATION == BOTTOM_LEFT
153         #if DRAW_HW_BUFFER_ORIENTATION == HORIZONTAL
154             buffer_loc = y * DRAW_PIXELS_WIDE + x;
155         #else
156         // DRAW_HW_BUFFER_ORIENTATION == VERTICAL
157             buffer_loc = x * DRAW_PIXELS_HIGH + y;
158         #endif
159     #endif
160 /*    serial_print_int(x);
161     serial_putc(' ');
162     serial_print_int(y);
163     serial_print_nl();
164 
165     serial_print_str("->");
166     serial_print_int(buffer_loc);
167 */
168 
169     buffer_loc = buffer_loc * DRAW_BITS_PER_PIXEL;
170 //  loc_byte = buffer_loc / DRAW_PIXELS_PER_BYTE;
171     loc_byte = buffer_loc / 8;
172     loc_bit = (buffer_loc & (0x07));
173 
174 /*byte  0        1        2
175 bit     0 2 4 6  0 2 4 6  0 2 4 6
176 pix     0 1 2 3  4 5 6 7  8 9 a b
177 bit pos 0 2 4 6  8 a c e
178 */
179     loc_in_buffer = loc_byte & 0xff;
180     buffer_num = loc_byte >> 8;
181 
182     /*
183     // For debugging
184     serial_print_str(" x=");
185     serial_print_int(x);
186     serial_print_str(" y=");
187     serial_print_int(y);
188     serial_print_str(" buffer_loc=");
189     serial_print_int(buffer_loc);
190     serial_print_str(" loc_byte=");
191     serial_print_int(loc_byte);
192     serial_print_str(" loc_bit=");
193     serial_print_int(loc_bit);
194     serial_print_str(" LIB=");
195     serial_print_int(loc_in_buffer);
196     serial_print_nl();
197     serial_print_str(" bnum=");
198     serial_print_int(buffer_num);
199     serial_print_str("\n");
200     */
201     if (buffer_num == 0) {
202         buffer = &draw_buffer0;
203     }
204     #if DRAW_TOTAL_BUFFER_SIZE > 256
205 
206     else if (buffer_num == 1) {
207         buffer = &draw_buffer1;
208 
209     }
210         #if DRAW_TOTAL_BUFFER_SIZE > 512
```

```
211         else if (buffer_num == 2) {
212             buffer = &draw_buffer2;
213 
214         }
215             #if DRAW_TOTAL_BUFFER_SIZE > 768
216             else if (buffer_num == 3) {
217             buffer = &draw_buffer3;
218 
219             }
220             #endif
221         #endif
222     #endif
223     #if DRAW_BITS_PER_PIXEL > 1
224 
225         bit count = 0;
226 
227         while (bit_count < DRAW_BITS_PER_PIXEL) {
228             if (test_bit(colour, bit_count)) {
229                 set_bit(buffer[loc_in_buffer], loc_bit);
230             } else {
231                 clear_bit(buffer[loc_in_buffer], loc_bit);
232             }
233             bit count++;
234             loc_bit++;
235         }
236     #else
237 
238         if (colour) {
239             set_bit(buffer[loc_in_buffer], loc_bit);
240         } else {
241             clear_bit(buffer[loc_in_buffer], loc_bit);
242         }
243     #endif
244 
245 }
```

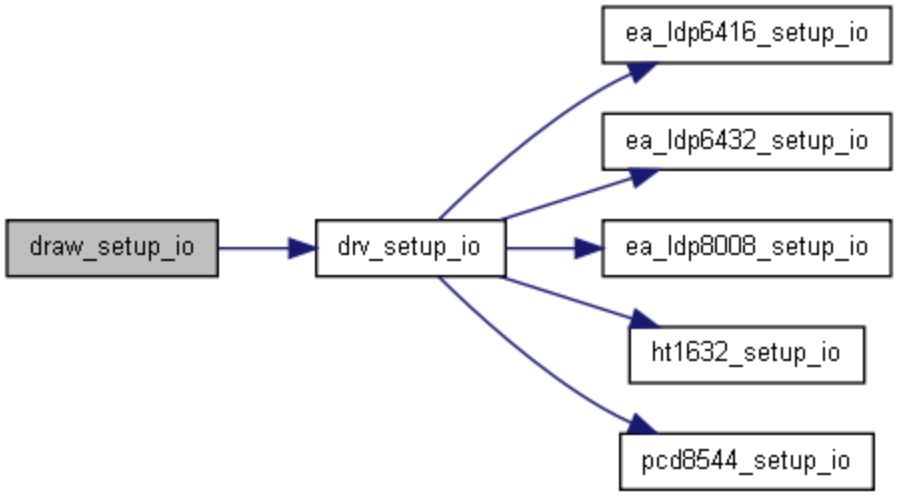
Here is the caller graph for this function:



void draw_setup_io ()

```
124                     {
125     drv_setup_io();
126 }
```

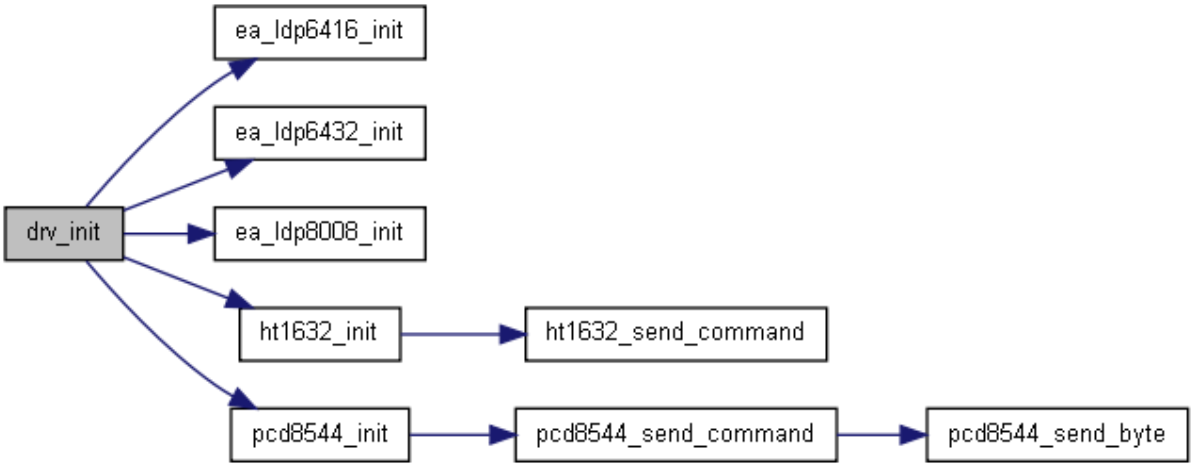
Here is the call graph for this function:



void drv_init ()

```
287                 {
288     ea_ldp6416_init();
289 }
```

Here is the call graph for this function:

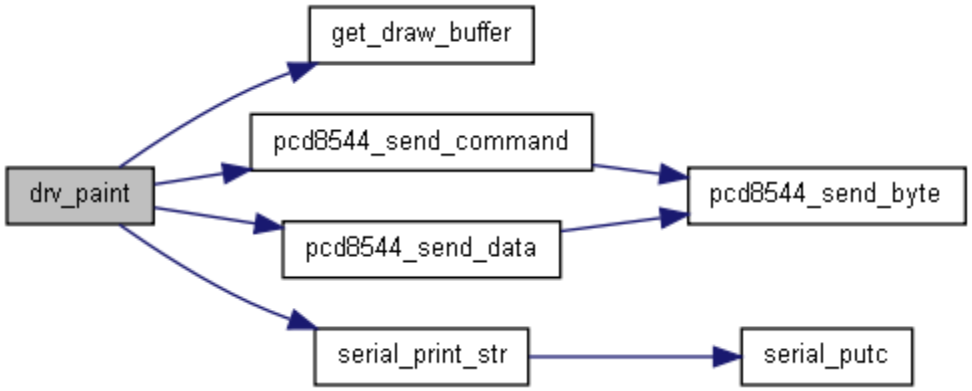


Here is the caller graph for this function:

void drv_paint ()

```
73                  {
74     uns8 count;
75     // start_crit_sec();
76     count = 0;
77
78     do {
79         buffer0[count] = draw_buffer0[count];
80         #if ea_ldp6416_displays > 1
81             buffer1[count] = draw_buffer1[count];
82         #endif;
83         #if ea_ldp6416_displays > 2
84             buffer2[count] = draw_buffer2[count];
85         #endif
86         #if ea_ldp6416_displays > 3
87             buffer3[count] = draw_buffer3[count];
88         #endif
89         count++;
90     } while (count !=0);
91     //end crit sec();
92 }
```

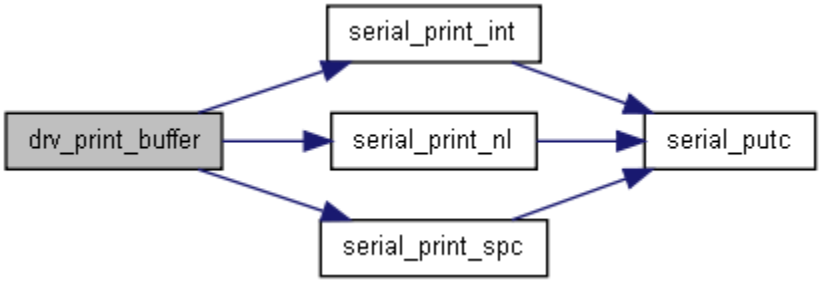
Here is the call graph for this function:



void drv_print_buffer ()

```
94                          {
95 uns8 count;
96
97     count = 0;
98     do {
99         serial_print_int(buffer0[count]);
100         serial_print_spc();
101         count++;
102     } while (count != 0);
103     serial_print_nl();
104
105 }
```

Here is the call graph for this function:



void drv_refresh ()

```
116                     {
117
118
119 uns8 x;
120 uns8 data;
121
122
123 #if ea_ldp6416_max_brightness > 0
124     bright count++;
125
126     if (bright count > bright level) {
127         // Turn off display
128         set pin(ea_ldp6416_en_port, ea_ldp6416_en_pin);
129     }
130
131     if (bright count == ea_ldp6416_max_brightness) {
132         bright count = 255;
133 #endif
134
135         for (x = 0; x < x_refresh; x++) {   // WAS 16
136             #if ea_ldp6416_displays == 1
137                 data = buffer0[buffer position];
138             #endif
139
140             #if ea_ldp6416_displays == 2 && ea_ldp6416_display_orientation == HORIZONTAL
141                 if (current buffer == 0) {
142                     data = buffer0[buffer position];
143                 } else {
144                     data = buffer1[buffer position];
145                 }
146             #endif
147             #if ea_ldp6416_displays == 3 && ea_ldp6416_display_orientation == HORIZONTAL
148                 switch (current buffer) {
149                     case 0: data = buffer0[buffer position]; break;
150                     case 1: data = buffer1[buffer position]; break;
151                     case 2: data = buffer2[buffer position]; break;
152                 }
153             #endif
154             #if ea_ldp6416_displays == 4 && ea_ldp6416_display_orientation == HORIZONTAL
155                 switch (current buffer) {
156                     case 0: data = buffer0[buffer position]; break;
157                     case 1: data = buffer1[buffer position]; break;
158                     case 2: data = buffer2[buffer position]; break;
159                     case 3: data = buffer3[buffer position]; break;
160                 }
161             #endif
162
163             set pins r1 g1();
164
165             if (data.0) { clear pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
166             if (data.1) { clear pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
167
168             clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
```

```
169             set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
170
171             set pins r1 g1();
172
173             if (data.2) { clear pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
174             if (data.3) { clear pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
175
176             clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
177             set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
178
179             set pins r1 g1();
180
181             if (data.4) { clear pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
182             if (data.5) { clear pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
183
184             clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
185             set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
186
187             set pins r1 g1();
188
189             if (data.6) { clear pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
190             if (data.7) { clear pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
191
192             clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
193             set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
194
195             buffer position++;
196
197             #if ea_ldp6416_displays > 1 && ea_ldp6416_display_orientation == HORIZONTAL
198                 if (buffer position == 0) {
199                     current buffer++;
200                 }
201             #endif
202         }
203
204         #if ea_ldp6416_displays == 2 && ea_ldp6416_display_orientation == VERTICAL
205
206             for (x = 0; x < x_refresh; x++) {
207                 data = buffer1[buffer_position1];
208
209                 set pins r1 g1();
210
211                 if (data_upper.0) { clear pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
212                 if (data upper.1) { clear pin(ea ldp6416 g1 port, ea ldp6416 g1 pin); }
213
214                 clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
215                 set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
216
217                 set pins r1 g1();
218
219                 if (data_upper.2) { clear pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
220                 if (data_upper.3) { clear pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
221
222                 clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
223                 set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
224
225                 set pins r1 g1();
226
227                 if (data_upper.4) { clear pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
228                 if (data_upper.5) { clear pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
229
230                 clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
231                 set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
232
233                 set pins r1 g1();
234
235                 if (data_upper.6) { clear pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
236                 if (data_upper.7) { clear pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
237
238                 clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
239                 set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
```

```
240
241                 buffer_position1++;
242             } // x loop
243
244         #endif
245
246
247         set_pin(ea_ldp6416_en_port, ea_ldp6416_en_pin); // turn enable off
248
249         clear_pin(ea_ldp6416_a_port, ea_ldp6416_a_pin);
250         clear_pin(ea_ldp6416_b_port, ea_ldp6416_b_pin);
251         clear_pin(ea_ldp6416_c_port, ea_ldp6416_c_pin);
252         clear_pin(ea_ldp6416_d_port, ea_ldp6416_d_pin);
253
254         if (current_row.0) { set_pin(ea_ldp6416_a_port, ea_ldp6416_a_pin); }
255         if (current_row.1) { set_pin(ea_ldp6416_b_port, ea_ldp6416_b_pin); }
256         if (current_row.2) { set_pin(ea_ldp6416_c_port, ea_ldp6416_c_pin); }
257         if (current_row.3) { set_pin(ea_ldp6416_d_port, ea_ldp6416_d_pin); }
258
259         // latch data
260         set_pin(ea_ldp6416_l_port, ea_ldp6416_l_pin);
261         clear_pin(ea_ldp6416_l_port, ea_ldp6416_l_pin);
262
263         // enable display of row
264         clear_pin(ea_ldp6416_en_port, ea_ldp6416_en_pin);
265
266         current_row++;
267         if (current_row == y_refresh) {
268             current_buffer = 0;
269             buffer_position = 0;
270             current_row = 0;
271         }
272
273 #if ea_ldp6416_max_brightness > 0
274
275     }
276 #endif
277
278 }
```

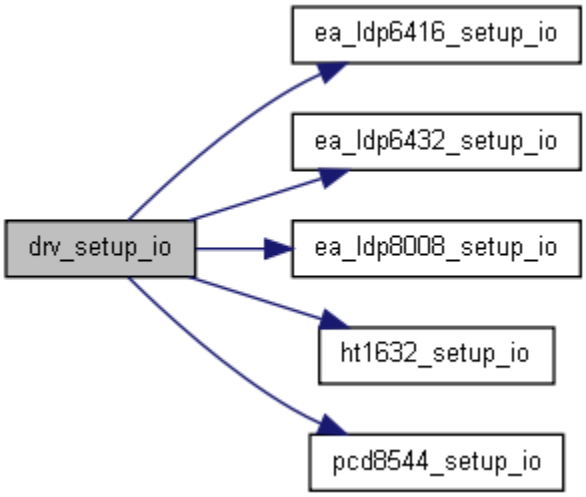
void drv_set_display_brightness (uns8 *brightness*)

```
108                                                 {
109     if (brightness < ea_ldp6416_max_brightness) {
110         bright_level = brightness;
111     } else {
112         bright_level = ea_ldp6416_max_brightness;
113     }
114 }
```

void drv_setup_io ()

```
283                     {
284     ea_ldp6416_setup_io();
285 }
```

Here is the call graph for this function:

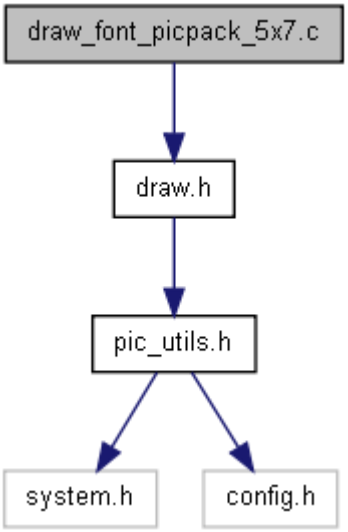


Here is the caller graph for this function:

---

draw_font_picpack_5x7.c File Reference

Simple 5x7 font for Draw library.
Include dependency graph for draw_font_picpack_5x7.c:



Defines

- #define FONT_FIRST_CHAR 32
- #define FONT_LAST_CHAR 128

Variables

- rom char PicPack5x7_bitmap_0 []
- rom char PicPack5x7_bitmap_1 []
- uns16 PicPack5x7_index []

Detailed Description

Define Documentation

#define FONT_FIRST_CHAR 32

#define FONT_LAST_CHAR 128

Variable Documentation

rom char PicPack5x7_bitmap_0[]

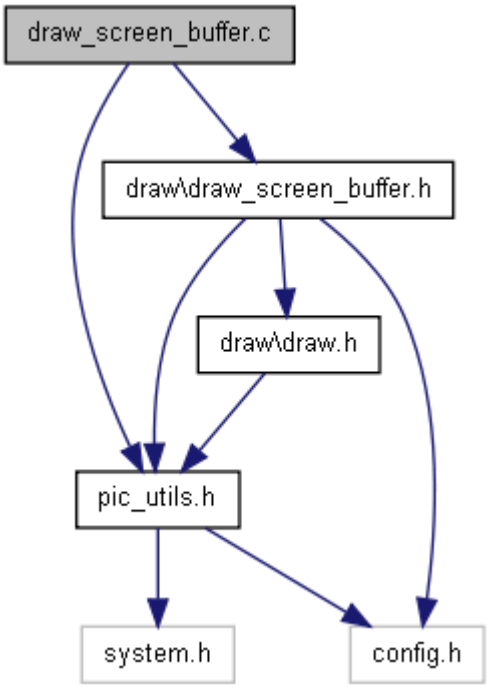
rom char PicPack5x7_bitmap_1[]

uns16 PicPack5x7_index[]

---

draw_screen_buffer.c File Reference

Include dependency graph for draw_screen_buffer.c:



Functions

- uns8 get_draw_buffer (uns16 address)
- void set_draw_buffer (uns16 address, uns8 data)

Variables

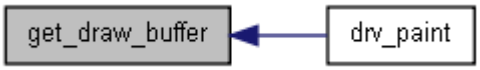
- uns8 draw_buffer0 [DRAW_TOTAL_BUFFER_SIZE]

Function Documentation

uns8 get_draw_buffer (uns16 *address*)

```
33                                      {
34
35      if (address < 256) {
36          return draw_buffer0[address];
37      }
38      #if DRAW_TOTAL_BUFFER_SIZE > 256
39          else if (address < 512) {
40              return draw_buffer1[address - 256];
41          }
42          #if DRAW_TOTAL_BUFFER_SIZE > 512
43              else if (address < 768) {
44                  return draw_buffer2[address - 512];
45              }
46              #if DRAW_TOTAL_BUFFER_SIZE > 768
47                  else if (address < 1024) {
48                      return draw_buffer3[address - 768];
49                  }
50              #endif
51          #endif
52      #endif
53  }
```

Here is the caller graph for this function:



void set_draw_buffer (uns16 *address*, uns8 *data*)

```
55                                                  {
56
57      if (address < 256) {
58          draw_buffer0[address] = data;
59      }
60      #if DRAW_TOTAL_BUFFER_SIZE > 256
61          else if (address < 512) {
62              draw_buffer1[address - 256] = data;
63          }
64          #if DRAW_TOTAL_BUFFER_SIZE > 512
65              else if (address < 768) {
66              draw_buffer2[address - 512] = data;
67              }
68              #if DRAW_TOTAL_BUFFER_SIZE > 768
69                  else if (address < 1024) {
70                      draw_buffer3[address - 768] = data;
71                  }
72              #endif
73          #endif
74      #endif
75  }
```

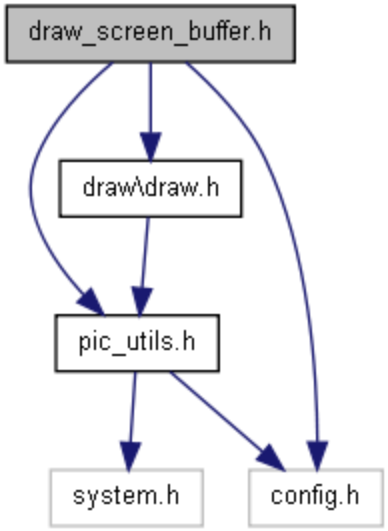
Variable Documentation

uns8 draw_buffer0[DRAW_TOTAL_BUFFER_SIZE]

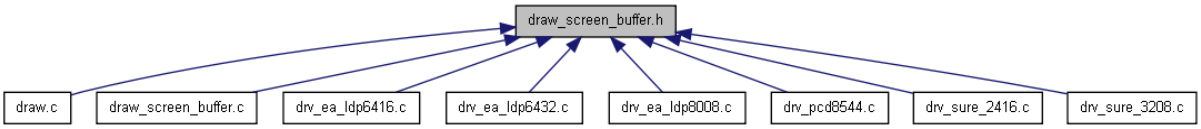
---

draw_screen_buffer.h File Reference

Include dependency graph for draw_screen_buffer.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define DRAW_BUFFERS 1
- #define DRAW_TOTAL_BUFFER_SIZE (DRAW_PIXELS_WIDE * DRAW_PIXELS_HIGH / DRAW_PIXELS_PER_BYTE)

Functions

- uns8 get_draw_buffer (uns16 address)
- void set_draw_buffer (uns16 address, uns8 data)

Variables

- uns8 draw_buffer0 [DRAW_TOTAL_BUFFER_SIZE]

Define Documentation

#define DRAW_BUFFERS 1

#define DRAW_TOTAL_BUFFER_SIZE (DRAW_PIXELS_WIDE * DRAW_PIXELS_HIGH / DRAW_PIXELS_PER_BYTE)

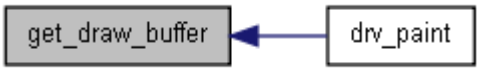
Function Documentation

uns8 get_draw_buffer (uns16 *address*)

```
33                                      {
```

```
34
35      if (address < 256) {
36          return draw buffer0[address];
37      }
38      #if DRAW_TOTAL_BUFFER_SIZE > 256
39          else if (address < 512) {
40              return draw_buffer1[address - 256];
41          }
42          #if DRAW_TOTAL_BUFFER_SIZE > 512
43              else if (address < 768) {
44                  return draw buffer2[address - 512];
45              }
46              #if DRAW_TOTAL_BUFFER_SIZE > 768
47                  else if (address < 1024) {
48                      return draw buffer3[address - 768];
49                  }
50              #endif
51          #endif
52      #endif
53  }
```

Here is the caller graph for this function:



void set_draw_buffer (uns16 *address*, uns8 *data*)

```
55                                                  {
56
57      if (address < 256) {
58          draw buffer0[address] = data;
59      }
60      #if DRAW_TOTAL_BUFFER_SIZE > 256
61          else if (address < 512) {
62              draw buffer1[address - 256] = data;
63          }
64          #if DRAW_TOTAL_BUFFER_SIZE > 512
65              else if (address < 768) {
66              draw_buffer2[address - 512] = data;
67              }
68              #if DRAW_TOTAL_BUFFER_SIZE > 768
69                  else if (address < 1024) {
70                      draw_buffer3[address - 768] = data;
71                  }
72              #endif
73          #endif
74      #endif
75  }
```

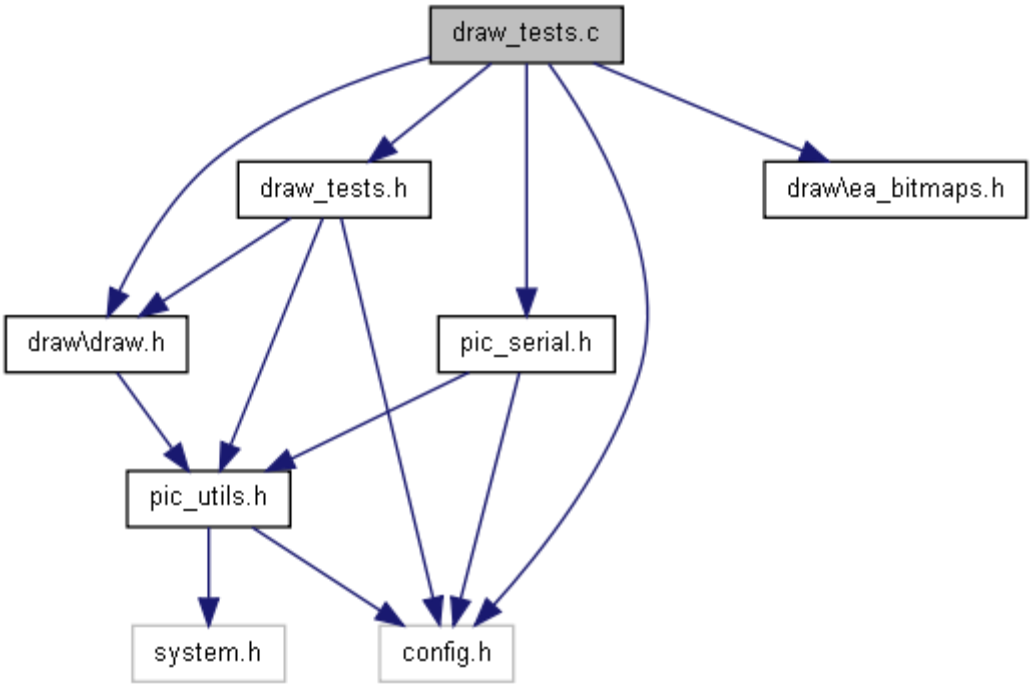
Variable Documentation

uns8 draw_buffer0[DRAW_TOTAL_BUFFER_SIZE]

---

draw_tests.c File Reference

Routines to test the Draw graphics functions.
Include dependency graph for draw_tests.c:



Defines

- #define TEST_RADIUS DRAW_PIXELS_WIDE / 2 -1

Functions

- uns8 draw_tests_run (uns8 test_num)

Detailed Description

Define Documentation

#define TEST_RADIUS DRAW_PIXELS_WIDE / 2 -1

Function Documentation

uns8 draw_tests_run (uns8 *test_num*)

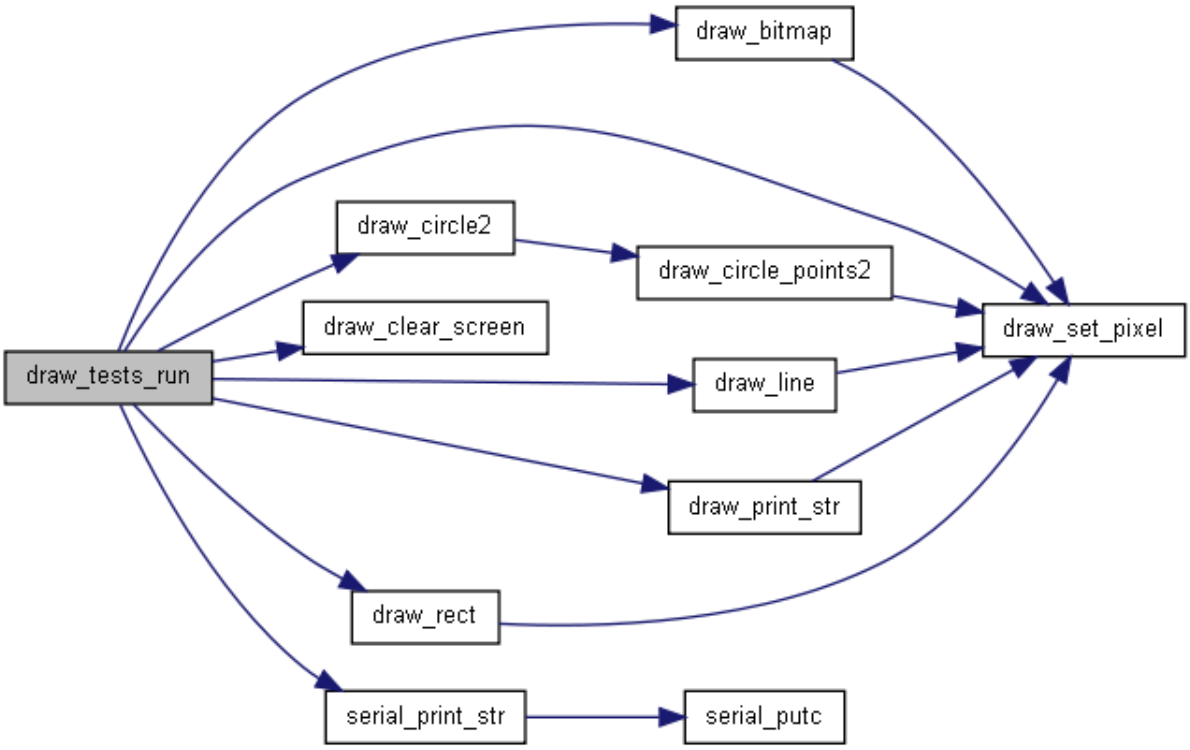
```
46                                      {
47
48 uns8 test_result;
```

```
49      test_result = TEST_RESULT_RAN;
50      draw_clear_screen();
51
52      switch (test_num) {
53          case 0:
54                  serial_print_str("0:Top left pixel\n");
55                  draw_set_pixel(0, DRAW_TOP_PIXEL_Y, 1);
56                  break;
57
58          case 1:
59                  serial_print_str("1:Top right pixel\n");
60                  draw_set_pixel(DRAW_PIXELS_WIDE - 1, DRAW_TOP_PIXEL_Y, 1);
61                  break;
62
63          case 2:
64                  serial_print_str("2:Bottom right pixel\n");
65                  draw_set_pixel(DRAW_PIXELS_WIDE - 1, DRAW_BOTTOM_PIXEL_Y, 1);
66                  break;
67
68          case 3:
69                  serial_print_str("3:Bottom left pixel\n");
70                  draw_set_pixel(0, DRAW_BOTTOM_PIXEL_Y, 1);
71                  break;
72
73          case 4:
74                  serial_print_str("4:Full screen colour 1\n");
75                  draw_rect(0, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE, DRAW_PIXELS_HIGH, 1);
76                  break;
77
78          case 5:
79                  #if DRAW_BITS_PER_PIXEL > 1
80                      serial_print_str("5:Full screen colour 2\n");
81                      draw_rect(0, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE, DRAW_PIXELS_HIGH, 2);
82                  #else
83                      serial_print_str("5:Not applicable\n");
84                      test_result = TEST_RESULT_NOT_APPLICABLE;
85                  #endif
86                  break;
87
88          case 6:
89                  #if DRAW_BITS_PER_PIXEL > 1
90                      serial_print_str("6:Full screen colour 3\n");
91                      draw_rect(0, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE, DRAW_PIXELS_HIGH, 3);
92                           #else
93                      serial_print_str("6:Not applicable\n");
94                      test_result = TEST_RESULT_NOT_APPLICABLE;
95                  #endif
96                  break;
97
98          case 7:
99                  serial_print_str("7:Circle\n");
100                  // circle 2 is centred mid-pixel, good for even number of pixels
101                  #if DRAW_PIXELS_WIDE > DRAW_PIXELS_HIGH
102                      #define TEST_RADIUS DRAW_PIXELS_HIGH / 2 -1
103                  #else
104                      #define TEST_RADIUS DRAW_PIXELS_WIDE / 2 -1
105                  #endif
106                  draw_circle2(DRAW_PIXELS_WIDE / 2 -1, DRAW_PIXELS_HIGH / 2 -1, TEST_RADIUS,
1);
107                  break;
108
109           case 8:
110                  serial_print_str("8:Rect 1 pixel from edges colour 1\n");
111                  #if DRAW_TOP_PIXEL_Y == 0
112                      draw_rect(1, DRAW_TOP_PIXEL_Y + 1, DRAW_PIXELS_WIDE -2, DRAW_PIXELS_HIGH
-2, 1);
113                  #else
114                      draw_rect(1, DRAW_TOP_PIXEL_Y - 1, DRAW_PIXELS_WIDE -2, DRAW_PIXELS_HIGH
-2, 1);
115                  #endif
116                  break;
```

```
117
118         case 9:
119                 #if DRAW_BITS_PER_PIXEL > 1
120                     serial_print_str("9:Rect 1 pixel from edges colour 2\n");
121                     #if DRAW_TOP_PIXEL_Y == 0
122                         draw_rect(1, DRAW_TOP_PIXEL_Y + 1, DRAW_PIXELS_WIDE -2,
DRAW_PIXELS_HIGH -2, 2);
123                     #else
124                         draw_rect(1, DRAW_TOP_PIXEL_Y - 1, DRAW_PIXELS_WIDE -2,
DRAW_PIXELS_HIGH -2, 2);
125                     #endif
126                 #else
127                     serial_print_str("9:Not applicable\n");
128                     test_result = TEST_RESULT_NOT_APPLICABLE;
129                 #endif
130                 break;
131
132         case 10:
133                 #if DRAW_BITS_PER_PIXEL > 1
134                     serial_print_str("10:Rect 1 pixel from edges colour 3\n");
135                     #if DRAW_TOP_PIXEL_Y == 0
136                         draw_rect(1, DRAW_TOP_PIXEL_Y + 1, DRAW_PIXELS_WIDE -2,
DRAW_PIXELS_HIGH -2, 3);
137                     #else
138                         draw_rect(1, DRAW_TOP_PIXEL_Y - 1, DRAW_PIXELS_WIDE -2,
DRAW_PIXELS_HIGH -2, 3);
139                     #endif
140                 #else
141                     serial_print_str("10:Not applicable\n");
142                     test_result = TEST_RESULT_NOT_APPLICABLE;
143                 #endif
144                 break;
145
146         case 11:
147                 serial_print_str("11:Lines around edges\n");
148                 draw_line(0, DRAW_TOP_PIXEL_Y, 0, DRAW_BOTTOM_PIXEL_Y, 3);  // Left side
149                 draw_line(DRAW_PIXELS_WIDE - 1, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE - 1,
DRAW_BOTTOM_PIXEL_Y, 3);    // right side
150                 draw_line(0, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE - 1, DRAW_TOP_PIXEL_Y, 3);
// top
151                 draw_line(0, DRAW_BOTTOM_PIXEL_Y, DRAW_PIXELS_WIDE - 1, DRAW_BOTTOM_PIXEL_Y,
3);    // top
152                 break;
153         case 12:
154                 serial_print_str("12:Static Text\n");
155                 draw_print_str(0, DRAW_BOTTOM_PIXEL_Y, DRAW_PIXELS_WIDE, 0, 1,
"EmbeddedAdventures");
156                 break;
157         case 13:
158                 serial_print_str("13: 8 pixel high bitmaps\n");
159                 draw_bitmap(0, DRAW_TOP_PIXEL_Y, 1, &embedded_bitmap);
160                 #if DRAW_PIXELS_HIGH > 15
161                     #if DRAW_TOP_PIXEL_Y == 0
162                         draw_bitmap(0, DRAW_TOP_PIXEL_Y + 8, 2, &adventures_bitmap);
163                     #else
164                         draw_bitmap(0, DRAW_TOP_PIXEL_Y - 8, 2, &adventures_bitmap);
165                     #endif
166                 #endif
167                 break;
168
169         case 14:
170                 #if DRAW_PIXELS_HIGH > 15
171                     serial_print_str("14: 16 pixel high bitmaps\n");
172                     draw_bitmap(DRAW_PIXELS_WIDE / 2 - 17 , DRAW_TOP_PIXEL_Y, 1, &e_bitmap);
173                     draw_bitmap(DRAW_PIXELS_WIDE / 2, DRAW_TOP_PIXEL_Y, 2, &a_bitmap);
174                 #else
175                     serial_print_str("14:Not applicable\n");
176                 #endif
177                 break;
178         case 15:
179                 #if DRAW_PIXELS_HIGH > 31
```

```
180                     serial_print_str("15: 32 pixel high bitmaps\n");
181                     draw_bitmap(0 , DRAW_TOP_PIXEL_Y, 1, &e_big_bitmap);
182                     draw_bitmap(29, DRAW_TOP_PIXEL_Y, 2, &a_big_bitmap);
183                 #else
184                     serial_print_str("15:Not applicable\n");
185                 #endif
186                 break;
187
188         default:
189                 test_result = TEST_RESULT_NO_MORE_TESTS;
190
191     }
192     draw_paint();
193     return test_result;
194 }
```

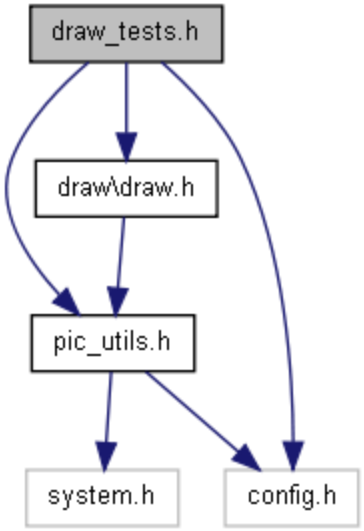
Here is the call graph for this function:



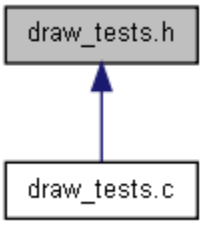
---

draw_tests.h File Reference

Routines to test the Draw graphics functions.
Include dependency graph for draw_tests.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define DRAW_BOTTOM_PIXEL_Y DRAW_PIXELS_HIGH - 1
- #define DRAW_TOP_PIXEL_Y 0
- #define TEST_RESULT_NO_MORE_TESTS 2
- #define TEST_RESULT_NOT_APPLICABLE 1
- #define TEST_RESULT_RAN 0

Functions

- uns8 draw_tests_run (uns8 test_num)

Detailed Description

Define Documentation

#define DRAW_BOTTOM_PIXEL_Y DRAW_PIXELS_HIGH - 1

#define DRAW_TOP_PIXEL_Y 0

#define TEST_RESULT_NO_MORE_TESTS 2

#define TEST_RESULT_NOT_APPLICABLE 1

#define TEST_RESULT_RAN 0

Function Documentation

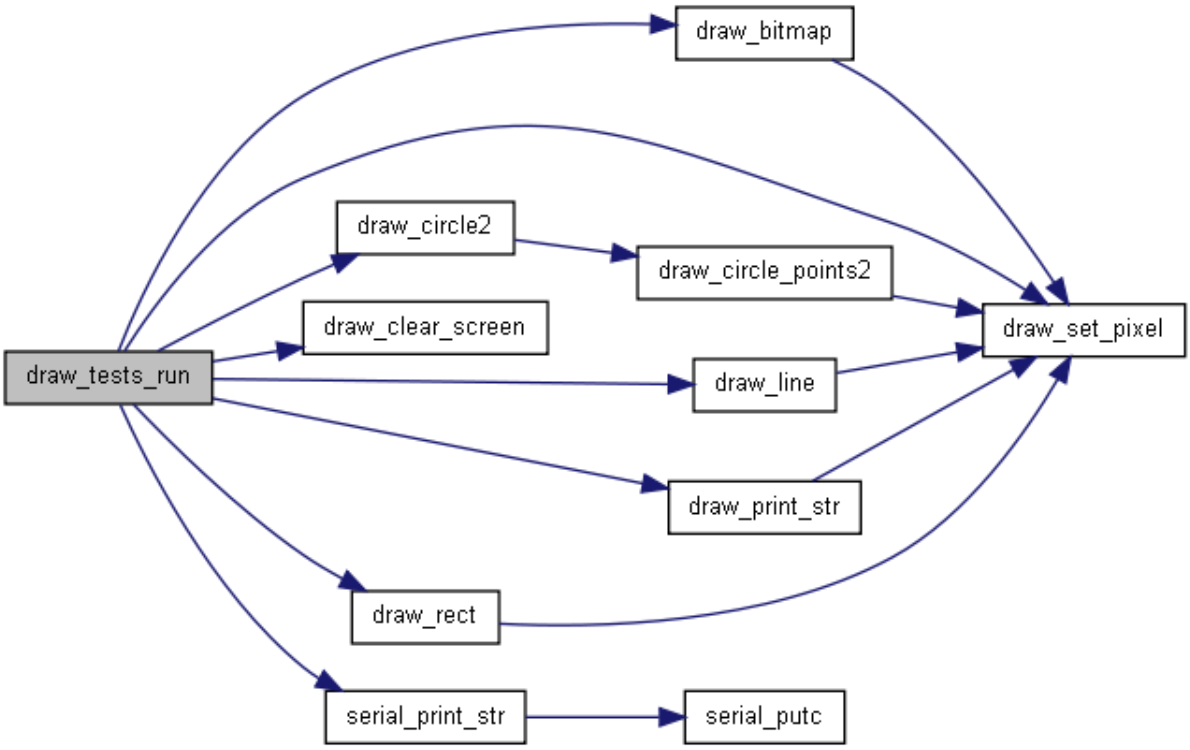
uns8 draw_tests_run (uns8 *test_num*)

```
46                                      {
47
48 uns8 test_result;
49     test_result = TEST_RESULT_RAN;
50     draw_clear_screen();
51
52     switch (test_num) {
53         case 0:
54                 serial_print_str("0:Top left pixel\n");
55                 draw_set_pixel(0, DRAW_TOP_PIXEL_Y, 1);
56                 break;
57
58         case 1:
59                 serial_print_str("1:Top right pixel\n");
60                 draw_set_pixel(DRAW_PIXELS_WIDE - 1, DRAW_TOP_PIXEL_Y, 1);
61                 break;
62
63         case 2:
64                 serial_print_str("2:Bottom right pixel\n");
65                 draw_set_pixel(DRAW_PIXELS_WIDE - 1, DRAW_BOTTOM_PIXEL_Y, 1);
66                 break;
67
68         case 3:
69                 serial_print_str("3:Bottom left pixel\n");
70                 draw_set_pixel(0, DRAW_BOTTOM_PIXEL_Y, 1);
71                 break;
72
73         case 4:
74                 serial_print_str("4:Full screen colour 1\n");
75                 draw_rect(0, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE, DRAW_PIXELS_HIGH, 1);
76                 break;
77
78         case 5:
79                 #if DRAW_BITS_PER_PIXEL > 1
80                     serial_print_str("5:Full screen colour 2\n");
81                     draw_rect(0, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE, DRAW_PIXELS_HIGH, 2);
82                 #else
83                     serial_print_str("5:Not applicable\n");
84                     test_result = TEST_RESULT_NOT_APPLICABLE;
85                 #endif
86                 break;
87
88         case 6:
89                 #if DRAW_BITS_PER_PIXEL > 1
90                     serial_print_str("6:Full screen colour 3\n");
91                     draw_rect(0, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE, DRAW_PIXELS_HIGH, 3);
92                         #else
```

```
93                      serial_print_str("6:Not applicable\n");
94                      test_result = TEST_RESULT_NOT_APPLICABLE;
95                  #endif
96                  break;
97
98          case 7:
99                  serial_print_str("7:Circle\n");
100                 // circle 2 is centred mid-pixel, good for even number of pixels
101                 #if DRAW_PIXELS_WIDE > DRAW_PIXELS_HIGH
102                     #define TEST_RADIUS DRAW_PIXELS_HIGH / 2 -1
103                 #else
104                     #define TEST_RADIUS DRAW_PIXELS_WIDE / 2 -1
105                 #endif
106                 draw_circle2(DRAW_PIXELS_WIDE / 2 -1, DRAW_PIXELS_HIGH / 2 -1, TEST_RADIUS,
1);
107                 break;
108
109         case 8:
110                 serial_print_str("8:Rect 1 pixel from edges colour 1\n");
111                 #if DRAW_TOP_PIXEL_Y == 0
112                     draw_rect(1, DRAW_TOP_PIXEL_Y + 1, DRAW_PIXELS_WIDE -2, DRAW_PIXELS_HIGH
-2, 1);
113                 #else
114                     draw_rect(1, DRAW_TOP_PIXEL_Y - 1, DRAW_PIXELS_WIDE -2, DRAW_PIXELS_HIGH
-2, 1);
115                 #endif
116                 break;
117
118         case 9:
119                 #if DRAW_BITS_PER_PIXEL > 1
120                     serial_print_str("9:Rect 1 pixel from edges colour 2\n");
121                     #if DRAW_TOP_PIXEL_Y == 0
122                         draw_rect(1, DRAW_TOP_PIXEL_Y + 1, DRAW_PIXELS_WIDE -2,
DRAW_PIXELS_HIGH -2, 2);
123                     #else
124                         draw_rect(1, DRAW_TOP_PIXEL_Y - 1, DRAW_PIXELS_WIDE -2,
DRAW_PIXELS_HIGH -2, 2);
125                     #endif
126                 #else
127                     serial_print_str("9:Not applicable\n");
128                     test_result = TEST_RESULT_NOT_APPLICABLE;
129                 #endif
130                 break;
131
132         case 10:
133                 #if DRAW_BITS_PER_PIXEL > 1
134                     serial_print_str("10:Rect 1 pixel from edges colour 3\n");
135                     #if DRAW_TOP_PIXEL_Y == 0
136                         draw_rect(1, DRAW_TOP_PIXEL_Y + 1, DRAW_PIXELS_WIDE -2,
DRAW_PIXELS_HIGH -2, 3);
137                     #else
138                         draw_rect(1, DRAW_TOP_PIXEL_Y - 1, DRAW_PIXELS_WIDE -2,
DRAW_PIXELS_HIGH -2, 3);
139                     #endif
140                 #else
141                     serial_print_str("10:Not applicable\n");
142                     test_result = TEST_RESULT_NOT_APPLICABLE;
143                 #endif
144                 break;
145
146         case 11:
147                 serial_print_str("11:Lines around edges\n");
148                 draw_line(0, DRAW_TOP_PIXEL_Y, 0, DRAW_BOTTOM_PIXEL_Y, 3);  // Left side
149                 draw_line(DRAW_PIXELS_WIDE - 1, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE - 1,
DRAW_BOTTOM_PIXEL_Y, 3);    // right side
150                 draw_line(0, DRAW_TOP_PIXEL_Y, DRAW_PIXELS_WIDE - 1, DRAW_TOP_PIXEL_Y, 3);
// top
151                 draw_line(0, DRAW_BOTTOM_PIXEL_Y, DRAW_PIXELS_WIDE - 1, DRAW_BOTTOM_PIXEL_Y,
3);    // top
152                 break;
153         case 12:
```

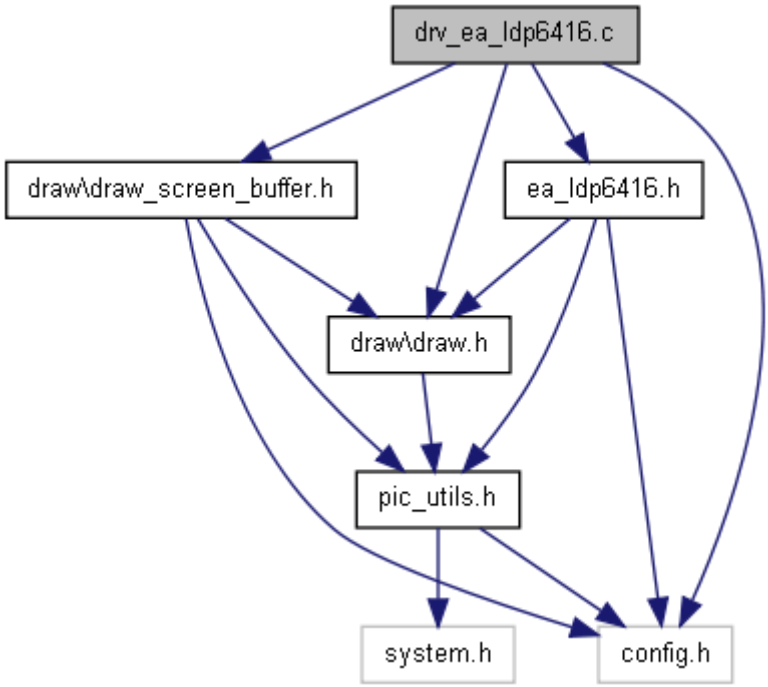
```
154                 serial_print_str("12:Static Text\n");
155                 draw_print_str(0, DRAW_BOTTOM_PIXEL_Y, DRAW_PIXELS_WIDE, 0, 1,
"EmbeddedAdventures");
156                 break;
157         case 13:
158                 serial_print_str("13: 8 pixel high bitmaps\n");
159                 draw_bitmap(0, DRAW_TOP_PIXEL_Y, 1, &embedded_bitmap);
160                 #if DRAW_PIXELS_HIGH > 15
161                     #if DRAW_TOP_PIXEL_Y == 0
162                         draw_bitmap(0, DRAW_TOP_PIXEL_Y + 8, 2, &adventures_bitmap);
163                     #else
164                         draw_bitmap(0, DRAW_TOP_PIXEL_Y - 8, 2, &adventures_bitmap);
165                     #endif
166                 #endif
167                 break;
168
169         case 14:
170                 #if DRAW_PIXELS_HIGH > 15
171                     serial_print_str("14: 16 pixel high bitmaps\n");
172                     draw_bitmap(DRAW_PIXELS_WIDE / 2 - 17 , DRAW_TOP_PIXEL_Y, 1, &e_bitmap);
173                     draw_bitmap(DRAW_PIXELS_WIDE / 2, DRAW_TOP_PIXEL_Y, 2, &a_bitmap);
174                 #else
175                     serial_print_str("14:Not applicable\n");
176                 #endif
177                 break;
178         case 15:
179                 #if DRAW_PIXELS_HIGH > 31
180                     serial_print_str("15: 32 pixel high bitmaps\n");
181                     draw_bitmap(0 , DRAW_TOP_PIXEL_Y, 1, &e_big_bitmap);
182                     draw_bitmap(29, DRAW_TOP_PIXEL_Y, 2, &a_big_bitmap);
183                 #else
184                     serial_print_str("15:Not applicable\n");
185                 #endif
186                 break;
187
188         default:
189                 test_result = TEST_RESULT_NO_MORE_TESTS;
190
191     }
192     draw_paint();
193     return test_result;
194 }
```

Here is the call graph for this function:



drv_ea_ldp6416.c File Reference

Draw drivers for Embedded Adventures LDP-6416 LED panel and similar.
Include dependency graph for drv_ea_ldp6416.c:



Defines

- #define set_pins_r1_g1()

Functions

- uns8 drv_get_pixel (uns8 x, uns8 y)
- void drv_init ()
- void drv_paint ()
- void drv_print_buffer ()
- void drv_refresh ()
- void drv_set_display_brightness (uns8 brightness)
- void drv_setup_io ()

Variables

- uns8 bright_count = 0
- uns8 bright_level = 0
- char buffer0 [256]
- uns8 buffer_position = 0
- uns8 current_buffer = 0
- uns8 current_row = 0

Detailed Description

Define Documentation

#define set_pins_r1_g1()

```
Value:set_pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); \
    set_pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin);
```

Function Documentation

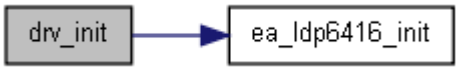
uns8 drv_get_pixel (uns8 *x*, uns8 *y*)

```
279                                     {
280     //Should do something to look up the pixel
281 }
```

void drv_init ()

```
287                 {
288     ea_ldp6416_init();
289 }
```

Here is the call graph for this function:



void drv_paint ()

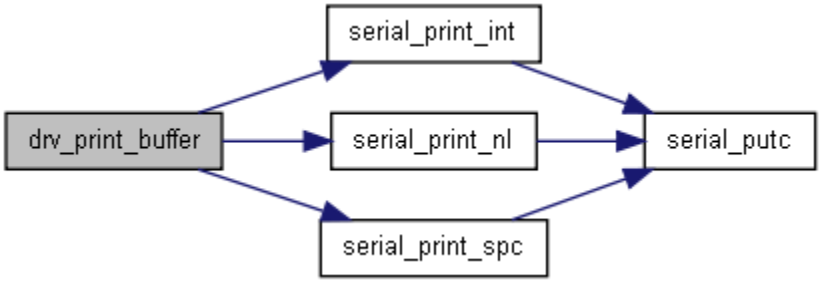
```
73                  {
74     uns8 count;
75     // start_crit_sec();
76     count = 0;
77
78     do {
79         buffer0[count] = draw_buffer0[count];
80         #if ea_ldp6416_displays > 1
81             buffer1[count] = draw_buffer1[count];
82         #endif;
83         #if ea_ldp6416_displays > 2
84             buffer2[count] = draw_buffer2[count];
85         #endif
86         #if ea_ldp6416_displays > 3
87             buffer3[count] = draw_buffer3[count];
88         #endif
89         count++;
90     } while (count !=0);
91     //end_crit_sec();
92 }
```

void drv_print_buffer ()

```
94                          {
95 uns8 count;
96
97     count = 0;
98     do {
99         serial_print_int(buffer0[count]);
```

```
100         serial_print_spc();
101         count++;
102     } while (count != 0);
103     serial_print_nl();
104 
105 }
```

Here is the call graph for this function:



void drv_refresh ()

```
116                     {
117 
118 
119 uns8 x;
120 uns8 data;
121 
122 
123 #if ea_ldp6416_max_brightness > 0
124     bright_count++;
125 
126     if (bright_count > bright_level) {
127         // Turn off display
128         set_pin(ea_ldp6416_en_port, ea_ldp6416_en_pin);
129     }
130 
131     if (bright_count == ea_ldp6416_max_brightness) {
132         bright_count = 255;
133 #endif
134 
135         for (x = 0; x < x_refresh; x++) {   // WAS 16
136             #if ea_ldp6416_displays == 1
137                 data = buffer0[buffer_position];
138             #endif
139 
140             #if ea_ldp6416_displays == 2 && ea_ldp6416_display_orientation == HORIZONTAL
141                 if (current_buffer == 0) {
142                     data = buffer0[buffer_position];
143                 } else {
144                     data = buffer1[buffer_position];
145                 }
146             #endif
147             #if ea_ldp6416_displays == 3 && ea_ldp6416_display_orientation == HORIZONTAL
148                 switch (current_buffer) {
149                     case 0: data = buffer0[buffer_position]; break;
150                     case 1: data = buffer1[buffer_position]; break;
151                     case 2: data = buffer2[buffer_position]; break;
152                 }
153             #endif
154             #if ea_ldp6416_displays == 4 && ea_ldp6416_display_orientation == HORIZONTAL
155                 switch (current_buffer) {
156                     case 0: data = buffer0[buffer_position]; break;
157                     case 1: data = buffer1[buffer_position]; break;
158                     case 2: data = buffer2[buffer_position]; break;
159                     case 3: data = buffer3[buffer_position]; break;
```

```
160                 }
161             #endif
162
163             set_pins_r1_g1();
164
165             if (data.0) { clear_pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
166             if (data.1) { clear_pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
167
168             clear_pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
169             set_pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
170
171             set_pins_r1_g1();
172
173             if (data.2) { clear_pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
174             if (data.3) { clear_pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
175
176             clear_pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
177             set_pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
178
179             set_pins_r1_g1();
180
181             if (data.4) { clear_pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
182             if (data.5) { clear_pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
183
184             clear_pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
185             set_pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
186
187             set_pins_r1_g1();
188
189             if (data.6) { clear_pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
190             if (data.7) { clear_pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
191
192             clear_pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
193             set_pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
194
195             buffer_position++;
196
197             #if ea_ldp6416_displays > 1 && ea_ldp6416_display_orientation == HORIZONTAL
198                 if (buffer_position == 0) {
199                     current_buffer++;
200                 }
201             #endif
202         }
203
204         #if ea_ldp6416_displays == 2 && ea_ldp6416_display_orientation == VERTICAL
205
206             for (x = 0; x < x_refresh; x++) {
207                 data = buffer1[buffer_position1];
208
209                 set_pins_r1_g1();
210
211                 if (data_upper.0) { clear_pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
212                 if (data_upper.1) { clear_pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
213
214                 clear_pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
215                 set_pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
216
217                 set_pins_r1_g1();
218
219                 if (data_upper.2) { clear_pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
220                 if (data_upper.3) { clear_pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
221
222                 clear_pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
223                 set_pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
224
225                 set_pins_r1_g1();
226
227                 if (data_upper.4) { clear_pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
228                 if (data_upper.5) { clear_pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
229
230                 clear_pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
```

```
231                 set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
232
233                 set pins r1 g1();
234
235                 if (data_upper.6) { clear pin(ea_ldp6416_r1_port, ea_ldp6416_r1_pin); }
236                 if (data_upper.7) { clear pin(ea_ldp6416_g1_port, ea_ldp6416_g1_pin); }
237
238                 clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
239                 set pin   (ea_ldp6416_s_port, ea_ldp6416_s_pin);
240
241                 buffer position1++;
242             } // x loop
243
244         #endif
245
246
247         set pin(ea_ldp6416_en_port, ea_ldp6416_en_pin); // turn enable off
248
249         clear pin(ea_ldp6416_a_port, ea_ldp6416_a_pin);
250         clear pin(ea_ldp6416_b_port, ea_ldp6416_b_pin);
251         clear pin(ea_ldp6416_c_port, ea_ldp6416_c_pin);
252         clear pin(ea_ldp6416_d_port, ea_ldp6416_d_pin);
253
254         if (current row.0) { set pin(ea_ldp6416_a_port, ea_ldp6416_a_pin); }
255         if (current row.1) { set pin(ea_ldp6416_b_port, ea_ldp6416_b_pin); }
256         if (current row.2) { set pin(ea_ldp6416_c_port, ea_ldp6416_c_pin); }
257         if (current row.3) { set pin(ea_ldp6416_d_port, ea_ldp6416_d_pin); }
258
259         // latch data
260         set pin(ea_ldp6416_l_port, ea_ldp6416_l_pin);
261         clear pin(ea_ldp6416_l_port, ea_ldp6416_l_pin);
262
263         // enable display of row
264         clear pin(ea_ldp6416_en_port, ea_ldp6416_en_pin);
265
266         current row++;
267         if (current row == y_refresh) {
268             current buffer = 0;
269             buffer position = 0;
270             current row = 0;
271         }
272
273 #if ea_ldp6416_max_brightness > 0
274
275     }
276 #endif
277
278 }
```

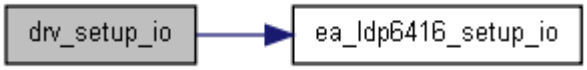
void drv_set_display_brightness (uns8 *brightness*)

```
108                                                 {
109     if (brightness < ea_ldp6416_max_brightness) {
110         bright level = brightness;
111     } else {
112         bright level = ea_ldp6416_max_brightness;
113     }
114 }
```

void drv_setup_io ()

```
283                     {
284     ea ldp6416 setup io();
285 }
```

Here is the call graph for this function:



Variable Documentation

uns8 bright_count = 0

uns8 bright_level = 0

char buffer0[256]

uns8 buffer_position = 0

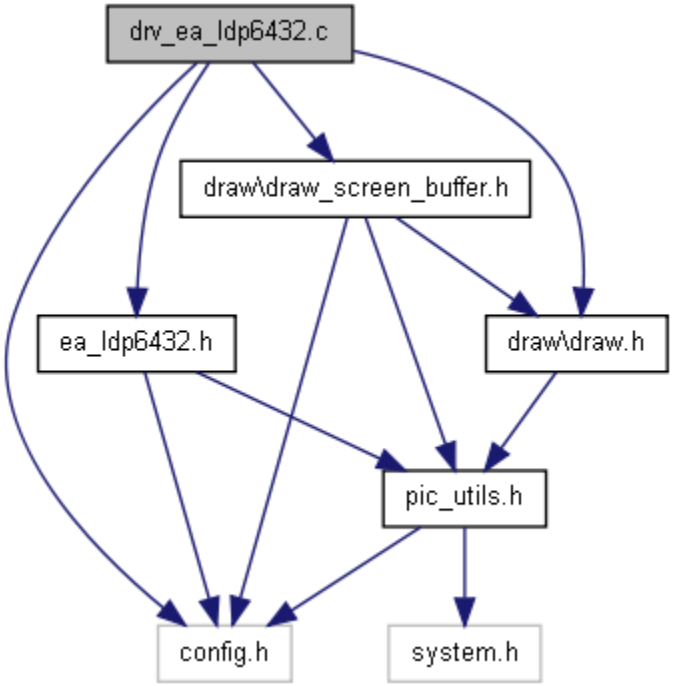
uns8 current_buffer = 0

uns8 current_row = 0

---

drv_ea_ldp6432.c File Reference

Draw drivers for Embedded Adventures LDP-6432 LED panel and similar.
Include dependency graph for drv_ea_ldp6432.c:



Defines

- #define MAX_BRIGHTNESS 3
- #define set_pins_r1_g1_r2_g2()

Functions

- void drv_clear_screen ()
- uns8 drv_get_pixel (uns8 x, uns8 y)
- void drv_init ()
- void drv_paint ()
- void drv_print_buffer ()
- void drv_refresh ()
- void drv_set_display_brightness (uns8 brightness)
- void drv_setup_io ()

Variables

- uns8 bright_count = 0
- uns8 bright_level = 3
- char buffer0 [256]
- char buffer1 [256]
- uns8 buffer_position0 = 0
- uns8 current_row = 0

Detailed Description

Define Documentation

#define MAX_BRIGHTNESS 3

#define set_pins_r1_g1_r2_g2()

```
Value:set_pin(ea_ldp6432_r1_port, ea_ldp6432_r1_pin); \
    set_pin(ea_ldp6432_r2_port, ea_ldp6432_r2_pin); \
    set_pin(ea_ldp6432_g1_port, ea_ldp6432_g1_pin); \
    set_pin(ea_ldp6432_g2_port, ea_ldp6432_g2_pin)
```

Function Documentation

void drv_clear_screen ()

```
331                          {
332     //ht1632_clear_screen();
333 }
```

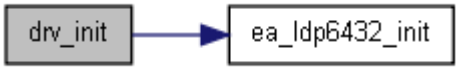
uns8 drv_get_pixel (uns8 *x*, uns8 *y*)

```
327                                     {
328     //ht1632_get_pixel(x, y);
329 }
```

void drv_init ()

```
339                     {
340     ea_ldp6432_init();
341 }
```

Here is the call graph for this function:



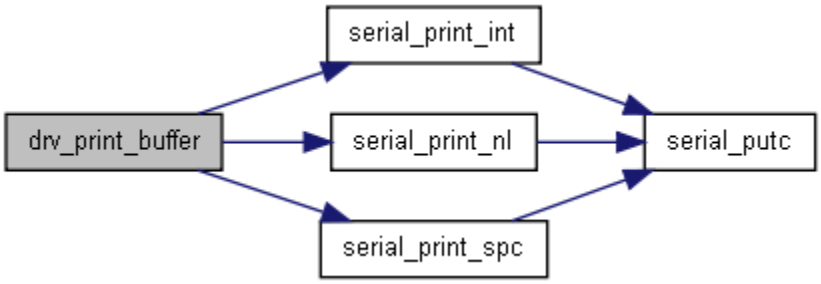
void drv_paint ()

```
74                      {
75     uns8 count;
76
77     count = 0;
78
79     do {
80         buffer0[count] = draw_buffer0[count];
81         buffer1[count] = draw_buffer1[count];
82         #if ea_ldp6432_displays == 2
83             buffer2[count] = draw_buffer2[count];
84             buffer3[count] = draw_buffer3[count];
85         #endif
86         count++;
87     } while (count !=0);
88 }
```

void drv_print_buffer ()

```
312                         {
313 uns8 count;
314
315     count = 0;
316     do {
317         serial_print_int(buffer0[count]);
318         serial_print_spc();
319         count++;
320     } while (count != 0);
321     serial_print_nl();
322
323 }
```

Here is the call graph for this function:



void drv_refresh ()

```
98                      {
99
100
101 uns8 x;
102 uns8 data_upper, data_lower;
103
104
105     bright count++;
106
107     if (bright count > bright level) {
108         // Turn off display
109         set pin(ea_ldp6432_en_port, ea_ldp6432_en_pin);
110     }
111
112     if (bright count == MAX BRIGHTNESS) {
113         bright count = 255;
114
115
116
117         for (x = 0; x < 16; x++) {
118             data_upper = buffer0[buffer position0];
119             data_lower = buffer1[buffer position0];
120
121             set pins r1 g1 r2 g2();
122
123             if (data_upper.0) {
124                 clear pin(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
125             }
126             if (data_upper.1) {
127                 clear pin(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
128             }
129             if (data_lower.0) {
130                 clear pin(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
131             }
132             if (data_lower.1) {
133                 clear pin(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
134             }
135
136             clear pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
137             set pin  (ea_ldp6432_s_port, ea_ldp6432_s_pin);
138
139             set pins r1 g1 r2 g2();
140
141             if (data_upper.2) {
142                 clear pin(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
143             }
144             if (data_upper.3) {
145                 clear pin(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
146             }
147             if (data_lower.2) {
148                 clear pin(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
149             }
150             if (data_lower.3) {
151                 clear pin(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
152             }
153
154             clear pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
155             set pin  (ea_ldp6432_s_port, ea_ldp6432_s_pin);
156
157             set pins r1 g1 r2 g2();
158
159             if (data_upper.4) {
160                 clear pin(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
161             }
162             if (data_upper.5) {
163                 clear pin(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
164             }
165             if (data_lower.4) {
```

```
166                 clear pin(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
167             }
168             if (data_lower.5) {
169                 clear pin(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
170             }
171
172             clear pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
173             set pin   (ea_ldp6432_s_port, ea_ldp6432_s_pin);
174
175             set pins r1 g1 r2 g2();
176
177
178             if (data_upper.6) {
179                 clear pin(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
180             }
181             if (data_upper.7) {
182                 clear pin(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
183             }
184             if (data_lower.6) {
185                 clear pin(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
186             }
187             if (data_lower.7) {
188                 clear pin(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
189             }
190
191             clear pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
192             set pin   (ea_ldp6432_s_port, ea_ldp6432_s_pin);
193
194             buffer position0++;
195         }
196
197             #if ea ldp6432 displays == 2
198                 for (x = 0; x < 16; x++) {
199                     data_upper = buffer2[buffer_position1];
200                     data_lower = buffer3[buffer_position1];
201
202                     set pins r1 g1 r2 g2();
203
204                     if (data_upper.0) {
205                         clear pin(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
206                     }
207                     if (data_upper.1) {
208                         clear pin(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
209                     }
210                     if (data_lower.0) {
211                         clear pin(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
212                     }
213                     if (data_lower.1) {
214                         clear pin(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
215                     }
216
217                     clear pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
218                     set pin   (ea_ldp6432_s_port, ea_ldp6432_s_pin);
219
220                     set pins r1 g1 r2 g2();
221
222                     if (data_upper.2) {
223                         clear pin(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
224                     }
225                     if (data_upper.3) {
226                         clear pin(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
227                     }
228                     if (data_lower.2) {
229                         clear pin(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
230                     }
231                     if (data_lower.3) {
232                         clear pin(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
233                     }
234
235                     clear pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
236                     set pin   (ea_ldp6432_s_port, ea_ldp6432_s_pin);
```

```
237
238                     set_pins_r1_g1_r2_g2();
239
240                     if (data_upper.4) {
241                         clear_pin(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
242                     }
243                     if (data_upper.5) {
244                         clear_pin(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
245                     }
246                     if (data_lower.4) {
247                         clear_pin(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
248                     }
249                     if (data_lower.5) {
250                         clear_pin(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
251                     }
252
253                     clear_pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
254                     set_pin  (ea_ldp6432_s_port, ea_ldp6432_s_pin);
255
256                     set_pins_r1_g1_r2_g2();
257
258
259                     if (data_upper.6) {
260                         clear_pin(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
261                     }
262                     if (data_upper.7) {
263                         clear_pin(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
264                     }
265                     if (data_lower.6) {
266                         clear_pin(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
267                     }
268                     if (data_lower.7) {
269                         clear_pin(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
270                     }
271
272                     clear_pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
273                     set_pin  (ea_ldp6432_s_port, ea_ldp6432_s_pin);
274
275                     buffer_position1++;
276                 } // x loop
277
278             #endif
279
280
281         set_pin(ea_ldp6432_en_port, ea_ldp6432_en_pin); // turn enable off
282
283         clear_pin(ea_ldp6432_a_port, ea_ldp6432_a_pin);
284         clear_pin(ea_ldp6432_b_port, ea_ldp6432_b_pin);
285         clear_pin(ea_ldp6432_c_port, ea_ldp6432_c_pin);
286         clear_pin(ea_ldp6432_d_port, ea_ldp6432_d_pin);
287         if (current_row.0) { set_pin(ea_ldp6432_a_port, ea_ldp6432_a_pin); }
288         if (current_row.1) { set_pin(ea_ldp6432_b_port, ea_ldp6432_b_pin); }
289         if (current_row.2) { set_pin(ea_ldp6432_c_port, ea_ldp6432_c_pin); }
290         if (current_row.3) { set_pin(ea_ldp6432_d_port, ea_ldp6432_d_pin); }
291
292         // latch data
293         set_pin(ea_ldp6432_l_port, ea_ldp6432_l_pin);
294         clear_pin(ea_ldp6432_l_port, ea_ldp6432_l_pin);
295
296         // enable display of line
297         clear_pin(ea_ldp6432_en_port, ea_ldp6432_en_pin);
298
299         current_row++;
300
301         if (current_row == 16) {
302             buffer_position0 = 0;
303             #if ea_ldp6432_displays == 2
304                 buffer_position1 = 0;
305             #endif
306             current_row = 0;
307         }
```

```
308
309     }
310 }
```

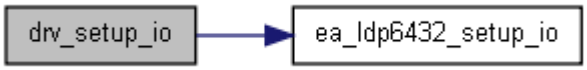
void drv_set_display_brightness (uns8 *brightness*)

```
90                                                  {
91     if (brightness < MAX_BRIGHTNESS) {
92         bright_level = brightness;
93     } else {
94         bright_level = MAX_BRIGHTNESS;
95     }
96 }
```

void drv_setup_io ()

```
335                     {
336     ea_ldp6432_setup_io();
337 }
```

Here is the call graph for this function:



Variable Documentation

uns8 bright_count = 0

uns8 bright_level = 3

char buffer0[256]

char buffer1[256]

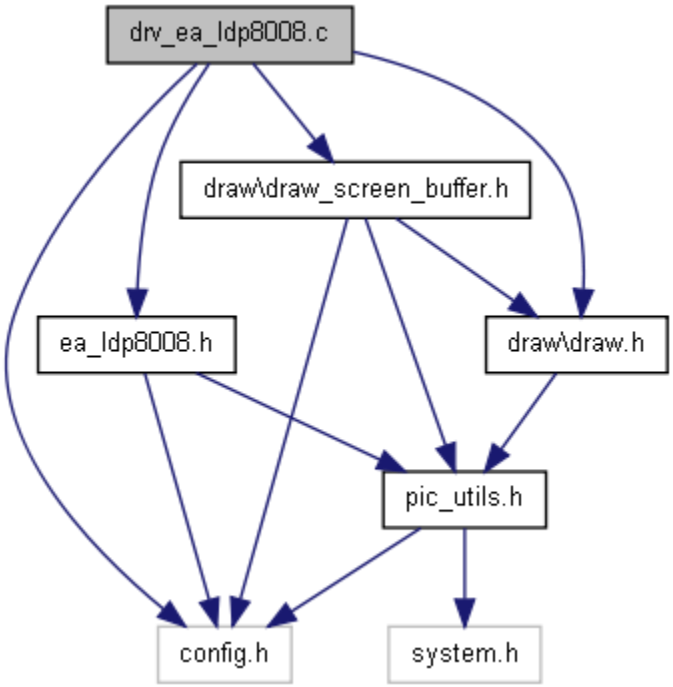
uns8 buffer_position0 = 0

uns8 current_row = 0

---

drv_ea_ldp8008.c File Reference

Draw drivers for Embedded Adventures LDP-8008 LED panel and similar.
Include dependency graph for drv_ea_ldp8008.c:



Defines

- #define MAX_BRIGHTNESS 3
- #define set_pins_r_g()

Functions

- uns8 drv_get_pixel (uns8 x, uns8 y)
- void drv_init ()
- void drv_paint ()
- void drv_print_buffer ()
- void drv_refresh ()
- void drv_set_display_brightness (uns8 brightness)
- void drv_setup_io ()

Variables

- uns8 bright_count = 0
- uns8 bright_level = 3
- char buffer0 [256]
- uns8 buffer_position0 = 0
- uns8 current_row = 0

Detailed Description

Define Documentation

#define MAX_BRIGHTNESS 3

#define set_pins_r_g()

```
Value:set_pin(ea_ldp8008_r_port, ea_ldp8008_r_pin); \
    set_pin(ea_ldp8008_g_port, ea_ldp8008_g_pin);
```

Function Documentation

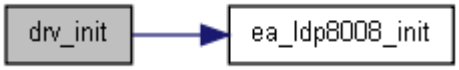
uns8 drv_get_pixel (uns8 *x*, uns8 *y*)

```
290                                     {
291     //Should do something to look up the pixel
292 }
```

void drv_init ()

```
298                 {
299     ea_ldp8008_init();
300 }
```

Here is the call graph for this function:



void drv_paint ()

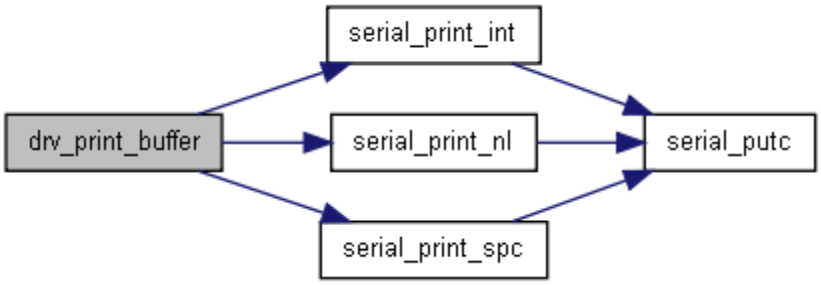
```
72                  {
73     uns8 count;
74     start_crit_sec();
75     count = 0;
76
77     do {
78         buffer0[count] = draw_buffer0[count];
79         #if ea_ldp8008_displays == 2
80             buffer1[count] = draw_buffer1[count];
81         #endif;
82         #if ea_ldp8008_displays == 3
83             buffer2[count] = draw_buffer2[count];
84         #endif
85         #if ea_ldp8008_displays == 4
86             buffer3[count] = draw_buffer3[count];
87         #endif
88         count++;
89     } while (count !=160);
90     end_crit_sec();
91 }
```

void drv_print_buffer ()

```
93                          {
94 uns8 count;
95
```

```
96      count = 0;
97      do {
98          serial print int(buffer0[count]);
99          serial print spc();
100          count++;
101      } while (count != 0);
102      serial print nl();
103
104 }
```

Here is the call graph for this function:



void drv_refresh ()

```
115                     {
116
117
118 uns8 x;
119 uns8 data;
120
121     bright count++;
122
123     if (bright count > bright level) {
124         // Turn off display
125         clear pin(ea_ldp8008_en_port, ea_ldp8008_en_pin);
126     }
127     //if (bright count == 1) {
128     //  bright_count = 0;
129     //}
130     if (bright count == MAX BRIGHTNESS) {
131         bright count = 255;
132
133
134     // Start upper at the top
135
136     //  count_lower = 64 * 2 * 16 / 8;  // Start lower at the second half
137     #if ea_ldp8008_displays == 2
138     #warning "This doesn't work!!"
139     //      uns8 count byte disp2 = 0;
140         //uns16 count_upper_disp2, count_lower_disp2;
141     //  count_upper_disp2 = (64 * 2 * 16 / 8) * 2;
142     //  count_lower_disp2 = (64 * 2 * 16 / 8) * 3;
143     #endif
144
145     for (x = 0; x < 20; x++) {  // 20*8bits/2bpp = 80 pixels across
146         data = buffer0[buffer position0];
147
148         set pins r g();
149
150         if (data.0) {
151             clear pin(ea_ldp8008_r_port, ea_ldp8008_r_pin);
152         }
153         if (data.1) {
154             clear pin(ea_ldp8008_g_port, ea_ldp8008_g_pin);
155         }
```

```
156
157         clear pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
158         set pin  (ea_ldp8008_s_port, ea_ldp8008_s_pin);
159
160         set pins r g();
161
162         if (data.2) {
163             clear pin(ea_ldp8008_r_port, ea_ldp8008_r_pin);
164         }
165         if (data.3) {
166             clear pin(ea_ldp8008_g_port, ea_ldp8008_g_pin);
167         }
168
169         clear pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
170         set pin  (ea_ldp8008_s_port, ea_ldp8008_s_pin);
171
172         set pins r g();
173
174         if (data.4) {
175             clear pin(ea_ldp8008_r_port, ea_ldp8008_r_pin);
176         }
177         if (data.5) {
178             clear pin(ea_ldp8008_g_port, ea_ldp8008_g_pin);
179         }
180
181         clear pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
182         set pin  (ea_ldp8008_s_port, ea_ldp8008_s_pin);
183
184         set pins r g();
185
186
187         if (data.6) {
188             clear pin(ea_ldp8008_r_port, ea_ldp8008_r_pin);
189         }
190         if (data.7) {
191             clear pin(ea_ldp8008_g_port, ea_ldp8008_g_pin);
192         }
193
194         clear pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
195         set pin  (ea_ldp8008_s_port, ea_ldp8008_s_pin);
196
197         buffer_position0++;
198     }
199
200 #if ea_ldp8008_displays == 2
201     #warning "need to work on this"
202     for (x = 0; x < 16; x++) {
203         data_upper = buffer2[count_byte_disp2];
204         data_lower = buffer3[count_byte_disp2];
205
206         set pins r g();
207
208         if (data_upper.0) {
209             clear pin(ea_ldp8008_r_port, ea_ldp8008_r_pin);
210         }
211         if (data_upper.1) {
212             clear pin(ea_ldp8008_g_port, ea_ldp8008_g_pin);
213         }
214
215         clear pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
216         set pin  (ea_ldp8008_s_port, ea_ldp8008_s_pin);
217
218         set pins r g();
219
220         if (data_upper.2) {
221             clear pin(ea_ldp8008_r_port, ea_ldp8008_r_pin);
222         }
223         if (data_upper.3) {
224             clear pin(ea_ldp8008_g_port, ea_ldp8008_g_pin);
225         }
226
```

```
227         clear pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
228         set pin   (ea_ldp8008_s_port, ea_ldp8008_s_pin);
229
230         set pins r g();
231
232         if (data_upper.4) {
233             clear pin(ea_ldp8008_r_port, ea_ldp8008_r_pin);
234         }
235         if (data_upper.5) {
236             clear pin(ea_ldp8008_g_port, ea_ldp8008_g_pin);
237         }
238
239         clear pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
240         set pin   (ea_ldp8008_s_port, ea_ldp8008_s_pin);
241
242         set pins r g();
243
244
245         if (data_upper.6) {
246             clear pin(ea_ldp8008_r_port, ea_ldp8008_r_pin);
247         }
248         if (data_upper.7) {
249             clear pin(ea_ldp8008_g_port, ea_ldp8008_g_pin);
250         }
251
252         clear pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
253         set pin   (ea_ldp8008_s_port, ea_ldp8008_s_pin);
254
255         count_byte_disp2++;
256     } // x loop
257
258 #endif
259
260
261     clear pin(ea_ldp8008_en_port, ea_ldp8008_en_pin);   // turn enable off
262
263     clear pin(ea_ldp8008_a_port, ea_ldp8008_a_pin);
264     clear pin(ea_ldp8008_b_port, ea_ldp8008_b_pin);
265     clear pin(ea_ldp8008_c_port, ea_ldp8008_c_pin);
266     if (current_row.0) {
267         set pin(ea_ldp8008_a_port, ea_ldp8008_a_pin);
268     }
269     if (current_row.1) {
270         set pin(ea_ldp8008_b_port, ea_ldp8008_b_pin);
271     }
272     if (current_row.2) {
273         set pin(ea_ldp8008_c_port, ea_ldp8008_c_pin);
274     }
275     // latch data
276     set pin(ea_ldp8008_l_port, ea_ldp8008_l_pin);
277     clear pin(ea_ldp8008_l_port, ea_ldp8008_l_pin);
278
279     // enable display of row
280     set pin(ea_ldp8008_en_port, ea_ldp8008_en_pin);
281
282     current_row++;
283     if (current_row == 8) {
284         buffer_position0 = 0;
285         current_row = 0;
286     }
287 }
288
289 }
```

void drv_set_display_brightness (uns8 *brightness*)

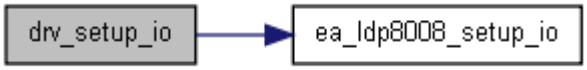
```
107                                                 {
108     if (brightness < MAX_BRIGHTNESS) {
```

```
109         bright_level = brightness;
110     } else {
111         bright_level = MAX_BRIGHTNESS;
112     }
113 }
```

void drv_setup_io ()

```
294                     {
295     ea_ldp8008_setup_io();
296 }
```

Here is the call graph for this function:



Variable Documentation

uns8 bright_count = 0

uns8 bright_level = 3

char buffer0[256]

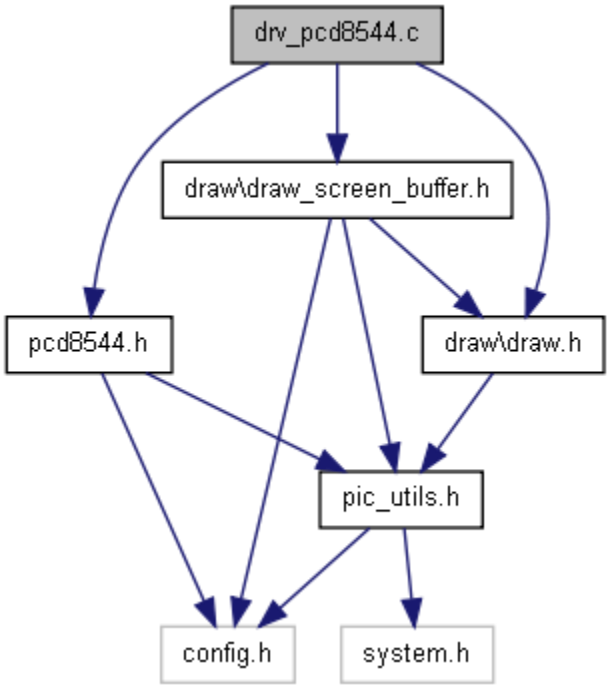
uns8 buffer_position0 = 0

uns8 current_row = 0

---

drv_pcd8544.c File Reference

Draw drivers for PCD8544 based LCD display (Nokia 3310).
Include dependency graph for drv_pcd8544.c:



Functions

- void drv_init ()
- void drv_paint ()
- void drv_setup_io ()

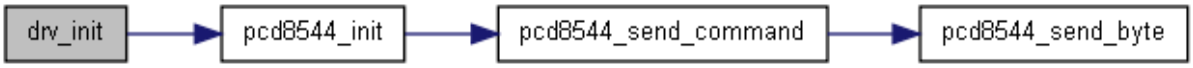
Detailed Description

Function Documentation

void drv_init ()

```
89                  {
90      pcd8544_init();
91  }
```

Here is the call graph for this function:

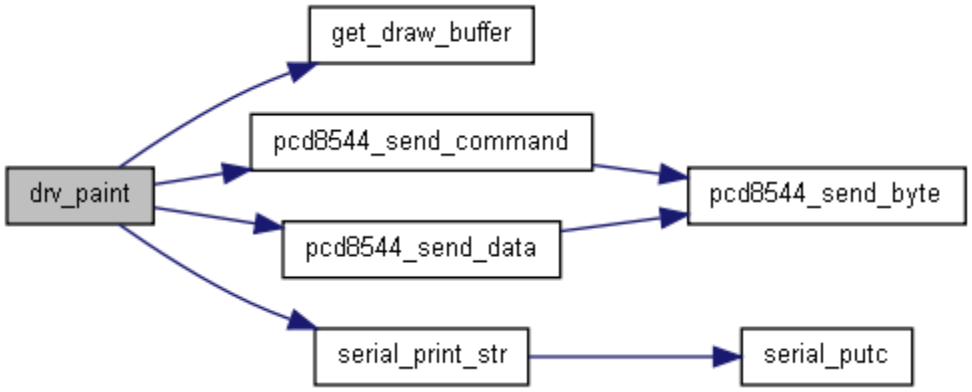


void drv_paint ()

```
45                  {
46
47  uns8 count;
```

```
48 uns8 x, y, inv_y;
49 uns16 buffer_loc;
50 uns16 byte_loc;
51 uns8  bit_loc, byte_out;
52
53 serial_print_str("paint! ");
54
55 // set x,y location to 0
56 pcd8544_send_command(0x80);
57
58 pcd8544_send_command(0x40);
59
60
61 byte_out = 0;
62
63 for(x = 0 ; x < DRAW_PIXELS_WIDE  ; x++) {
64
65     for(y = 0 ; y < DRAW_PIXELS_HIGH ; y++) {
66             buffer_loc = y * DRAW_PIXELS_WIDE + x;
67             byte_loc = buffer_loc / DRAW_PIXELS_PER_BYTE;
68             bit_loc = buffer_loc & (DRAW_PIXELS_PER_BYTE -1);
69
70             byte_out = byte_out >> 1;
71             if (test_bit(get_draw_buffer(byte_loc), bit_loc)) {
72                 byte_out.7  = 1;
73             } else {
74                 byte_out.7  = 0;
75             }
76             if ((y & 0b00000111) == 0b00000111) {
77                 pcd8544_send_data(byte_out);
78             }
79         }
80     }
81 }
```

Here is the call graph for this function:



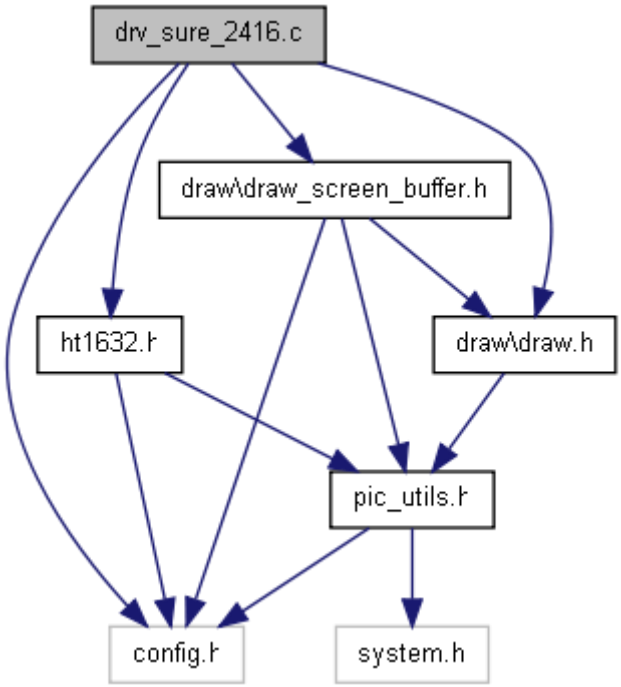
void drv_setup_io ()

```
85                      {
86     pcd8544_setup_io();
87 }
```

Here is the call graph for this function:

drv_sure_2416.c File Reference

Include dependency graph for drv_sure_2416.c:



Functions

- void drv_clear_screen ()
- uns8 drv_get_pixel (uns8 x, uns8 y)
- void drv_init ()
- void drv_paint ()
- void drv_setup_io ()

Function Documentation

void drv_clear_screen ()

```
203                          {
204     //ht1632_clear_screen();
205 }
```

uns8 drv_get_pixel (uns8 *x*, uns8 *y*)

```
199                                  {
200     //ht1632_get_pixel(x, y);
201 }
```

void drv_init ()

```
211                 {
212     // 2416 board is configured as 16 COMMONs
213     ht1632_init(HT1632_CMD_PMOS_16_COMMON);
214 }
```

Here is the call graph for this function:

void drv_paint ()

```
137                 {
138 
139 uns8 count;
140 uns8 x, y, inv_y;
141 uns16 buffer_loc;
142 uns8 byte_loc, bit_loc;
143 uns8 data;
144 
145     clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
146 
147     // send WR command
148 
149     // send 1
150     set_pin   (ht1632_data_port, ht1632_data_pin);
151     // pulse wr
152     clear_pin(ht1632_wr_port, ht1632_wr_pin);
153     set_pin   (ht1632_wr_port, ht1632_wr_pin);
154 
155     // send 0
156     clear_pin   (ht1632_data_port, ht1632_data_pin);
157     // pulse wr
158     clear_pin(ht1632_wr_port, ht1632_wr_pin);
159     set_pin   (ht1632_wr_port, ht1632_wr_pin);
160 
161     // send 1
162     set_pin   (ht1632_data_port, ht1632_data_pin);
163     // pulse wr
164     clear_pin(ht1632_wr_port, ht1632_wr_pin);
165     set_pin   (ht1632_wr_port, ht1632_wr_pin);
166 
167     // send mem address of zero
168     clear_pin(ht1632_data_port, ht1632_data_pin);
169 
170     // write mem addr, bits 6 -> 0
171     for(count = 0 ; count < 7 ; count++) {
172 
173         // pulse wr
174         clear_pin(ht1632_wr_port, ht1632_wr_pin);
175         set_pin   (ht1632_wr_port, ht1632_wr_pin);
176     }
177 
178     buffer_loc = 0;
179     for (count = 0; count < 48; count++) {
180         data = draw_buffer0[count];
181         for (bit_loc = 0; bit_loc < 8; bit_loc++) {
182             if (data.0) {
183                 set_pin(ht1632_data_port, ht1632_data_pin);
184             } else {
185                 clear_pin(ht1632_data_port, ht1632_data_pin);
186             }
187             data = data >> 1;
```

```
188             clear_pin(ht1632_wr_port, ht1632_wr_pin);
189             set_pin   (ht1632_wr_port, ht1632_wr_pin);
190         }
191     }
192
193     // reset CS
194
195     set_pin(ht1632_cs1_port, ht1632_cs1_pin);
196 }
```

void drv_setup_io ()

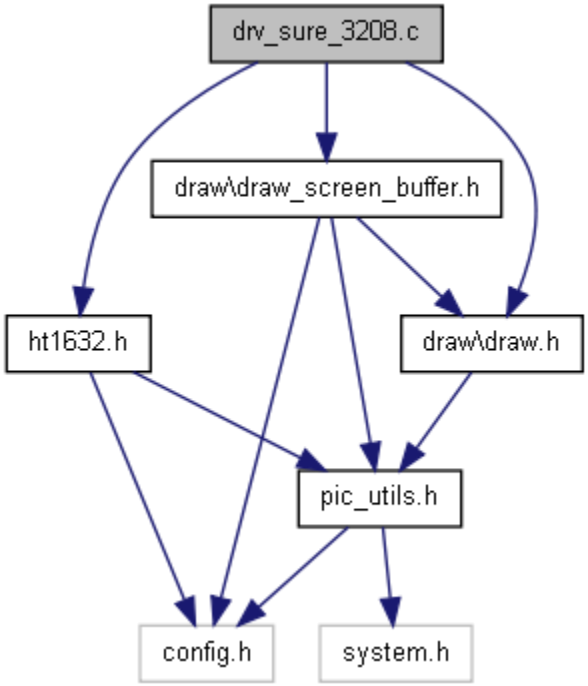
```
207                     {
208     ht1632_setup_io();
209 }
```

Here is the call graph for this function:

---

drv_sure_3208.c File Reference

Draw drivers for HT1632 based displays such as Sure 3208 and similar.
Include dependency graph for drv_sure_3208.c:



Functions

- void drv_clear_screen ()
- uns8 drv_get_pixel (uns8 x, uns8 y)
- void drv_init ()
- void drv_paint ()
- void drv_setup_io ()

Detailed Description

Function Documentation

void drv_clear_screen ()

```
161                             {
162     //ht1632_clear_screen();
163 }
```

uns8 drv_get_pixel (uns8 *x*, uns8 *y*)

```
157                                     {
158     //ht1632_get_pixel(x, y);
159 }
```

void drv_init ()

```
169                     {
170     ht1632_init();
171 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

void drv_paint ()

```
47                      {
48
49 uns8 count;
50 uns8 x, y, inv_y;
51 uns16 buffer loc;
52 uns8 byte_loc, bit_loc;
53 uns8 data;
```

```
54
55      //ht1632_send_command(HT1632_CMD_SYS_DISABLE);
56      clear pin(ht1632 cs1 port, ht1632 cs1 pin);
57
58      // send WR command
59
60      // send 1
61      set pin   (ht1632_data_port, ht1632_data_pin);
62      // pulse wr
63      clear pin(ht1632_wr_port, ht1632_wr_pin);
64      set pin   (ht1632 wr port, ht1632 wr pin);
65
66      // send 0
67      clear pin   (ht1632_data_port, ht1632_data_pin);
68      // pulse wr
69      clear pin(ht1632_wr_port, ht1632_wr_pin);
70      set pin   (ht1632_wr_port, ht1632_wr_pin);
71
72      // send 1
73      set pin   (ht1632_data_port, ht1632_data_pin);
74      // pulse wr
75      clear pin(ht1632_wr_port, ht1632_wr_pin);
76      set pin   (ht1632 wr port, ht1632 wr pin);
77
78      // send mem address of zero
79      clear pin(ht1632_data_port, ht1632_data_pin);
80
81      // write mem addr, bits 6 -> 0
82      for(count = 0 ; count < 7 ; count++) {
83
84          //change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(mem_addr, 6));
85          // pulse wr
86          clear pin(ht1632_wr_port, ht1632_wr_pin);
87          set pin   (ht1632_wr_port, ht1632_wr_pin);
88          // shift mem addr along
89
90      }
91  /*
92      for(x = 0 ; x < DRAW_PIXELS_WIDE  ; x++) {
93          for(y = 0 ; y < DRAW PIXELS HIGH ; y++) {
94              inv_y = DRAW_PIXELS_HIGH - 1 - y;
95              buffer_loc = inv_y * DRAW_PIXELS_WIDE + x;
96              byte_loc = buffer_loc / DRAW_PIXELS_PER_BYTE;
97              bit loc = buffer loc & (DRAW PIXELS PER BYTE -1);
98              if (test_bit(draw_buffer0[byte_loc], bit_loc)) {
99                  set_pin(ht1632_data_port, ht1632_data_pin);
100             } else {
101                 clear_pin(ht1632_data_port, ht1632_data_pin);
102             }
103             clear_pin(ht1632_wr_port, ht1632_wr_pin);
104             set_pin   (ht1632_wr_port, ht1632_wr_pin);
105         }
106     }
107 */
108     buffer_loc = 0;
109     for (count = 0; count < 32; count++) {
110         data = draw buffer0[count];
111         for (bit_loc = 0; bit_loc < 8; bit_loc++) {
112             if (data.0) {
113                 set pin(ht1632_data_port, ht1632_data_pin);
114             } else {
115                 clear pin(ht1632_data_port, ht1632_data_pin);
116             }
117             data = data >> 1;
118             clear pin(ht1632_wr_port, ht1632_wr_pin);
119             set pin   (ht1632_wr_port, ht1632_wr_pin);
120         }
121     }
122
123     /*uns8 xbyte = 0x2d;
124     uns8 xbit = 0b00000001;
```

```
125
126     for(x = 0 ; x < DRAW_PIXELS_WIDE  ; x++) {
127         for(y = 0 ; y < 16 ; y++) {
128         if (draw_buffer0[xbyte] & xbit) {
129                 set_pin(ht1632_data_port, ht1632_data_pin);
130             } else {
131                 clear_pin(ht1632_data_port, ht1632_data_pin);
132             }
133             clear_pin(ht1632_wr_port, ht1632_wr_pin);
134             set_pin  (ht1632_wr_port, ht1632_wr_pin);
135             if (xbyte < 3) {
136                 if (xbit == 0b10000000) {
137                     xbyte = xbyte + 0x2e;
138                     xbit = 0b00000001;
139                 } else {
140                     xbit = xbit << 1;
141                     xbyte = xbyte + 0x2d;
142                 }
143             } else {
144                 xbyte = xbyte - 3;
145             }
146         }
147     }
148 */  // reset CS
149
150     set_pin(ht1632_cs1_port, ht1632_cs1_pin);
151     //ht1632_send_command(HT1632_CMD_SYS_ENABLE);
152
153
154 }
```

void drv_setup_io ()

```
165                     {
166     ht1632_setup_io();
167 }
```

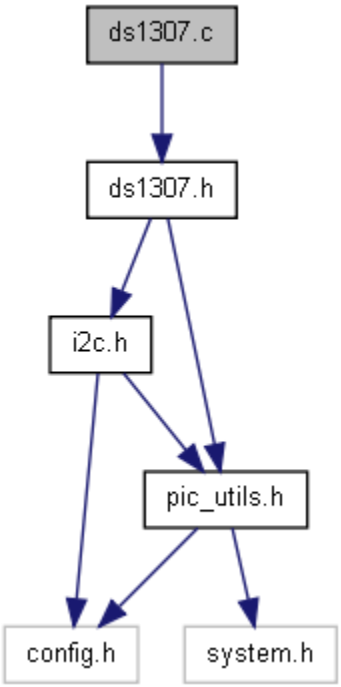
Here is the call graph for this function:

Here is the caller graph for this function:

---

ds1307.c File Reference

Include dependency graph for ds1307.c:



Functions

- uns8 bcd_to_dec (uns8 bcd)
- uns8 dec_to_bcd (uns8 dec)
- uns8 rtc_get_config ()
- *Get the config register from the ds1307.* uns8 rtc_get_date ()
- *Get the date register from the ds1307.* uns8 rtc_get_day ()
- *Get the day register from the ds1307.* uns8 rtc_get_hours ()
- *Get the decoded hours register from the ds1307.* uns8 rtc_get_minutes ()
- *Get the decoded minutes register from the ds1307.* uns8 rtc_get_month ()
- *Get the month register from the ds1307.* uns8 rtc_get_seconds ()
- *Get the decoded seconds register from the ds1307.* uns8 rtc_get_year ()
- *Get the year register from the ds1307.* uns8 rtc_set_config (uns8 config)
- *Set the config register in the ds1307.* void rtc_set_date (uns8 date)
- *Set the date register from the ds1307.* void rtc_set_day (uns8 day)
- *Set the day of the week register from the ds1307.* void rtc_set_hours (uns8 hours)
- *Set the hours register in the ds1307.* void rtc_set_minutes (uns16 minutes)
- *Set the minutes register from the ds1307.* void rtc_set_month (uns8 month)
- *Set the month register in the ds1307.* void rtc_set_seconds (uns8 seconds)
- *Set the seconds register in the ds1307.* void rtc_set_year (uns16 year)
- *Set the year register from the ds1307.* void rtc_setup_io ()
- *Setup ports and pins for use in the ds1307.* void rtc_start_clock ()
- *Starts the clock in the ds1307.* void rtc_stop_clock ()

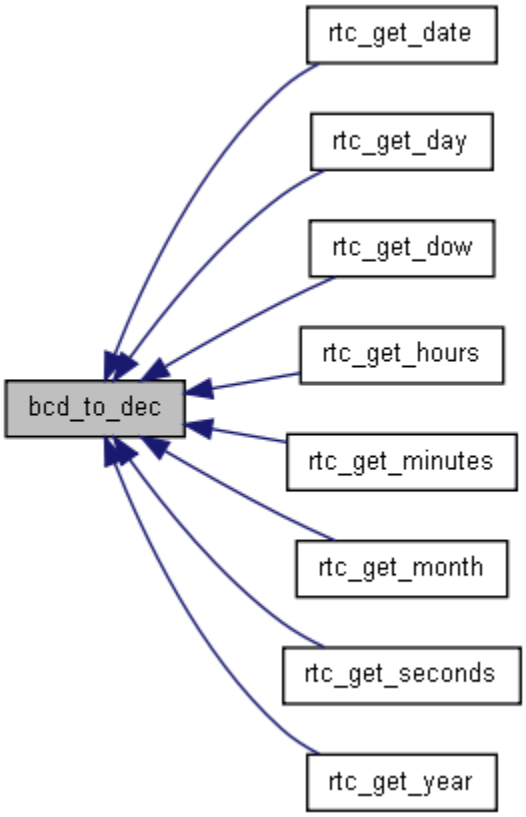
Stop the clock in the ds1307.

Function Documentation

uns8 bcd_to_dec (uns8 *bcd*)

```
39                              {
40      return (bcd & 0b00001111) + ((bcd >> 4) * 10);
41  }
```

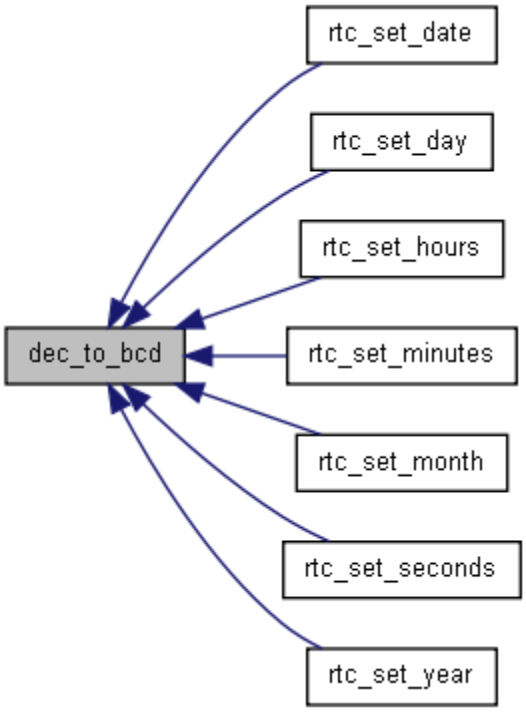
Here is the caller graph for this function:



uns8 dec_to_bcd (uns8 *dec*)

```
43                              {
44      return ((dec / 10) << 4) + (dec % 10);
45  }
```

Here is the caller graph for this function:



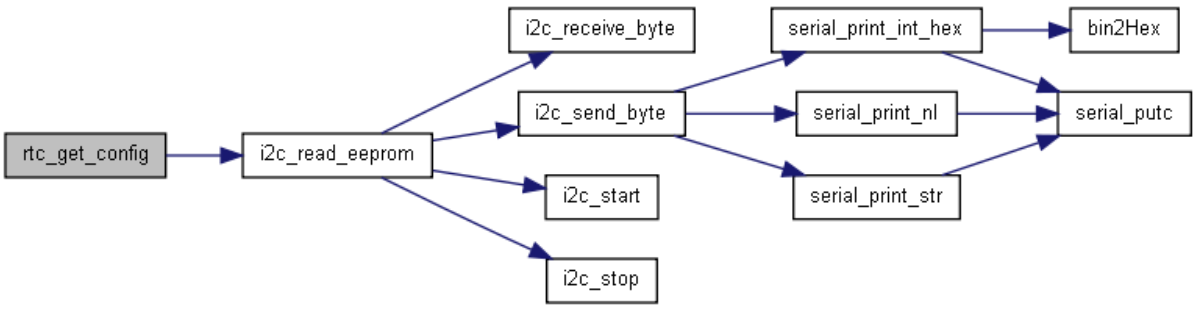
uns8 rtc_get_config ()

Returns the config register from the ds1307. Bit 7 - Out - Value on SQWE pin if not outputting square wave Bit 6 - 0 Bit 5 - 0 Bit 4 - SQWE - Enable square wave output Bit 3 - 0 Bit 2 - 0 Bit 1 - RS1 Bit 0 - RS0

RS1/0 determin the speed of the square wave output. Set to 0/0 for 1 Hz.

```
76                          {
77     return i2c_read_eeprom(ds1307_device, ds1307_control_register);
78 }
```

Here is the call graph for this function:



uns8 rtc_get_date ()

Get the date register from the m41t81s.

Returns the date in month from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 1 through 28/29/30/31 depending on month

```
65                          {
66     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_date_register));
```

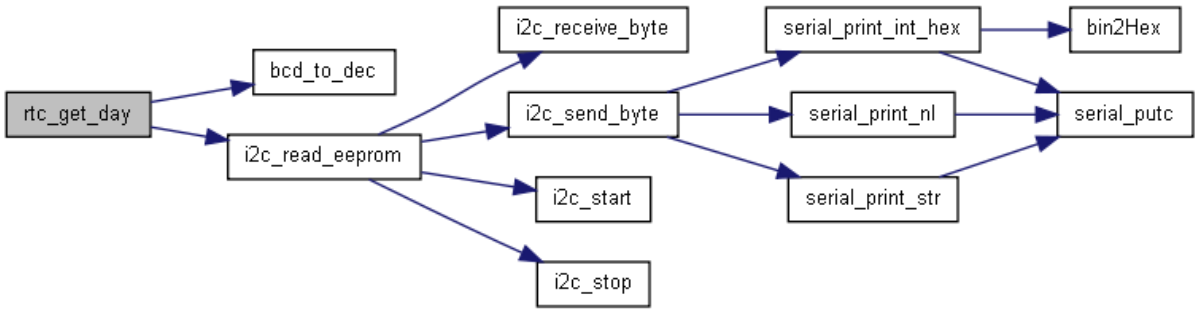
```
67 }
```

uns8 rtc_get_day ()

Returns the day of the week from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 1 through 7

```
61                      {
62     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_day_register));
63 }
```

Here is the call graph for this function:



uns8 rtc_get_hours ()

Get the decoded hours register from the m41t81s.

Returns hour from the ds1307. The result is coverted to decimal from BCD and is ready to use. These routines assume the ds1307 is running in 24 hour mode. Range - 0 through 23

```
50                      {
51
52 // Always assume it's in 24 hour mode
53
54     return bcd_to_dec(0b00111111 & i2c_read_eeprom(ds1307_device, ds1307_hours_register));
55 }
```

uns8 rtc_get_minutes ()

Get the decoded minutes register from the m41t81s.

Returns the number of minutes past the hour from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

```
47                        {
48     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_minutes_register));
49 }
```

uns8 rtc_get_month ()

Get the month register from the m41t81s.

Returns the month of the year from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 1 through 12

```
69                      {
70     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_month_register));
71 }
```

uns8 rtc_get_seconds ()

Get the decoded seconds register from the m41t81s.

Returns seconds from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

```
57                          {
58     return bcd_to_dec(0b01111111 & i2c_read_eeprom(ds1307_device, ds1307_seconds_register));
59 }
```

uns8 rtc_get_year ()

Get the year register from the m41t81s.

Returns the year from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 0 through 99

```
72                          {
73     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_year_register));
74 }
```

uns8 rtc_set_config (uns8 *config*)

Set the config register in the m41t81s.

Sets the config register in the ds1307.

Bit 7 - Out - Value on SQWE pin if not outputting square wave Bit 6 - 0 Bit 5 - 0 Bit 4 - SQWE - Enable square wave output Bit 3 - 0 Bit 2 - 0 Bit 1 - RS1 Bit 0 - RS0

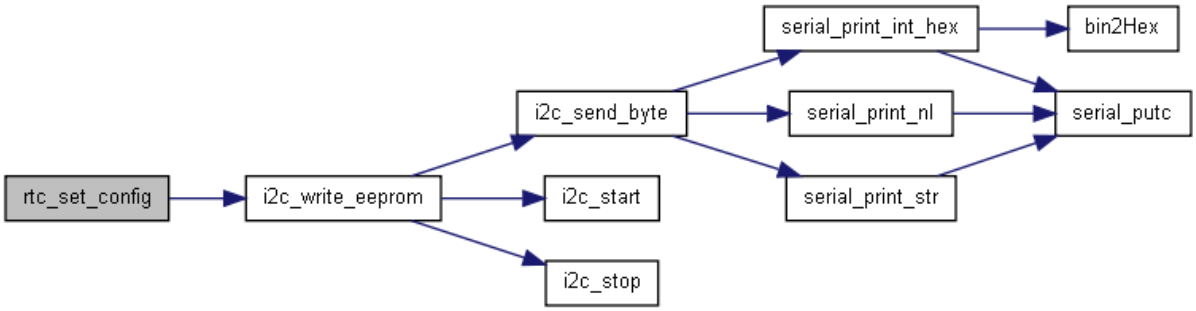
RS1/0 determin the speed of the square wave output. Set to 0/0 for 1 Hz.

Parameters:

config Value to set the config register to

```
80                                  {
81     i2c_write_eeprom(ds1307_device, ds1307_control_register, config);
82 }
```

Here is the call graph for this function:



void rtc_set_date (uns8 *date*)

Set the date register from the m41t81s.

Changes the date in the ds1307.

Parameters:

seconds Value to set date to

```
102                                 {
103     i2c_write_eeprom(ds1307_device, ds1307_date_register, dec_to_bcd(date));
104 }
```

void rtc_set_day (uns8 *day*)

Set the day of the week register from the m41t81s.

Changes the day of the week in the ds1307.

Parameters:

seconds Value to set day to

```
99                              {
100     i2c_write_eeprom(ds1307_device, ds1307_day_register, dec_to_bcd(day));
101 }
```

void rtc_set_hours (uns8 *hours*)

Set the hours register in the m41t81s.

Changes the hours in the ds1307. Forces the ds1307 into 24 hour mode.

```
110                                 {
111     // by doing this we clear the 12/24 flag, making it 24 hour mode
112         i2c_write_eeprom(ds1307_device, ds1307_hours_register, dec_to_bcd(hours));
113 }
```

void rtc_set_minutes (uns16 *minutes*)

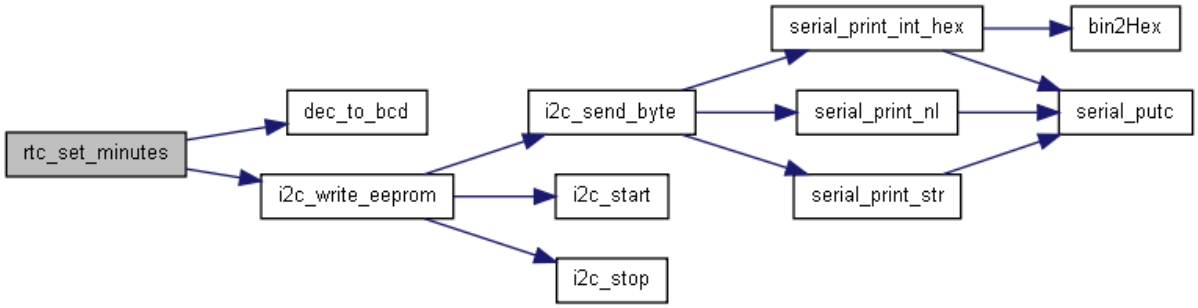
Changes the minutes in the ds1307.

Parameters:

seconds Value to set minutes to

```
96                                  {
97     i2c_write_eeprom(ds1307_device, ds1307_minutes_register, dec_to_bcd(minutes));
98 }
```

Here is the call graph for this function:



void rtc_set_month (uns8 *month*)

Set the month register in the m41t81s.

Changes the month in the ds1307.

```
115                                 {
116     i2c_write_eeprom(ds1307_device, ds1307_month_register, dec_to_bcd(month));
117 }
```

void rtc_set_seconds (uns8 *seconds*)

Set the seconds register in the m41t81s.

Changes the seconds in the ds1307.

Parameters:

seconds Value to set seconds to

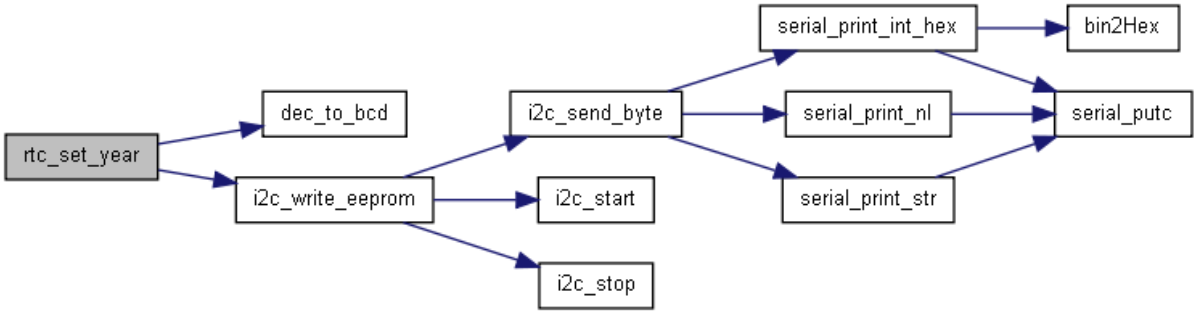
```
106                                     {
107     i2c_write_eeprom(ds1307_device, ds1307_seconds_register, (0b10000000 &
i2c_read_eeprom(ds1307_device, ds1307_seconds_register)) + dec_to_bcd(seconds));
108 }
```

void rtc_set_year (uns16 *year*)

Changes the year in the ds1307.

```
93                                  {
94     i2c_write_eeprom(ds1307_device, ds1307_year_register, dec_to_bcd(year));
95 }
```

Here is the call graph for this function:



void rtc_setup_io ()

Setup ports and pins for use in the m41t81s.

Calls i2c_setup() to configure ports and pins ready for use

```
119                         {
120     i2c_setup_io();
```

121 }

void rtc_start_clock ()

Starts the clock in the m41t81s.

Resume time in the ds1307

```
89                          {
90     i2c_write_eeprom(ds1307_device, ds1307_seconds_register, 0b01111111 &
i2c_read_eeprom(ds1307_device, ds1307_seconds_register));
91 }
```

void rtc_stop_clock ()

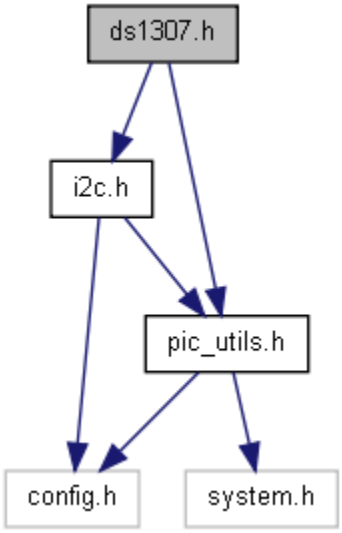
Stop the clock in the m41t81s.

Pauses time in the ds1307

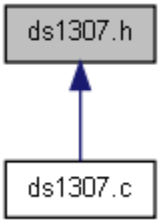
```
85                          {
86     i2c_write_eeprom(ds1307_device, ds1307_seconds_register, 0b10000000 |
i2c_read_eeprom(ds1307_device, ds1307_seconds_register));
87 }
```

ds1307.h File Reference

Routines for communicating with the ds1307 real time clock.
Include dependency graph for ds1307.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define __ds1307_h defined
- #define ds1307_control_register 0x07
- #define ds1307_date_register 0x04
- #define ds1307_day_register 0x03
- #define ds1307_device 0xD0
- #define ds1307_hours_register 0x02
- #define ds1307_minutes_register 0x01
- #define ds1307_month_register 0x05
- #define ds1307_seconds_register 0x00
- #define ds1307_year_register 0x06
- #define rtc_setup() rtc_setup_io()

Functions

- uns8 rtc_get_config ()
- *Get the config register from the ds1307.* uns8 rtc_get_date ()

- *Get the date register from the ds1307.* uns8 rtc_get_day ()
- *Get the day register from the ds1307.* uns8 rtc_get_hours ()
- *Get the decoded hours register from the ds1307.* uns8 rtc_get_minutes ()
- *Get the decoded minutes register from the ds1307.* uns8 rtc_get_month ()
- *Get the month register from the ds1307.* uns8 rtc_get_seconds ()
- *Get the decoded seconds register from the ds1307.* uns8 rtc_get_year ()
- *Get the year register from the ds1307.* uns8 rtc_set_config (uns8 config)
- *Set the config register in the ds1307.* void rtc_set_date (uns8 date)
- *Set the date register from the ds1307.* void rtc_set_day (uns8 day)
- *Set the day of the week register from the ds1307.* void rtc_set_hours (uns8 hours)
- *Set the hours register in the ds1307.* void rtc_set_minutes (uns16 minutes)
- *Set the minutes register from the ds1307.* void rtc_set_month (uns8 month)
- *Set the month register in the ds1307.* void rtc_set_seconds (uns8 seconds)
- *Set the seconds register in the ds1307.* void rtc_set_year (uns16 year)
- *Set the year register from the ds1307.* void rtc_setup_io ()
- *Setup ports and pins for use in the ds1307.* void rtc_start_clock ()
- *Starts the clock in the ds1307.* void rtc_stop_clock ()

Stop the clock in the ds1307.

Detailed Description

Define Documentation

#define __ds1307_h defined

#define ds1307_control_register 0x07

ds1307 control register

#define ds1307_date_register 0x04

ds1307 date in month register

#define ds1307_day_register 0x03

ds1307 day of week register

#define ds1307_device 0xD0

The ds1307 device address

#define ds1307_hours_register 0x02

ds1307 hours register

#define ds1307_minutes_register 0x01

ds1307 minutes register

#define ds1307_month_register 0x05

ds1307 month register

#define ds1307_seconds_register 0x00

ds1307 seconds register

#define ds1307_year_register 0x06

ds1307 year register

#define rtc_setup() rtc_setup_io()

Function Documentation

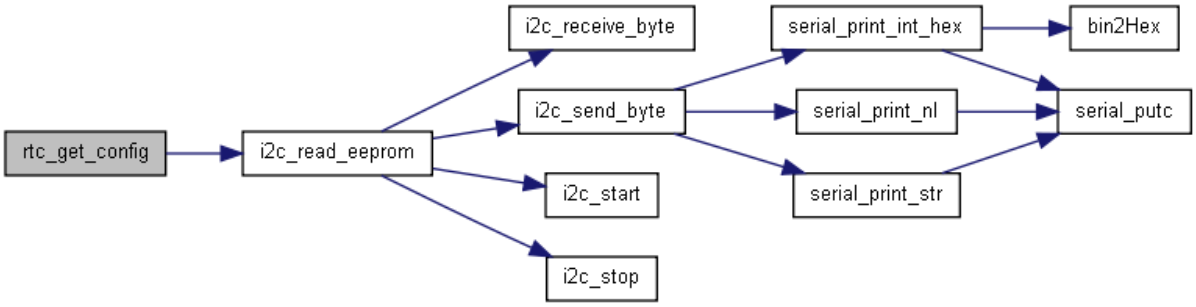
uns8 rtc_get_config ()

Returns the config register from the ds1307. Bit 7 - Out - Value on SQWE pin if not outputting square wave Bit 6 - 0 Bit 5 - 0 Bit 4 - SQWE - Enable square wave output Bit 3 - 0 Bit 2 - 0 Bit 1 - RS1 Bit 0 - RS0

RS1/0 determin the speed of the square wave output. Set to 0/0 for 1 Hz.

```
76                          {
77     return i2c_read_eeprom(ds1307_device, ds1307_control_register);
78 }
```

Here is the call graph for this function:



uns8 rtc_get_date ()

Returns the date in month from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 1 through 28/29/30/31 depending on month

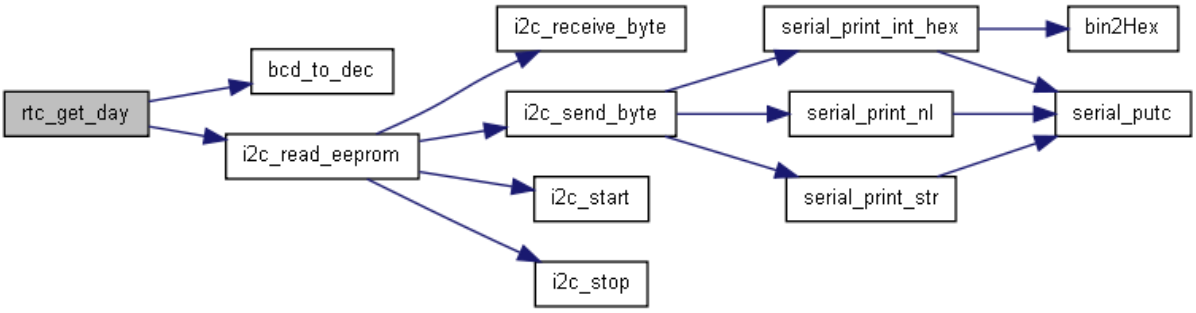
```
65                          {
66     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_date_register));
67 }
```

uns8 rtc_get_day ()

Returns the day of the week from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 1 through 7

```
61                          {
62     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_day_register));
63 }
```

Here is the call graph for this function:



uns8 rtc_get_hours ()

Returns hour from the ds1307. The result is coverted to decimal from BCD and is ready to use. These routines assume the ds1307 is running in 24 hour mode. Range - 0 through 23

```
50                      {
51
52 // Always assume it's in 24 hour mode
53
54     return bcd_to_dec(0b00111111 & i2c_read_eeprom(ds1307_device, ds1307_hours_register));
55 }
```

uns8 rtc_get_minutes ()

Returns the number of minutes past the hour from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

```
47                        {
48     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_minutes_register));
49 }
```

uns8 rtc_get_month ()

Returns the month of the year from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 1 through 12

```
69                      {
70     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_month_register));
71 }
```

uns8 rtc_get_seconds ()

Returns seconds from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

```
57                        {
58     return bcd_to_dec(0b01111111 & i2c_read_eeprom(ds1307_device, ds1307_seconds_register));
59 }
```

uns8 rtc_get_year ()

Returns the year from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 0 through 99

```
72                      {
73     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_year_register));
74 }
```

uns8 rtc_set_config (uns8 *config*)

Sets the config register in the ds1307.

Bit 7 - Out - Value on SQWE pin if not outputting square wave Bit 6 - 0 Bit 5 - 0 Bit 4 - SQWE - Enable square wave output Bit 3 - 0 Bit 2 - 0 Bit 1 - RS1 Bit 0 - RS0

RS1/0 determin the speed of the square wave output. Set to 0/0 for 1 Hz.

Parameters:

config Value to set the config register to

```
80                                      {
81     i2c_write_eeprom(ds1307_device, ds1307_control_register, config);
82 }
```

void rtc_set_date (uns8 *date*)

Changes the date in the ds1307.

Parameters:

seconds Value to set date to

```
102                                    {
103     i2c_write_eeprom(ds1307_device, ds1307_date_register, dec_to_bcd(date));
104 }
```

void rtc_set_day (uns8 *day*)

Changes the day of the week in the ds1307.

Parameters:

seconds Value to set day to

```
99                                  {
100     i2c_write_eeprom(ds1307_device, ds1307_day_register, dec_to_bcd(day));
101 }
```

void rtc_set_hours (uns8 *hours*)

Changes the hours in the ds1307. Forces the ds1307 into 24 hour mode.

```
110                                      {
111     // by doing this we clear the 12/24 flag, making it 24 hour mode
112         i2c_write_eeprom(ds1307_device, ds1307_hours_register, dec_to_bcd(hours));
113 }
```

void rtc_set_minutes (uns16 *minutes*)

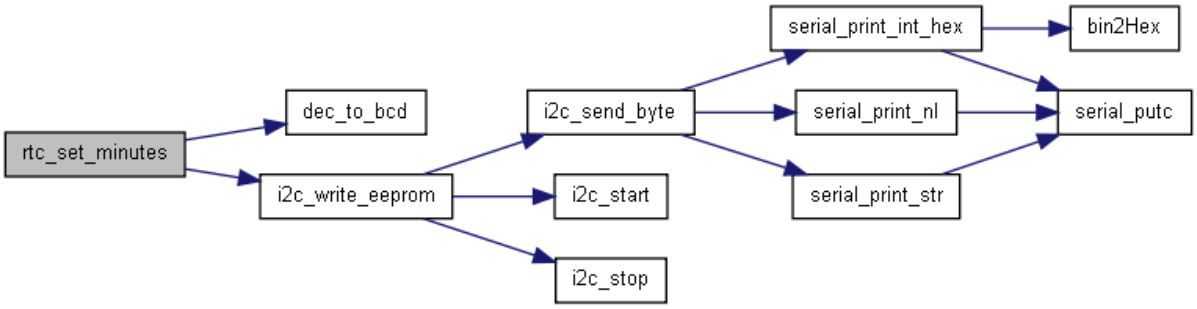
Changes the minutes in the ds1307.

Parameters:

seconds Value to set minutes to

```
96                                      {
97     i2c_write_eeprom(ds1307_device, ds1307_minutes_register, dec_to_bcd(minutes));
98 }
```

Here is the call graph for this function:



void rtc_set_month (uns8 *month*)

Changes the month in the ds1307.

```
115                                 {
116     i2c_write_eeprom(ds1307_device, ds1307_month_register, dec_to_bcd(month));
117 }
```

void rtc_set_seconds (uns8 *seconds*)

Changes the seconds in the ds1307.

Parameters:

seconds Value to set seconds to

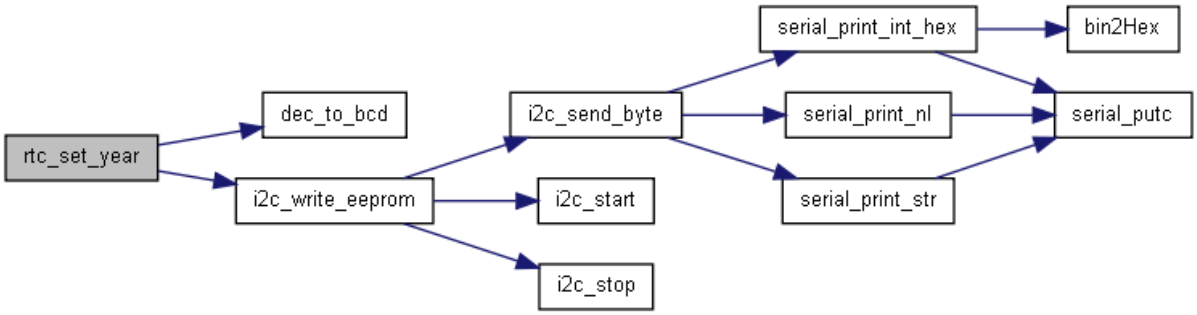
```
106                                 {
107     i2c_write_eeprom(ds1307_device, ds1307_seconds_register, (0b10000000 &
i2c_read_eeprom(ds1307_device, ds1307_seconds_register)) + dec_to_bcd(seconds));
108 }
```

void rtc_set_year (uns16 *year*)

Changes the year in the ds1307.

```
93                                  {
94     i2c_write_eeprom(ds1307_device, ds1307_year_register, dec_to_bcd(year));
95 }
```

Here is the call graph for this function:



void rtc_setup_io ()

Calls i2c_setup() to configure ports and pins ready for use

```
119                     {
120     i2c_setup_io();
121 }
```

void rtc_start_clock ()

Resume time in the ds1307

```
89                          {
90     i2c_write_eeprom(ds1307_device, ds1307_seconds_register, 0b01111111 &
i2c_read_eeprom(ds1307_device, ds1307_seconds_register));
91 }
```

void rtc_stop_clock ()

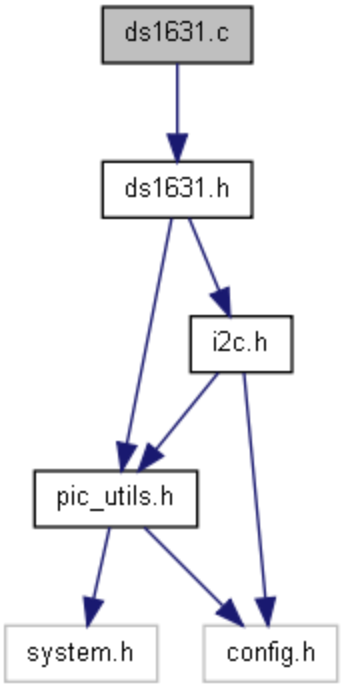
Pauses time in the ds1307

```
85                          {
86     i2c_write_eeprom(ds1307_device, ds1307_seconds_register, 0b10000000 |
i2c_read_eeprom(ds1307_device, ds1307_seconds_register));
87 }
```

---

ds1631.c File Reference

Include dependency graph for ds1631.c:



Functions

- void ds1631_convert_temp (uns8 addr)
- *Start temperature conversion on ds1631.* uns8 ds1631_get_config (uns8 addr)
- *Get ds1631 config register.* uns16 ds1631_get_temp (uns8 addr)
- *Read temperature from ds1631.* void ds1631_set_config (uns8 addr, uns8 config)
- *Set ds1631 config register.* void ds1631_setup ()

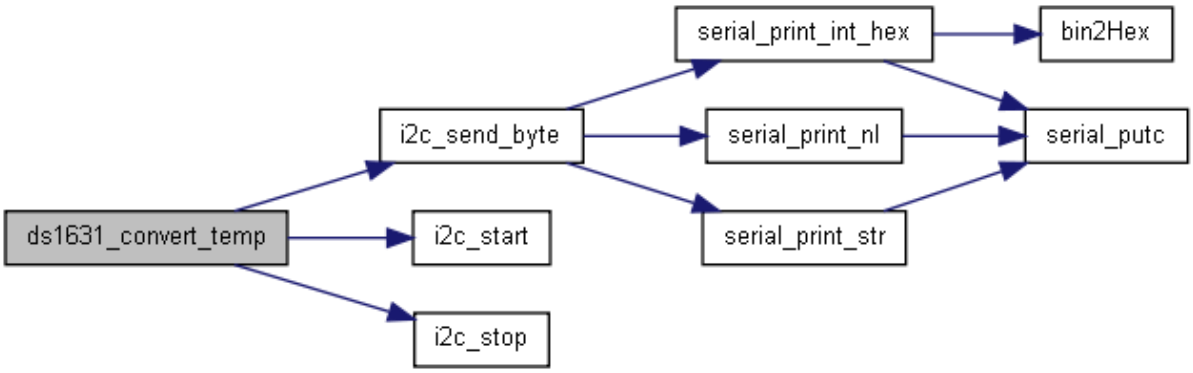
Function Documentation

void ds1631_convert_temp (uns8 *addr*)

This routine starts the temperature conversion in the ds1631. Issue this command before actually reading the temperature.

```
52                                      {
53     i2c_start();
54     i2c_send_byte(0x90 + addr);
55     i2c_send_byte(ds1631_start_convert);
56     i2c_stop();
57 }
```

Here is the call graph for this function:

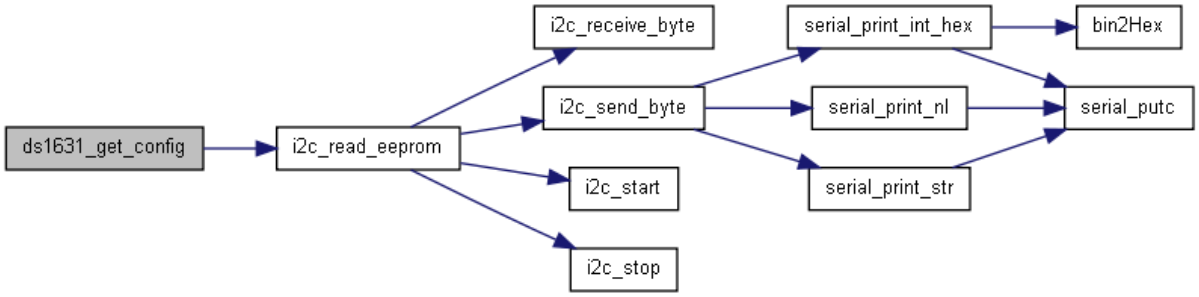


uns8 ds1631_get_config (uns8 *addr*)

Gets the ds1631 config register (memory location 0x01)

```
48 {
49     return i2c_read_eeprom(0x90 + addr, ds1631_access_config);
50 }
```

Here is the call graph for this function:

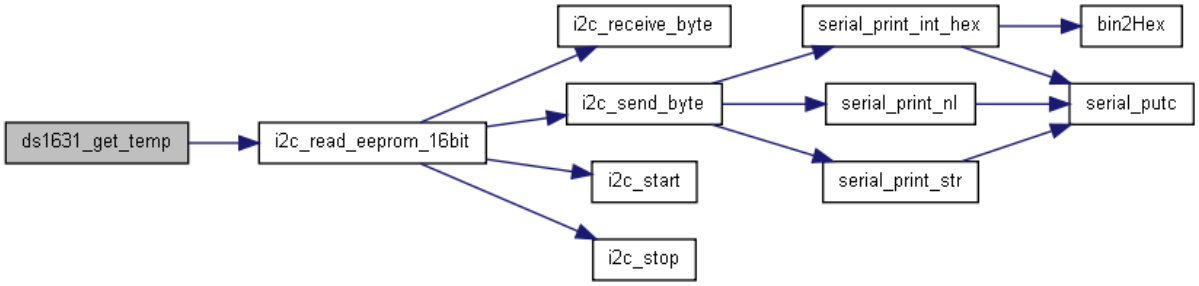


uns16 ds1631_get_temp (uns8 *addr*)

Returns 16bit raw temperature register from ds1631. Note that if you are in one-shot mode (the default) you must have already issued a start_convert and waited until it is complete (to check for completion you can either wait long enough, or query the config register to check).

```
60 {
61
62     return i2c_read_eeprom_16bit(0x90 + addr, ds1631_read_temp);
63
64 }
```

Here is the call graph for this function:

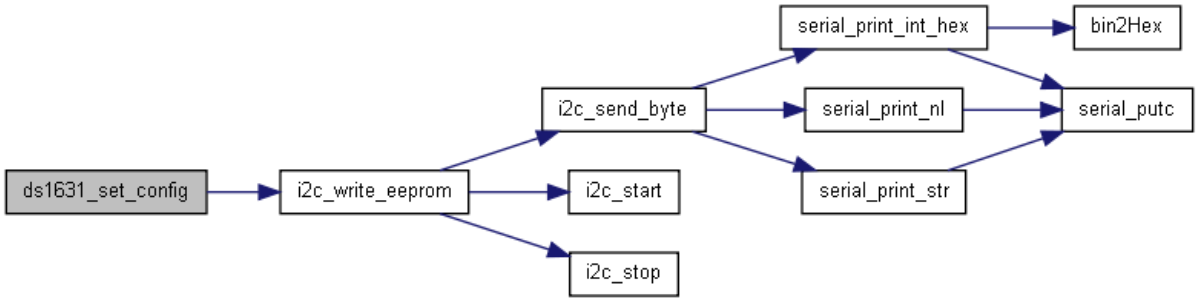


void ds1631_set_config (uns8 *addr*, uns8 *config*)

Sets the ds1631 config register

```
43 {
44     i2c_write_eeprom(0x90 + addr, ds1631_access_config, config);
45 }
```

Here is the call graph for this function:



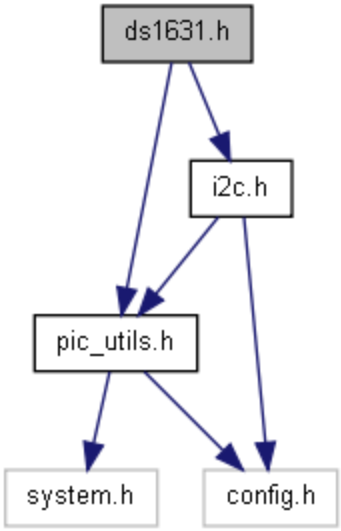
void ds1631_setup ()

```
38                      {
39     i2c_setup();
40 }
```

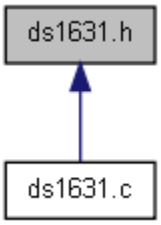
---

ds1631.h File Reference

DS1631 temperature sensor routines.
Include dependency graph for ds1631.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define ds1631_access_config 0xAC
- #define ds1631_access_th 0xA1
- #define ds1631_access_tl 0xA2
- #define ds1631_read_temp 0xAA
- #define ds1631_setup() ds1631_setup_io()
- *Setup ds1631 ports and pins.* #define ds1631_software_por 0x54
- #define ds1631_start_convert 0x51
- #define ds1631_stop_convert 0x22

Functions

- void ds1631_convert_temp (uns8 addr)
- *Start temperature conversion on ds1631.* uns8 ds1631_get_config (uns8 addr)
- *Get ds1631 config register.* uns16 ds1631_get_temp (uns8 addr)
- *Read temperature from ds1631.* void ds1631_set_config (uns8 addr, uns8 config)
- *Set ds1631 config register.* void ds1631_setup_io (void)

Detailed Description

Define Documentation

#define ds1631_access_config 0xAC

#define ds1631_access_th 0xA1

#define ds1631_access_tl 0xA2

#define ds1631_read_temp 0xAA

#define ds1631_setup() ds1631_setup_io()

#define ds1631_software_por 0x54

#define ds1631_start_convert 0x51

#define ds1631_stop_convert 0x22

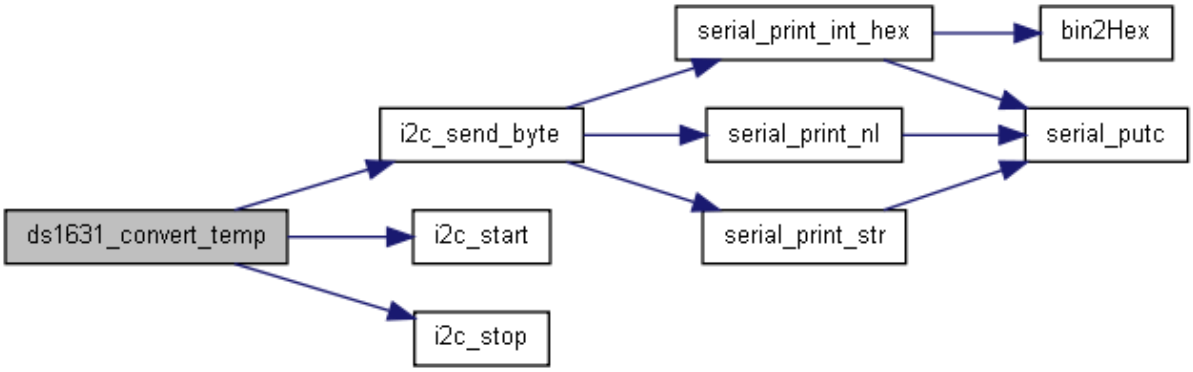
Function Documentation

void ds1631_convert_temp (uns8 *addr*)

This routine starts the temperature conversion in the ds1631. Issue this command before actually reading the temperature.

```
52                                      {
53     i2c_start();
54     i2c_send_byte(0x90 + addr);
55     i2c_send_byte(ds1631_start_convert);
56     i2c_stop();
57 }
```

Here is the call graph for this function:

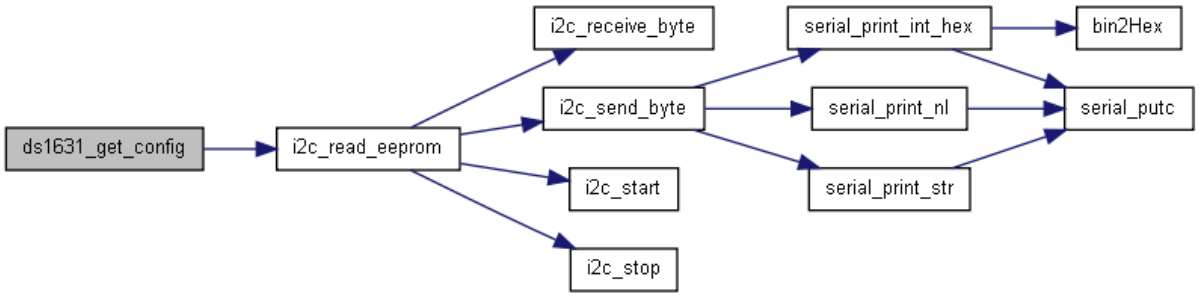


uns8 ds1631_get_config (uns8 *addr*)

Gets the ds1631 config register (memory location 0x01)

```
48 {
49     return i2c_read_eeprom(0x90 + addr, ds1631_access_config);
50 }
```

Here is the call graph for this function:

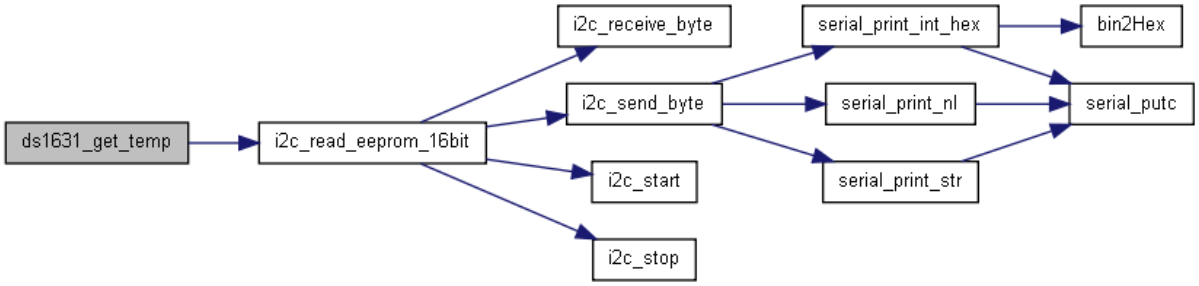


uns16 ds1631_get_temp (uns8 *addr*)

Returns 16bit raw temperature register from ds1631. Note that if you are in one-shot mode (the default) you must have already issued a start_convert and waited until it is complete (to check for completion you can either wait long enough, or query the config register to check).

```
60 {
61
62     return i2c_read_eeprom_16bit(0x90 + addr, ds1631_read_temp);
63
64 }
```

Here is the call graph for this function:

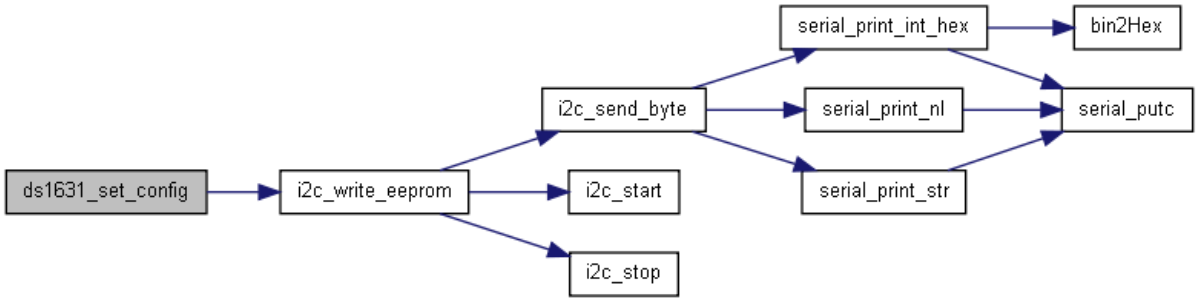


void ds1631_set_config (uns8 *addr*, uns8 *config*)

Sets the ds1631 config register

```
43 {
44     i2c_write_eeprom(0x90 + addr, ds1631_access_config, config);
45 }
```

Here is the call graph for this function:



void ds1631_setup_io (void)

---

ea_bitmaps.c File Reference

Bitmaps for draw library testing.

Variables

- char a_big_bitmap []
- char a_bitmap []
- char adventures_bitmap []
- char e_big_bitmap []
- char e_bitmap []
- char embedded_bitmap []

Detailed Description

Variable Documentation

char a_big_bitmap[]

char a_bitmap[]

```
Initial value: {
0x12,
0x10,
0x01,
0x24,
0x80,0x3F,0xC0,0x7F,0xC0,0x7F,0xE0,0xFF,0xE0,0xF1,0xE0,0xF1,0xE0,0xF1,0xE0,0xF1,
0xE6,0xF1,0xEF,0xF1,0xEF,0xF1,0xEF,0xF1,0xEF,0xF1,0xEF,0xF1,0xFF,0xFF,0xFE,0x7F,
0xFE,0x7F,0xF8,0x3F,
}
```

char adventures_bitmap[]

```
Initial value: {
0x40,
0x08,
0x01,
0x40,
0x70,0x90,0x90,0x90,0x94,0x94,0x7C,0x00,0x78,0x84,0x84,0x84,0x84,0x84,0x7F,0x00,
0x0C,0x18,0x60,0xC0,0x70,0x18,0x0C,0x00,0x78,0xA4,0xA4,0x24,0x24,0x38,0x00,0xF8,
0x04,0x04,0x04,0x04,0x04,0xF8,0x04,0xFF,0x04,0x04,0x7C,0x80,0x80,0x80,0x80,0xFC,
0x00,0xF8,0x04,0x04,0x00,0x78,0xA4,0xA4,0x24,0x24,0x38,0x00,0x98,0xA4,0xA4,0xC4,
}
```

char e_big_bitmap[]

char e_bitmap[]

```
Initial value: {
0x11,
0x10,
```

```
0x01,
0x22,

0xF8,0x1F,0xFE,0x7F,0xFE,0x7F,0xFF,0xFF,0xCF,0xF3,0xCF,0xF3,0xCF,0xF3,0xCF,0xF3,
0xCF,0xF3,0xCF,0xF3,0xCF,0x03,0xCF,0x03,0xCF,0x03,0xFF,0x03,0xFE,0x03,0xFE,0x03,
0xFC,0x03,
}
```

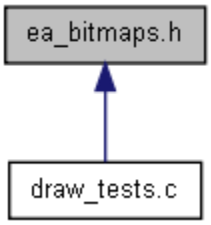
char embedded_bitmap[]

```
Initial value: {
0x40,
0x08,
0x01,
0x40,
0x78,0xA4,0xA4,0x24,0x24,0x38,0x00,0x00,0xF8,0x04,0x04,0xFC,0x04,0x04,0x04,0xF8,
0x00,0x7F,0x84,0x84,0x84,0x84,0x84,0x78,0x00,0x00,0x78,0xA4,0xA4,0x24,0x24,0x38,
0x00,0x78,0x84,0x84,0x84,0x84,0x84,0x7F,0x00,0x78,0x84,0x84,0x84,0x84,0x84,0x7F,
0x00,0x00,0x78,0xA4,0xA4,0x24,0x24,0x38,0x00,0x78,0x84,0x84,0x84,0x84,0x84,0x7F,
}
```

---

ea_bitmaps.h File Reference

Bitmaps for draw library testing.
This graph shows which files directly or indirectly include this file:



Variables

- char a_big_bitmap [1]
- char a_bitmap [1]
- char adventures_bitmap [1]
- char e_big_bitmap [1]
- char e_bitmap [1]
- char embedded_bitmap [1]

Detailed Description

Variable Documentation

char a_big_bitmap[1]

char a_bitmap[1]

char adventures_bitmap[1]

char e_big_bitmap[1]

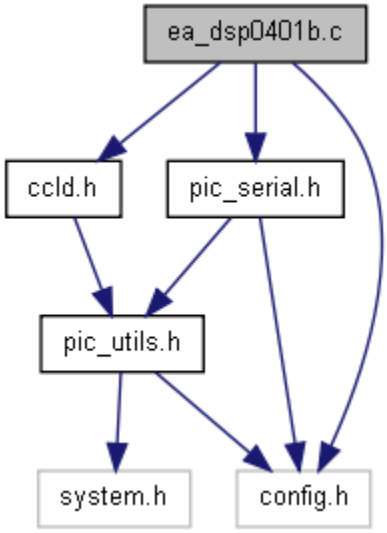
char e_bitmap[1]

char embedded_bitmap[1]

---

ea_dsp0401b.c File Reference

Include dependency graph for ea_dsp0401b.c:



Defines

- #define MAX_CURSOR DSP_0401B_TOTAL_CHARS

Functions

- void ea_dsp0401b_display (void)
- void ea_dsp0401b_print_str (char *str)
- void ea_dsp0401b_set_cursor (uns8 x)
- void ea_dsp0401b_set_raw (uns8 pos, uns16 raw)
- void ea_dsp0401b_setup_io (void)
- uns16 ea_dsp0401b_translate (uns8 my_char)

Variables

- uns8 cursor = 0

- uns16 data_array [MAX_CURSOR]
- uns8 max_cursor = MAX_CURSOR

Define Documentation

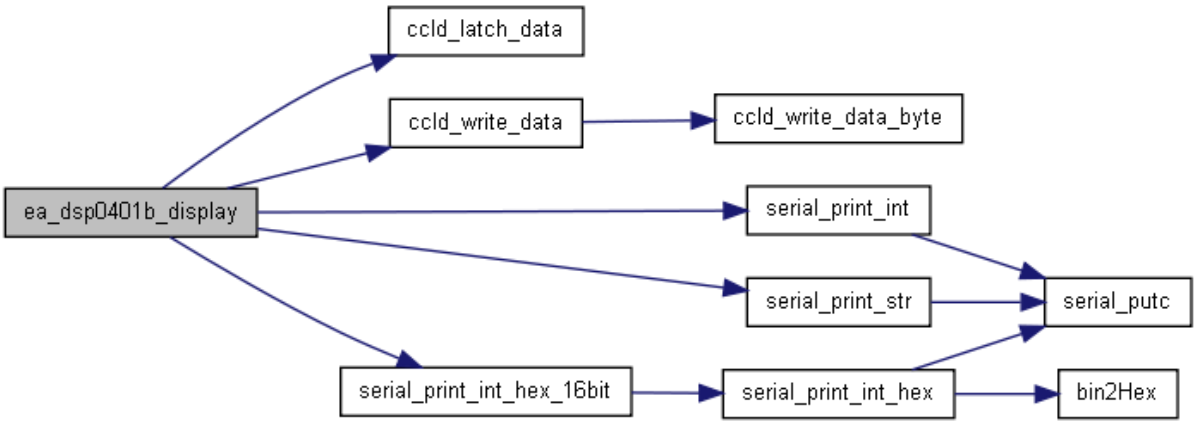
#define MAX_CURSOR DSP_0401B_TOTAL_CHARS

Function Documentation

void ea_dsp0401b_display (void)

```
100                                 {
101
102 uns8 count;
103
104     for (count = DSP_0401B_TOTAL_CHARS; count != 0xff ; count --) {
105         serial_print_str(" count ");
106         serial_print_int(count);
107         serial_print_str(" = 0x");
108         serial_print_int_hex_16bit(data_array[count]);
109         ccld_write_data(data_array[count]);
110     }
111     ccld_latch_data();
112 }
```

Here is the call graph for this function:



void ea_dsp0401b_print_str (char * *str*)

```
85                                      {
86     while ((*str != 0) && (cursor < max_cursor)) {
87         data_array[cursor++] = ea_dsp0401b_translate(*str);
88         str++;
89     }
90 }
```

Here is the call graph for this function:

void ea_dsp0401b_set_cursor (uns8 *x*)

```
15                                      {
16      cursor = x;
17  }
```

void ea_dsp0401b_set_raw (uns8 *pos*, uns16 *raw*)

```
92                                                  {
93      data_array[pos] = raw;
94  }
```

void ea_dsp0401b_setup_io (void)

```
96                                      {
97      ccld_setup_io();
98  }
```

uns16 ea_dsp0401b_translate (uns8 *my_char*)

```
19                                              {
20      uns16 result = 0;
21
22
23      switch (my_char) {      //fedcba9876543210
24          case ' ': result = 0b0000000000000000; break;
25          case '0': result = 0b1010111010101110; break;
26          case '1': result = 0b0010101000000000; break;
27          case '2': result = 0b1001110010000110; break;
28          case '3': result = 0b1011110010000100; break;
29          case '4': result = 0b0001000100110001; break;
30          case '5': result = 0b1011010010110100; break;
31          case '6': result = 0b1011000010111100; break;
32          case '7': result = 0b0000011010000010; break;
33          case '8': result = 0b1011110010111100; break;
34          case '9': result = 0b1011110010110000; break;
35          case 'A': result = 0b0011110010111000; break;
36          case 'B': result = 0b1011110110000101; break;
37          case 'C': result = 0b1000010010101100; break;
38          case 'D': result = 0b1010110110000101; break;
39          case 'E': result = 0b1000010010111100; break;
40          case 'F': result = 0b0000010010111000; break;
41          case 'G': result = 0b1011010010101100; break;
42
43                            //fedcba9876543210
44          case 'H': result = 0b0011100000111000; break;
45          case 'I': result = 0b1000010110000101; break;
46          case 'J': result = 0b1010100000001100; break;
47          case 'K': result = 0b0100001000111000; break;
48          case 'L': result = 0b1000000000101100; break;
49          case 'M': result = 0b0010101001101000; break;
50          case 'N': result = 0b0110100001101000; break;
51          case 'O': result = 0b1010110010101100; break;
52          case 'P': result = 0b0001110010111000; break;
53          case 'Q': result = 0b1110110010101100; break;
54          case 'R': result = 0b0101110010111000; break;
55
56                            //fedcba9876543210
```

```
57          case 'S': result = 0b1011010010110100; break;
58          case 'T': result = 0b0000010110000001; break;
59          case 'U': result = 0b1010100000101100; break;
60          case 'V': result = 0b0000001000101010; break;
61          case 'W': result = 0b0110100000101010; break;
62          case 'X': result = 0b0100001001000010; break;
63          case 'Y': result = 0b0000001001000001; break;
64          case 'Z': result = 0b1000011010000110; break;
65          case '$': result = 0b1011010110110101; break;
66          case '[': result = 0b1000010100000001; break;
67          case ']': result = 0b0000000110000101; break;
68          case '<': result = 0b0100001000000000; break;
69      case '>': result = 0b0000000001000010; break;
70          case '\\':result = 0b0100000001000000; break;
71          case '/': result = 0b0000001000000010; break;
72          case '_': result = 0b1000000000000100; break;
73          case '-': result = 0b0001000000010000; break;
74          case '+': result = 0b0001000100010001; break;
75          case '=': result = 0b1001000000010100; break;
76
77
78
79      }
80
81      return result;
82  }
```

Here is the caller graph for this function:

Variable Documentation

uns8 cursor = 0

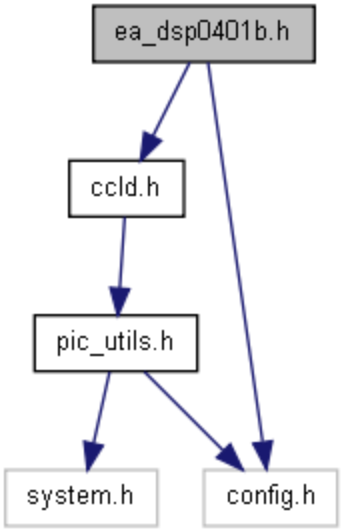
uns16 data_array[MAX_CURSOR]

uns8 max_cursor = MAX_CURSOR

---

ea_dsp0401b.h File Reference

Include dependency graph for ea_dsp0401b.h:



Functions

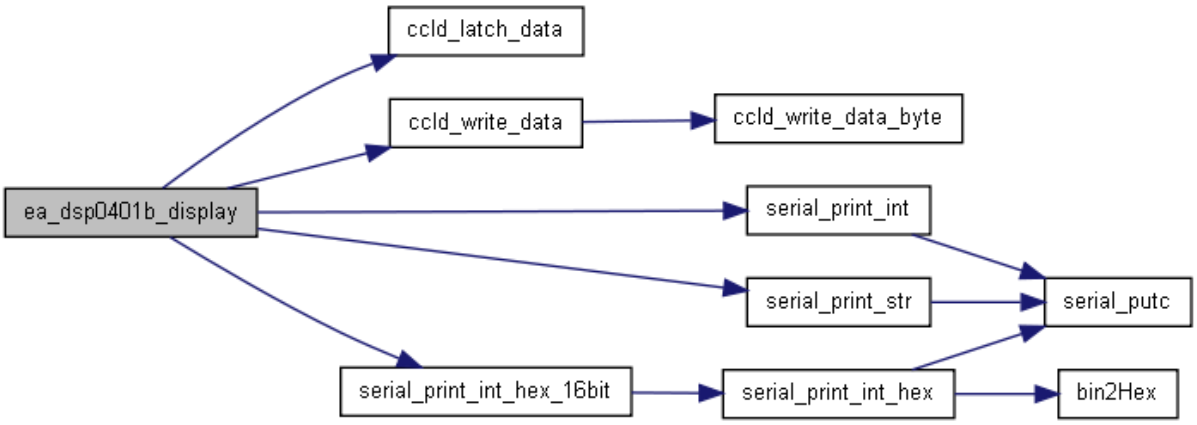
- void ea_dsp0401b_display (void)
- void ea_dsp0401b_print_str (char *str)
- void ea_dsp0401b_set_cursor (uns8 x)
- void ea_dsp0401b_set_raw (uns8 pos, uns16 raw)
- void ea_dsp0401b_setup_io (void)

Function Documentation

void ea_dsp0401b_display (void)

```
100                                 {
101
102 uns8 count;
103
104     for (count = DSP 0401B TOTAL CHARS; count != 0xff ; count --) {
105         serial print str(" count ");
106         serial print int(count);
107         serial print str(" = 0x");
108         serial print int hex 16bit(data array[count]);
109         ccld write data(data array[count]);
110     }
111     ccld latch data();
112 }
```

Here is the call graph for this function:



void ea_dsp0401b_print_str (char * *str*)

```
85                                          {
86      while ((*str != 0) && (cursor < max_cursor)) {
87          data_array[cursor++] = ea_dsp0401b_translate(*str);
88          str++;
89      }
90  }
```

Here is the call graph for this function:

void ea_dsp0401b_set_cursor (uns8 *x*)

```
15                                      {
16      cursor = x;
17  }
```

void ea_dsp0401b_set_raw (uns8 *pos*, uns16 *raw*)

```
92                                                  {
93      data_array[pos] = raw;
94  }
```

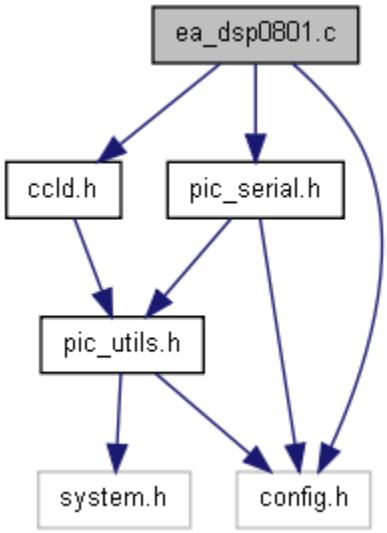
void ea_dsp0401b_setup_io (void)

```
96                                      {
97      ccld_setup_io();
98  }
```

Here is the call graph for this function:

ea_dsp0801.c File Reference

Include dependency graph for ea_dsp0801.c:



Defines

- #define MAX_CURSOR DSP_0801_TOTAL_CHARS

Functions

- void ea_dsp0401b_setup_io (void)
- void ea_dsp0801_clear_dot (uns8 pos)
- void ea_dsp0801_display (void)
- void ea_dsp0801_fill ()
- void ea_dsp0801_print_str (char *str)
- void ea_dsp0801_set_cursor (uns8 x)
- void ea_dsp0801_set_dot (uns8 pos)
- void ea_dsp0801_set_raw (uns8 pos, uns16 raw)
- uns16 ea_dsp0801_translate (uns8 my_char)

Variables

- uns8 cursor = 0
- uns16 data_array [MAX_CURSOR]
- uns8 max_cursor = MAX_CURSOR

Define Documentation

#define MAX_CURSOR DSP_0801_TOTAL_CHARS

Function Documentation

void ea_dsp0401b_setup_io (void)

```
112                                 {
113     ccld_setup_io();
114 }
```

Here is the call graph for this function:

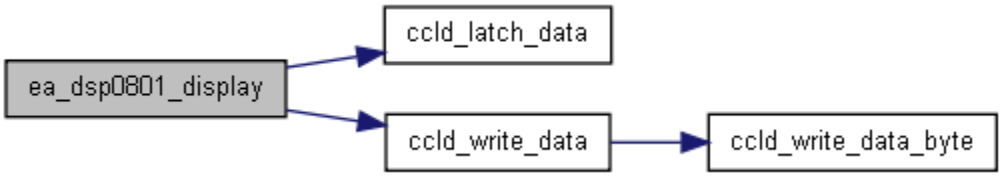
void ea_dsp0801_clear_dot (uns8 *pos*)

```
120                                         {
121     clear bit(data array[pos], 14);
122 }
```

void ea_dsp0801_display (void)

```
93                                  {
94
95 uns8 count;
96
97     for (count = 0; count < DSP_0801_TOTAL_CHARS ; count ++) {
98         ccld_write_data(data_array[count]);
99     }
100     ccld_latch_data();
101 }
```

Here is the call graph for this function:



void ea_dsp0801_fill (void)

```
13                          {
14     data_array[0] = 0x0102;
15     data_array[1] = 0x0304;
16     data_array[2] = 0x0506;
17     data_array[3] = 0x0708;
18     data_array[4] = 0x0910;
19     data_array[5] = 0x1112;
20     data_array[6] = 0x1314;
21     data_array[7] = 0x1516;
22 }
```

void ea_dsp0801_print_str (char * *str*)

```
79                                          {
80 uns8 temp_bit;
81     while ((*str != 0) && (cursor < max_cursor)) {
82         temp_bit = test_bit(data_array[cursor], 14);
83 
84         data_array[cursor] = ea_dsp0801_translate(*str);
85         if (temp_bit) { // preserve dot LED
86             set_bit(data_array[cursor], 14);
87         }
88         cursor++;
89         str++;
90     }
91 }
```

Here is the call graph for this function:

void ea_dsp0801_set_cursor (uns8 *x*)

```
25                                          {
26     cursor = x;
27 }
```

void ea_dsp0801_set_dot (uns8 *pos*)

```
116                                         {
117     set_bit(data_array[pos] ,14);
118 }
```

void ea_dsp0801_set_raw (uns8 *pos*, uns16 *raw*)

```
105                                                 {
106     data_array[pos] = raw;
107 }
```

uns16 ea_dsp0801_translate (uns8 *my_char*)

```
29                                          {
30     uns16 result = 0;
31 
32 
33     switch (my_char) {
34 
35         case ' ': result = 0b0000000000000000; break;
36         case '0': result = 0b0010010000111111; break;
37         case '1': result = 0b0000000000000110; break;
38         case '2': result = 0b0000000011011011; break;
39         case '3': result = 0b0000000010001111; break;
40         case '4': result = 0b0001001011100000; break;
41         case '5': result = 0b0000000011101101; break;
42         case '6': result = 0b0000000011111101; break;
43         case '7': result = 0b0000000000000111; break;
44         case '8': result = 0b0000000011111111; break;
```

```
45          case '9': result = 0b0000000011101111; break;
46          case 'A': result = 0b0000000011110111; break;
47          case 'B': result = 0b0001001010001111; break;
48          case 'C': result = 0b0000000000111001; break;
49          case 'D': result = 0b0001001000001111; break;
50          case 'E': result = 0b0000000001111001; break;
51          case 'F': result = 0b0000000001110001; break;
52          case 'G': result = 0b0000000010111101; break;
53          case 'H': result = 0b0000000011110110; break;
54          case 'I': result = 0b0001001000001001; break;
55          case 'J': result = 0b0000000000011110; break;
56          case 'K': result = 0b0000110001110000; break;
57          case 'L': result = 0b0000000000111000; break;
58          case 'M': result = 0b0000010100110110; break;
59          case 'N': result = 0b0000100100110110; break;
60          case 'O': result = 0b0000000000111111; break;
61          case 'P': result = 0b0000000011110011; break;
62          case 'Q': result = 0b0000100000111111; break;
63          case 'R': result = 0b0000100011110011; break;
64          case 'S': result = 0b0000000011101101; break;
65          case 'T': result = 0b0001001000000001; break;
66          case 'U': result = 0b0000000000111110; break;
67          case 'V': result = 0b0010010000110000; break;
68          case 'W': result = 0b0010100000110110; break;
69          case 'X': result = 0b0010110100000000; break;
70          case 'Y': result = 0b0001010100000000; break;
71          case 'Z': result = 0b0010010000001001; break;
72          case '*': result = 0b0011111111000000; break;
73
74      }
75      return result;
76  }
```

Here is the caller graph for this function:

Variable Documentation

uns8 cursor = 0

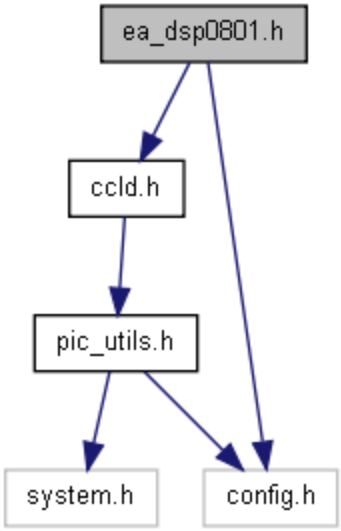
uns16 data_array[MAX_CURSOR]

uns8 max_cursor = MAX_CURSOR

---

ea_dsp0801.h File Reference

Include dependency graph for ea_dsp0801.h:



Functions

- void ea_dsp0801_clear_dot (uns8 pos)
- void ea_dsp0801_display (void)
- void ea_dsp0801_fill (void)
- void ea_dsp0801_print_str (char *str)
- void ea_dsp0801_set_cursor (uns8 x)
- void ea_dsp0801_set_dot (uns8 pos)
- void ea_dsp0801_set_raw (uns8 pos, uns16 raw)
- void ea_dsp0801_setup_io (void)
- uns16 ea_dsp0801_translate (uns8)

Function Documentation

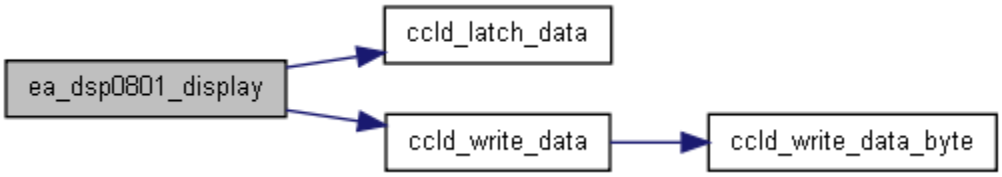
void ea_dsp0801_clear_dot (uns8 *pos*)

```
120                                     {
121     clear_bit(data_array[pos], 14);
122 }
```

void ea_dsp0801_display (void)

```
93                                  {
94
95 uns8 count;
96
97     for (count = 0; count < DSP_0801_TOTAL_CHARS ; count ++) {
98         ccld_write_data(data_array[count]);
99     }
100     ccld_latch_data();
101 }
```

Here is the call graph for this function:



void ea_dsp0801_fill (void)

```
13                          {
14      data_array[0] = 0x0102;
15      data_array[1] = 0x0304;
16      data_array[2] = 0x0506;
17      data_array[3] = 0x0708;
18      data_array[4] = 0x0910;
19      data_array[5] = 0x1112;
20      data_array[6] = 0x1314;
21      data_array[7] = 0x1516;
22  }
```

void ea_dsp0801_print_str (char * *str*)

```
79                                      {
80  uns8 temp_bit;
81      while ((*str != 0) && (cursor < max_cursor)) {
82          temp_bit = test_bit(data_array[cursor], 14);
83
84          data_array[cursor] = ea_dsp0801_translate(*str);
85          if (temp_bit) { // preserve dot LED
86              set_bit(data_array[cursor], 14);
87          }
88          cursor++;
89          str++;
90      }
91  }
```

Here is the call graph for this function:

void ea_dsp0801_set_cursor (uns8 *x*)

```
25                                      {
26      cursor = x;
27  }
```

void ea_dsp0801_set_dot (uns8 *pos*)

```
116                                     {
117      set_bit(data_array[pos] ,14);
118  }
```

void ea_dsp0801_set_raw (uns8 *pos*, uns16 *raw*)

```
105                                                 {
106     data_array[pos] = raw;
107 }
```

void ea_dsp0801_setup_io (void)

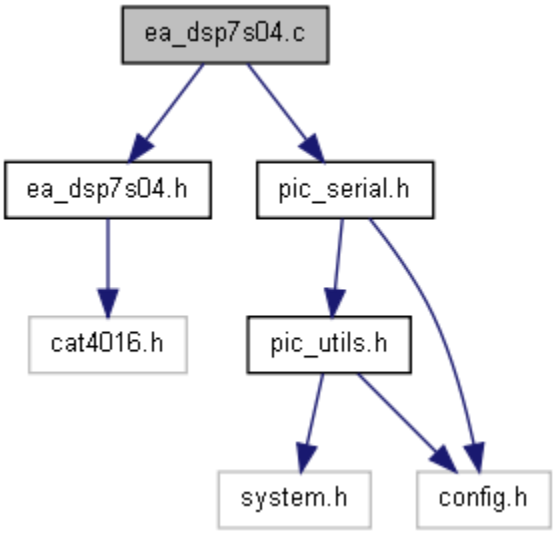
uns16 ea_dsp0801_translate (uns8)

```
29                                              {
30     uns16 result = 0;
31 
32 
33     switch (my_char) {
34 
35         case ' ': result = 0b0000000000000000; break;
36         case '0': result = 0b0010010000111111; break;
37         case '1': result = 0b0000000000000110; break;
38         case '2': result = 0b0000000011011011; break;
39         case '3': result = 0b0000000010001111; break;
40         case '4': result = 0b0001001011100000; break;
41         case '5': result = 0b0000000011101101; break;
42         case '6': result = 0b0000000011111101; break;
43         case '7': result = 0b0000000000000111; break;
44         case '8': result = 0b0000000011111111; break;
45         case '9': result = 0b0000000011101111; break;
46         case 'A': result = 0b0000000011110111; break;
47         case 'B': result = 0b0001001010001111; break;
48         case 'C': result = 0b0000000000111001; break;
49         case 'D': result = 0b0001001000001111; break;
50         case 'E': result = 0b0000000001111001; break;
51         case 'F': result = 0b0000000001110001; break;
52         case 'G': result = 0b0000000010111101; break;
53         case 'H': result = 0b0000000011110110; break;
54         case 'I': result = 0b0001001000001001; break;
55         case 'J': result = 0b0000000000011110; break;
56         case 'K': result = 0b0000110001110000; break;
57         case 'L': result = 0b0000000000111000; break;
58         case 'M': result = 0b0000010100110110; break;
59         case 'N': result = 0b0000100100110110; break;
60         case 'O': result = 0b0000000000111111; break;
61         case 'P': result = 0b0000000011110011; break;
62         case 'Q': result = 0b0000100000111111; break;
63         case 'R': result = 0b0000100011110011; break;
64         case 'S': result = 0b0000000011101101; break;
65         case 'T': result = 0b0001001000000001; break;
66         case 'U': result = 0b0000000000111110; break;
67         case 'V': result = 0b0010010000110000; break;
68         case 'W': result = 0b0010100000110110; break;
69         case 'X': result = 0b0010110100000000; break;
70         case 'Y': result = 0b0001010100000000; break;
71         case 'Z': result = 0b0010010000001001; break;
72         case '*': result = 0b0011111111000000; break;
73 
74     }
75     return result;
76 }
```

Here is the caller graph for this function:

ea_dsp7s04.c File Reference

Include dependency graph for ea_dsp7s04.c:



Functions

- void ea_dsp7s04_clear_dot (uns8 position)
- void ea_dsp7s04_init (void)
- void ea_dsp7s04_print_str (char *str)
- void ea_dsp7s04_set_display (char s, uns8 position)
- void ea_dsp7s04_set_dot (uns8 position)
- void ea_dsp7s04_setup_io (void)
- uns8 ea_dsp7s04_translate (char in)
- void ea_dsp7s04_update_display (void)

Variables

- uns8 display [4]
- uns8 dots

Function Documentation

void ea_dsp7s04_clear_dot (uns8 *position*)

```
126                                             {
127
128     dots = dots & ~(1 << position);
129 }
```

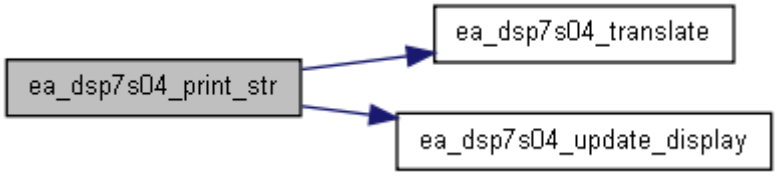
void ea_dsp7s04_init (void)

```
47                              {
48
49      display[0] = 0;
50      display[1] = 0;
51      display[2] = 0;
52      display[3] = 0;
53      dots = 0;
54
55  }
```

void ea_dsp7s04_print_str (char * *str*)

```
132                                         {
133
134 uns8 count;
135
136     for( count = 0; count < 4; count++ ) {
137         display[count] = ea dsp7s04 translate(str[count]);
138     }
139     ea dsp7s04 update display();
140
141 }
```

Here is the call graph for this function:



void ea_dsp7s04_set_display (char *s*, uns8 *position*)

```
113                                                     {
114
115     if (position < 4) {
116         display[position] = ea dsp7s04 translate(s);
117     }
118
119 }
```

Here is the call graph for this function:

void ea_dsp7s04_set_dot (uns8 *position*)

```
121                                         {
122
123     dots = dots | (1 << position);
124 }
```

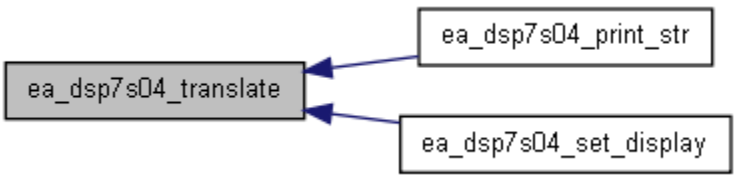
void ea_dsp7s04_setup_io (void)

```
41                                  {
42
43      cat4016 setup io();
44
45  }
```

uns8 ea_dsp7s04_translate (char *in*)

```
57                                      {
58
59  uns8 res;
60
61      switch (in) {
62          //                          R
63          //                          TT
64          case '$' : res = 0b00000010; break;
65          case '0' : res = 0b11111100; break;
66          case '1' : res = 0b01100000; break;
67          case '2' : res = 0b11011010; break;
68          case '3' : res = 0b11110010; break;
69          case '4' : res = 0b01100110; break;
70          case '5' : res = 0b10110110; break;
71          case '6' : res = 0b10111110; break;
72          case '7' : res = 0b11100000; break;
73          case '8' : res = 0b11111110; break;
74          case '9' : res = 0b11110110; break;
75          case 'A' : res = 0b11101110; break;
76          case 'b' : res = 0b00111110; break;
77          case 'c' : res = 0b00011010; break;
78          case 'C' : res = 0b10011100; break;
79          case 'd' : res = 0b01111010; break;
80          case 'E' : res = 0b10011110; break;
81          case 'F' : res = 0b10001110; break;
82          case 'G' : res = 0b10111100; break;
83          case 'h' : res = 0b00101110; break;
84          case 'H' : res = 0b01101110; break;
85          case 'I' : res = 0b01100000; break;
86          case 'J' : res = 0b01110000; break;
87          case 'L' : res = 0b00011100; break;
88          case 'n' : res = 0b00101010; break;
89          case 'N' : res = 0b11101100; break;
90          case 'o' : res = 0b00111010; break;
91          case 'O' : res = 0b11111100; break;
92          case 'P' : res = 0b11001110; break;
93          case 'r' : res = 0b00001010; break;
94          case 'S' : res = 0b10110110; break;
95          case 't' : res = 0b00011110; break;
96          case 'u' : res = 0b00111000; break;
97          case 'U' : res = 0b01111100; break;
98          default  : res = 0b00000000; break;
99      }
100
101      return res;
102
103 }
```

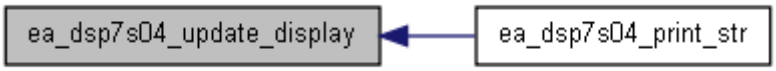
Here is the caller graph for this function:



void ea_dsp7s04_update_display (void)

```
105                                     {
106
107     cat4016_write_data(display[3] | (dots >> 3) , display[2] | ((dots >> 2) & 0x01));
108     cat4016_write_data(display[1] | ((dots >> 1) & 0x01), display[0] | (dots & 0x01));
109     cat4016_latch_data();
110 }
```

Here is the caller graph for this function:



Variable Documentation

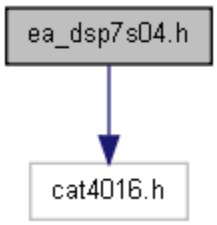
uns8 display[4]

uns8 dots

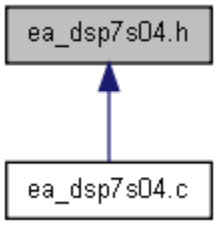
---

ea_dsp7s04.h File Reference

Include dependency graph for ea_dsp7s04.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define ea_dsp7s04_enable_display(on) cat4016_enable_display(on)

Functions

- void ea_dsp7s04_clear_dot (uns8 pos)
- void ea_dsp7s04_init (void)

- void ea_dsp7s04_print_str (char *str)
- void ea_dsp7s04_put_raw (uns8 pos, uns8 data)
- void ea_dsp7s04_set_display (char s, uns8 position)
- void ea_dsp7s04_set_dot (uns8 pos)
- void ea_dsp7s04_setup_io (void)
- void ea_dsp7s04_update_display (void)

Define Documentation

#define ea_dsp7s04_enable_display(on) cat4016_enable_display(on)

Function Documentation

void ea_dsp7s04_clear_dot (uns8 *pos*)

```
126                                             {
127
128     dots = dots & ~(1 << position);
129 }
```

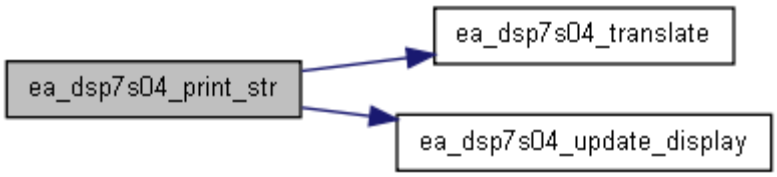
void ea_dsp7s04_init (void)

```
47                              {
48
49     display[0] = 0;
50     display[1] = 0;
51     display[2] = 0;
52     display[3] = 0;
53     dots = 0;
54
55 }
```

void ea_dsp7s04_print_str (char * *str*)

```
132                                         {
133
134 uns8 count;
135
136     for( count = 0; count < 4; count++ ) {
137         display[count] = ea_dsp7s04_translate(str[count]);
138     }
139     ea_dsp7s04_update_display();
140
141 }
```

Here is the call graph for this function:



void ea_dsp7s04_put_raw (uns8 *pos*, uns8 *data*)

void ea_dsp7s04_set_display (char *s*, uns8 *position*)

```
113                                                     {
114
115     if (position < 4) {
116         display[position] = ea_dsp7s04_translate(s);
117     }
118
119 }
```

Here is the call graph for this function:

void ea_dsp7s04_set_dot (uns8 *pos*)

```
121                                         {
122
123     dots = dots | (1 << position);
124 }
```

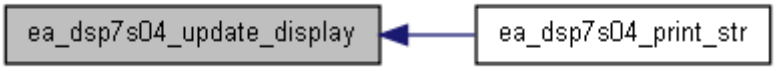
void ea_dsp7s04_setup_io (void)

```
41                                  {
42
43     cat4016_setup_io();
44
45 }
```

void ea_dsp7s04_update_display (void)

```
105                                         {
106
107     cat4016_write_data(display[3] | (dots >> 3) , display[2] | ((dots >> 2) & 0x01));
108     cat4016_write_data(display[1] | ((dots >> 1) & 0x01), display[0] | (dots & 0x01));
109     cat4016_latch_data();
110 }
```

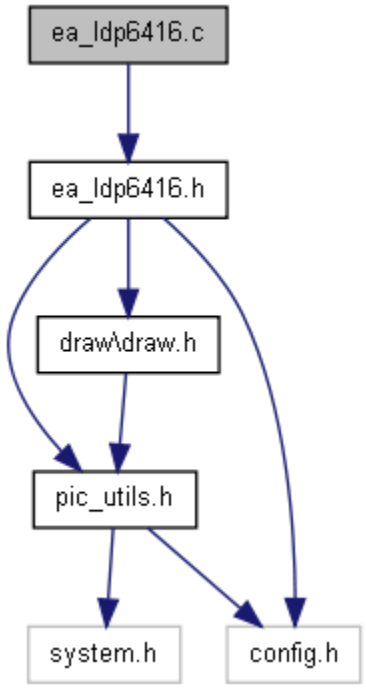
Here is the caller graph for this function:



---

ea_ldp6416.c File Reference

Include dependency graph for ea_ldp6416.c:



Functions

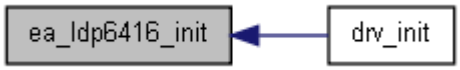
- void ea_ldp6416_init ()
- void ea_ldp6416_setup_io ()

Function Documentation

void ea_ldp6416_init ()

```
63                          {
64
65 }
```

Here is the caller graph for this function:

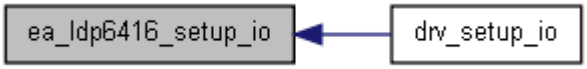


void ea_ldp6416_setup_io ()

```
40                              {
41
42
43     make_output(ea_ldp6416_en_port, ea_ldp6416_en_pin);
44     make_output(ea_ldp6416_r1_port, ea_ldp6416_r1_pin);
45
46     make_output(ea_ldp6416_g1_port, ea_ldp6416_g1_pin);
47
48
49     make_output(ea_ldp6416_a_port, ea_ldp6416_a_pin);
```

```
50      make output(ea_ldp6416_b_port, ea_ldp6416_b_pin);
51      make output(ea_ldp6416_c_port, ea_ldp6416_c_pin);
52      make output(ea_ldp6416_d_port, ea_ldp6416_d_pin);
53
54      make output(ea_ldp6416_s_port, ea_ldp6416_s_pin);
55      make output(ea_ldp6416_l_port, ea_ldp6416_l_pin);
56
57      clear pin(ea_ldp6416_l_port, ea_ldp6416_l_pin);
58      clear pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
59      set pin   (ea_ldp6416_en_port, ea_ldp6416_en_pin);
60
61  }
```

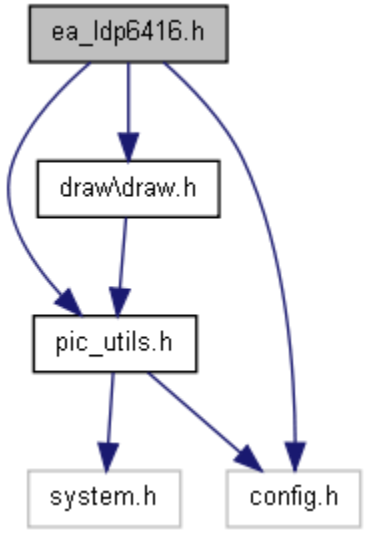
Here is the caller graph for this function:



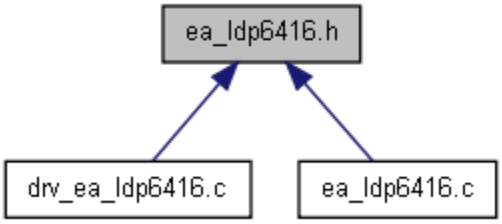
---

ea_ldp6416.h File Reference

Support routines for LDP-6416 LED display panel.
Include dependency graph for ea_ldp6416.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define ea_ldp6416_setup() ea_ldp6416_setup_io()

Functions

- void ea_ldp6416_init ()
- void ea_ldp6416_setup_io ()

Detailed Description

Define Documentation

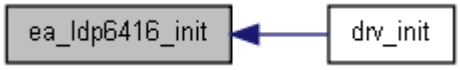
#define ea_ldp6416_setup() ea_ldp6416_setup_io()

Function Documentation

void ea_ldp6416_init ()

```
63                          {
64
65 }
```

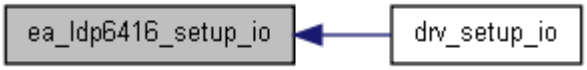
Here is the caller graph for this function:



void ea_ldp6416_setup_io ()

```
40                              {
41
42
43      make_output(ea_ldp6416_en_port, ea_ldp6416_en_pin);
44      make_output(ea_ldp6416_r1_port, ea_ldp6416_r1_pin);
45
46      make_output(ea_ldp6416_g1_port, ea_ldp6416_g1_pin);
47
48
49      make_output(ea_ldp6416_a_port, ea_ldp6416_a_pin);
50      make_output(ea_ldp6416_b_port, ea_ldp6416_b_pin);
51      make_output(ea_ldp6416_c_port, ea_ldp6416_c_pin);
52      make_output(ea_ldp6416_d_port, ea_ldp6416_d_pin);
53
54      make_output(ea_ldp6416_s_port, ea_ldp6416_s_pin);
55      make_output(ea_ldp6416_l_port, ea_ldp6416_l_pin);
56
57      clear_pin(ea_ldp6416_l_port, ea_ldp6416_l_pin);
58      clear_pin(ea_ldp6416_s_port, ea_ldp6416_s_pin);
59      set_pin   (ea_ldp6416_en_port, ea_ldp6416_en_pin);
60
61 }
```

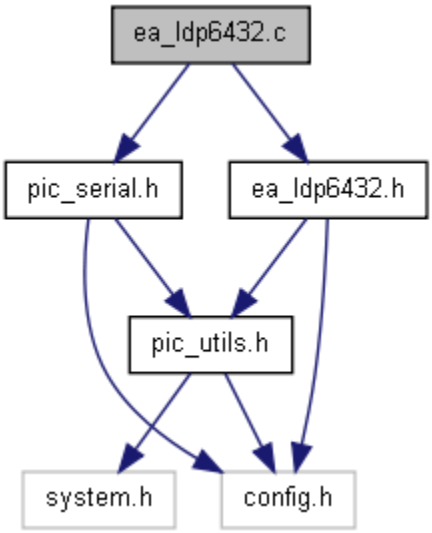
Here is the caller graph for this function:



---

ea_ldp6432.c File Reference

Include dependency graph for ea_ldp6432.c:



Functions

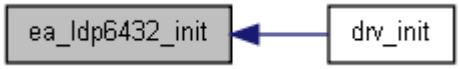
- void ea_ldp6432_init ()
- void ea_ldp6432_setup_io ()

Function Documentation

void ea_ldp6432_init ()

```
67                          {
68
69 }
```

Here is the caller graph for this function:

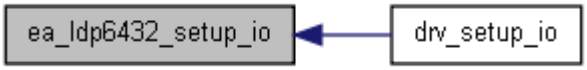


void ea_ldp6432_setup_io ()

```
44                              {
```

```
45
46
47      make_output(ea_ldp6432_en_port, ea_ldp6432_en_pin);
48      make_output(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
49      make_output(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
50      make_output(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
51      make_output(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
52
53      make_output(ea_ldp6432_a_port, ea_ldp6432_a_pin);
54      make_output(ea_ldp6432_b_port, ea_ldp6432_b_pin);
55      make_output(ea_ldp6432_c_port, ea_ldp6432_c_pin);
56      make_output(ea_ldp6432_d_port, ea_ldp6432_d_pin);
57
58      make_output(ea_ldp6432_s_port, ea_ldp6432_s_pin);
59      make_output(ea_ldp6432_l_port, ea_ldp6432_l_pin);
60
61      clear_pin(ea_ldp6432_l_port, ea_ldp6432_l_pin);
62      clear_pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
63      set_pin   (ea_ldp6432_en_port, ea_ldp6432_en_pin);
64
65  }
```

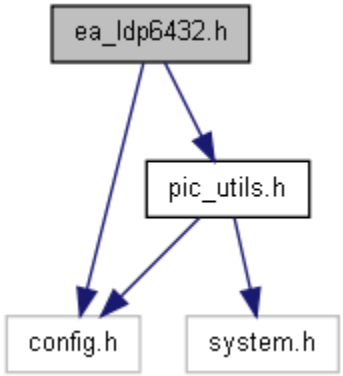
Here is the caller graph for this function:



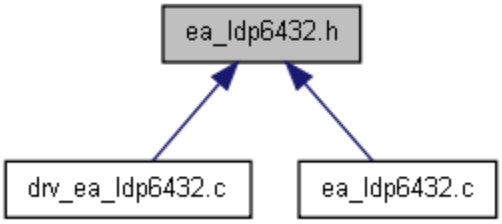
---

ea_ldp6432.h File Reference

Support routines for the LDP-6432 LED display panel.
Include dependency graph for ea_ldp6432.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define ea_ldp6432_setup() ea_ldp6432_setup_io()

Functions

- void ea_ldp6432_init ()
- void ea_ldp6432_setup_io ()

Detailed Description

Define Documentation

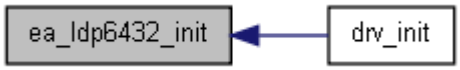
#define ea_ldp6432_setup() ea_ldp6432_setup_io()

Function Documentation

void ea_ldp6432_init ()

```
67                          {
68
69 }
```

Here is the caller graph for this function:

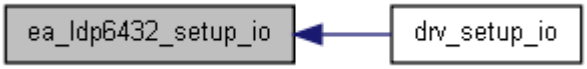


void ea_ldp6432_setup_io ()

```
44                              {
45
46
47      make_output(ea_ldp6432_en_port, ea_ldp6432_en_pin);
48      make_output(ea_ldp6432_r1_port, ea_ldp6432_r1_pin);
49      make_output(ea_ldp6432_r2_port, ea_ldp6432_r2_pin);
50      make_output(ea_ldp6432_g1_port, ea_ldp6432_g1_pin);
51      make_output(ea_ldp6432_g2_port, ea_ldp6432_g2_pin);
52
53      make_output(ea_ldp6432_a_port, ea_ldp6432_a_pin);
54      make_output(ea_ldp6432_b_port, ea_ldp6432_b_pin);
55      make_output(ea_ldp6432_c_port, ea_ldp6432_c_pin);
56      make_output(ea_ldp6432_d_port, ea_ldp6432_d_pin);
57
58      make_output(ea_ldp6432_s_port, ea_ldp6432_s_pin);
59      make_output(ea_ldp6432_l_port, ea_ldp6432_l_pin);
60
61      clear_pin(ea_ldp6432_l_port, ea_ldp6432_l_pin);
62      clear_pin(ea_ldp6432_s_port, ea_ldp6432_s_pin);
63      set_pin   (ea_ldp6432_en_port, ea_ldp6432_en_pin);
64
```

```
65 }
```

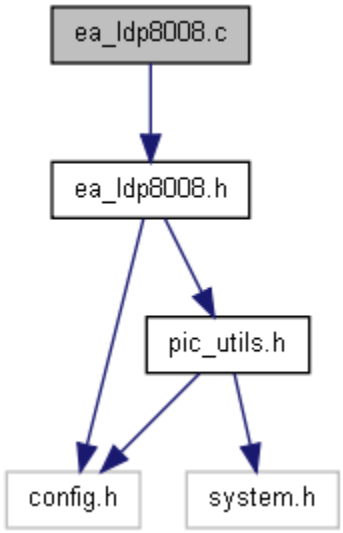
Here is the caller graph for this function:



---

ea_ldp8008.c File Reference

Include dependency graph for ea_ldp8008.c:



Functions

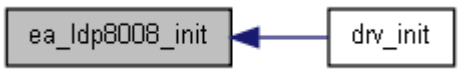
- void ea_ldp8008_init ()
- void ea_ldp8008_setup_io ()

Function Documentation

void ea_ldp8008_init ()

```
60                          {
61     // Should really clear the buffer...
62 }
```

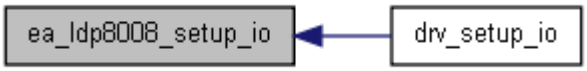
Here is the caller graph for this function:



void ea_ldp8008_setup_io ()

```
40                              {
41
42     make_output(ea_ldp8008_en_port, ea_ldp8008_en_pin);
43     make_output(ea_ldp8008_r_port, ea_ldp8008_r_pin);
44
45     make_output(ea_ldp8008_g_port, ea_ldp8008_g_pin);
46
47     make_output(ea_ldp8008_a_port, ea_ldp8008_a_pin);
48     make_output(ea_ldp8008_b_port, ea_ldp8008_b_pin);
49     make_output(ea_ldp8008_c_port, ea_ldp8008_c_pin);
50
51     make_output(ea_ldp8008_s_port, ea_ldp8008_s_pin);
52     make_output(ea_ldp8008_l_port, ea_ldp8008_l_pin);
53
54     clear_pin(ea_ldp8008_l_port, ea_ldp8008_l_pin);
55     clear_pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
56     clear_pin  (ea_ldp8008_en_port, ea_ldp8008_en_pin);
57
58 }
```

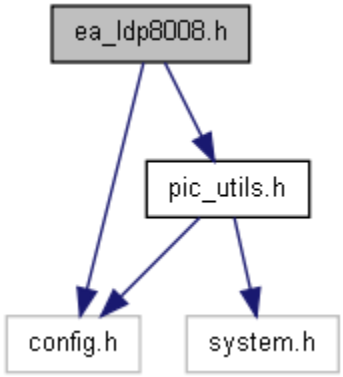
Here is the caller graph for this function:



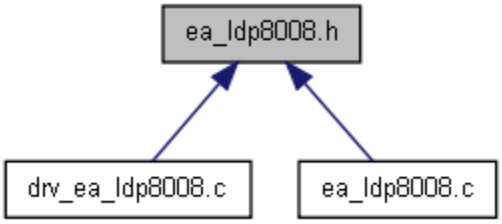
---

ea_ldp8008.h File Reference

Support for LDP-8008 80x08 LED panel.
Include dependency graph for ea_ldp8008.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define ea_ldp8008_setup() ea_ldp8008_setup_io()

Functions

- void ea_ldp8008_init ()
- void ea_ldp8008_setup_io ()

Detailed Description

Define Documentation

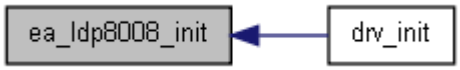
#define ea_ldp8008_setup() ea_ldp8008_setup_io()

Function Documentation

void ea_ldp8008_init ()

```
60                          {
61     // Should really clear the buffer...
62 }
```

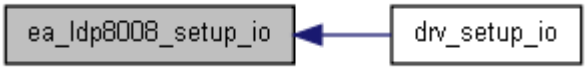
Here is the caller graph for this function:



void ea_ldp8008_setup_io ()

```
40                              {
41
42     make_output(ea_ldp8008_en_port, ea_ldp8008_en_pin);
43     make_output(ea_ldp8008_r_port, ea_ldp8008_r_pin);
44
45     make_output(ea_ldp8008_g_port, ea_ldp8008_g_pin);
46
47     make_output(ea_ldp8008_a_port, ea_ldp8008_a_pin);
48     make_output(ea_ldp8008_b_port, ea_ldp8008_b_pin);
49     make_output(ea_ldp8008_c_port, ea_ldp8008_c_pin);
50
51     make_output(ea_ldp8008_s_port, ea_ldp8008_s_pin);
52     make_output(ea_ldp8008_l_port, ea_ldp8008_l_pin);
53
54     clear_pin(ea_ldp8008_l_port, ea_ldp8008_l_pin);
55     clear_pin(ea_ldp8008_s_port, ea_ldp8008_s_pin);
56     clear_pin  (ea_ldp8008_en_port, ea_ldp8008_en_pin);
57
58 }
```

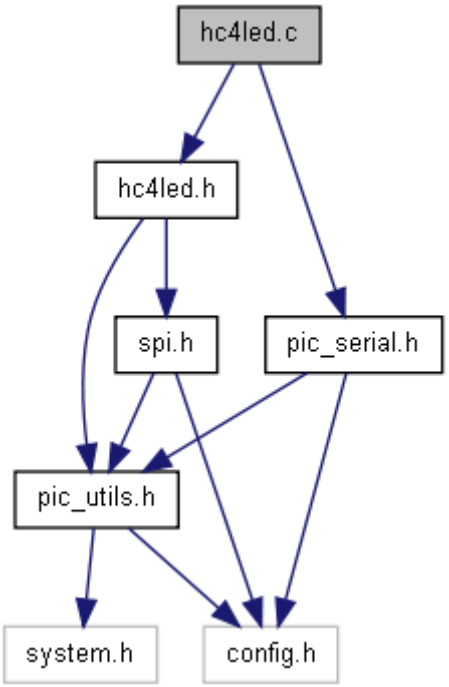
Here is the caller graph for this function:



---

hc4led.c File Reference

Include dependency graph for hc4led.c:



Functions

- uns8 hc4led_convert (uns8 digit)
- void hc4led_setup ()
- void hc4led_write_str (char *data)

Function Documentation

uns8 hc4led_convert (uns8 *digit*)

```
43                                  {
44      switch (digit) {
45      case ' ': return 0;
46      case '0': return 126;
47      case '1': return 24;
48      case '2': return 109;
49      case '3': return 61;
50      case '4': return 27;
51      case '5': return 55;
```

```
52      case '6': return 115;
53      case '7': return 28;
54      case '8': return 127;
55      case '9': return 31+32;
56      case '\'': return 15;
57      }
58  }
```

Here is the caller graph for this function:

void hc4led_setup ()

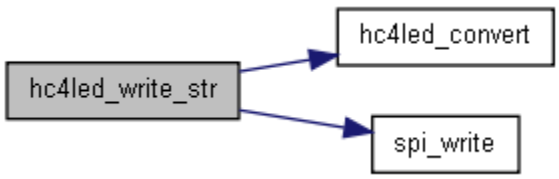
```
39                      {
40      spi_setup();
41  }
```

Here is the call graph for this function:

void hc4led_write_str (char * *data*)

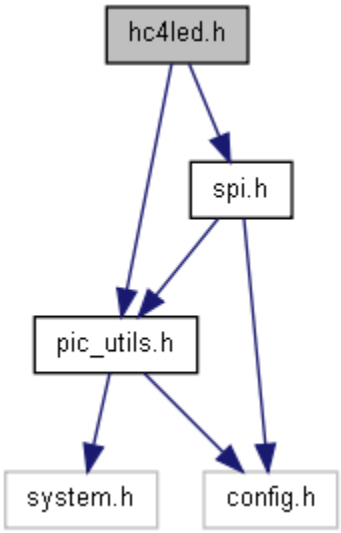
```
60                                      {
61
62  uns8 count, digit;
63  char converted[5];
64
65      count = 4;
66      do {
67          digit = data[count-1];
68          converted[count-1] = hc4led_convert(digit);
69          count--;
70         } while (count > 0);
71      count = 4;
72      do {
73          digit = converted[count-1];
74          spi_write(digit);
75          count--;
76         } while (count > 0);
77
78  }
```

Here is the call graph for this function:

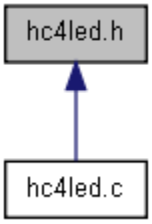


hc4led.h File Reference

Routines to access four digit LED display.
Include dependency graph for hc4led.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define __hc4led_h include

Functions

- void hc4led_setup ()
- void hc4led_write_str (char *data)

Detailed Description

Define Documentation

#define __hc4led_h include

Function Documentation

void hc4led_setup ()

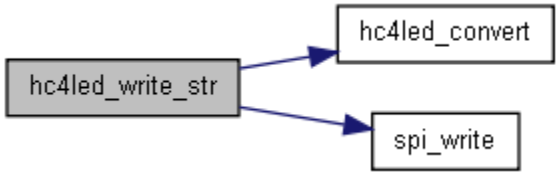
```
39                      {
40      spi_setup();
41  }
```

Here is the call graph for this function:

void hc4led_write_str (char * *data*)

```
60                                      {
61
62  uns8 count, digit;
63  char converted[5];
64
65      count = 4;
66      do {
67          digit = data[count-1];
68          converted[count-1] = hc4led_convert(digit);
69          count--;
70         } while (count > 0);
71      count = 4;
72      do {
73          digit = converted[count-1];
74          spi_write(digit);
75          count--;
76         } while (count > 0);
77
78  }
```

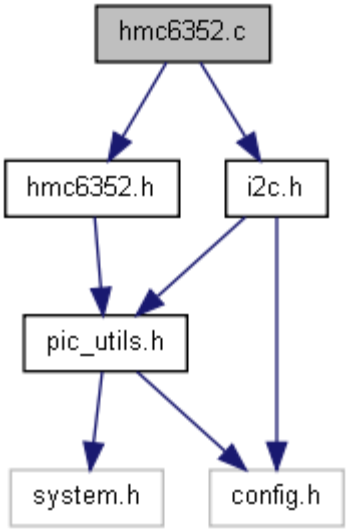
Here is the call graph for this function:



---

hmc6352.c File Reference

Include dependency graph for hmc6352.c:



Functions

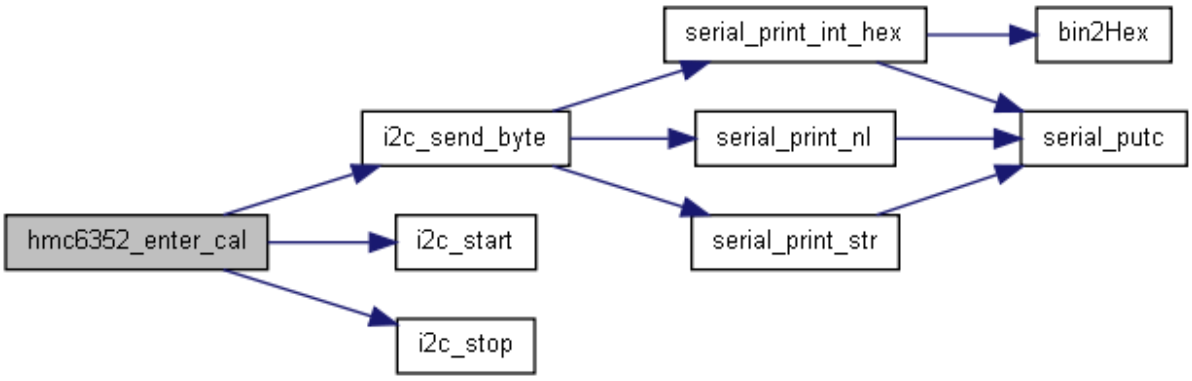
- void hmc6352_enter_cal ()
- void hmc6352_exit_cal ()
- uns16 hmc6352_get_data ()
- uns8 hmc6352_read_eeprom (uns8 addr)
- uns8 hmc6352_read_ram (uns8 addr)
- void hmc6352_save_op_mode ()
- void hmc6352_set_mode (uns8 mode)
- void hmc6352_setup_io ()
- void hmc6352_sleep ()
- void hmc6352_update_bridge_offsets ()
- void hmc6352_wake ()
- void hmc6352_write_eeprom (uns8 addr, uns8 data)
- void hmc6352_write_ram (uns8 addr, uns8 data)

Function Documentation

void hmc6352_enter_cal ()

```
160                             {
161
162     // Send hmc6352 addr (write)
163     // Send command C (Enter user calibration mode)
164
165     i2c_start();
166     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
167     i2c_send_byte(hmc6352_enter_cal_cmd);
168     i2c_stop();
169 }
```

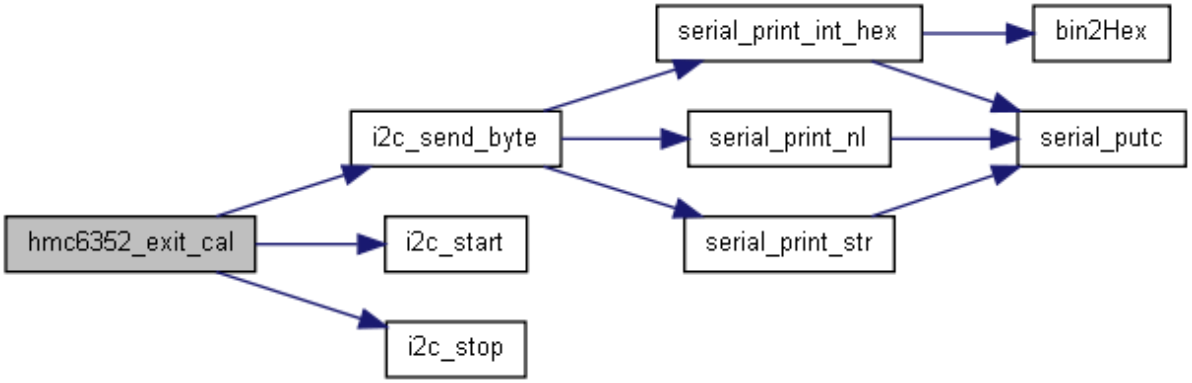
Here is the call graph for this function:



void hmc6352_exit_cal ()

```
171                          {
172
173     // Send hmc6352 addr (write)
174     // Send command E (Exit user calibration mode)
175
176     i2c_start();
177     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
178     i2c_send_byte(hmc6352_exit_cal_cmd);
179     i2c_stop();
180 }
```

Here is the call graph for this function:

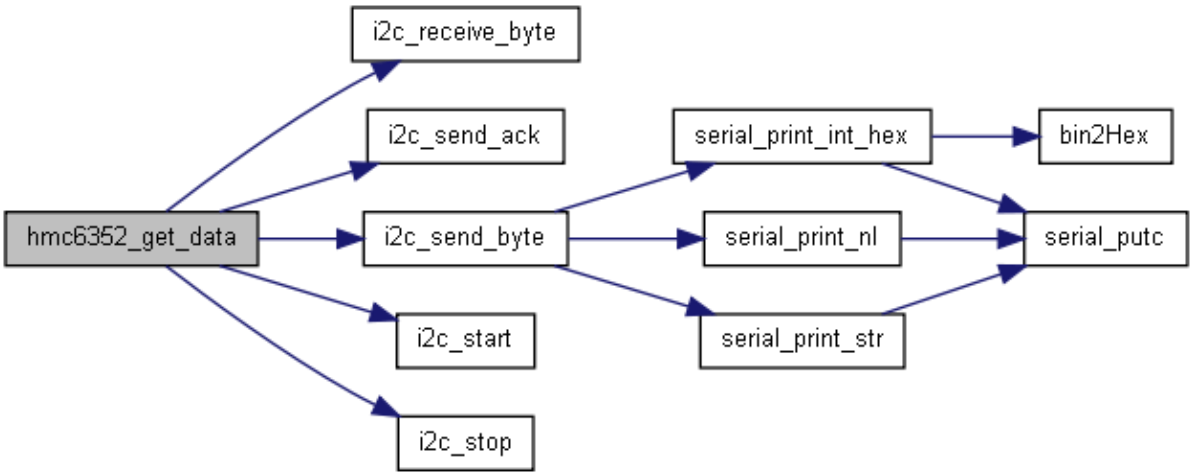


uns16 hmc6352_get_data ()

```
193                          {
194
195     // Send hmc6352 addr (write)
196     // Send command A (get data)
197
198     // Send hmc6352 addr (read)
199     // Read data
200     // Read data
201
202 uns16 data;
203
204     i2c_start();
205     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
206     i2c_send_byte(hmc6352_get_data_cmd);
```

```
207     i2c_stop();
208
209     i2c_start();
210     i2c_send_byte(hmc6352_device_addr | hmc6352_read);
211     data = i2c_receive_byte();
212     delay_ms(1);
213     i2c_send_ack();
214     data = data << 8;
215     data = data + i2c_receive_byte();
216     delay_ms(1);
217     i2c_send_ack();
218     i2c_stop();
219
220     return data;
221 }
```

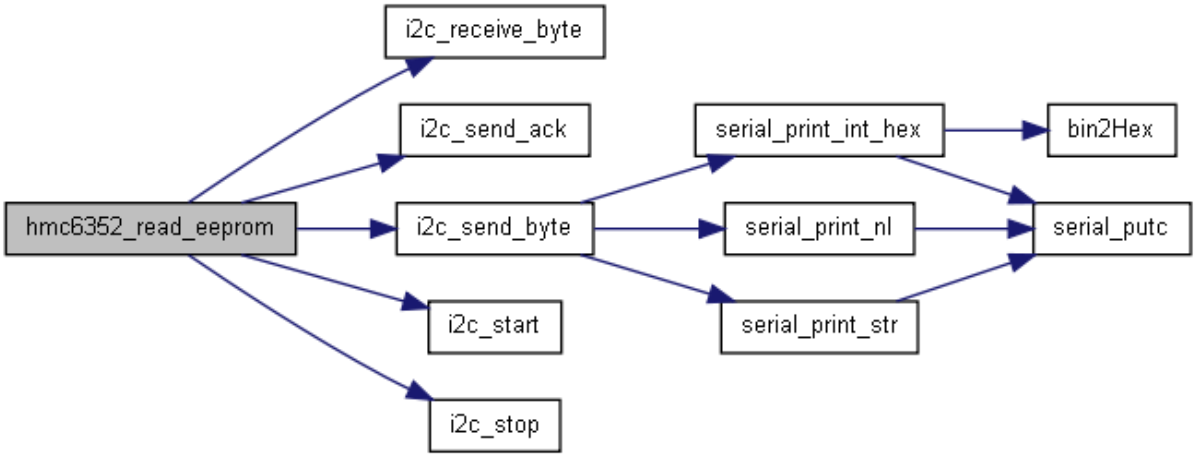
Here is the call graph for this function:



uns8 hmc6352_read_eeprom (uns8 *addr*)

```
55                                      {
56
57     // Send hmc6352 addr (write)
58     // Send command r (read from eeprom)
59     // Send eeprom addr
60
61     // Send hmc6352 addr (read)
62     // Read data
63
64 uns8 data;
65
66     i2c_start();
67     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
68     i2c_send_byte(hmc6352_read_from_eeprom);
69     i2c_send_byte(addr);
70     i2c_stop();
71
72     i2c_start();
73     i2c_send_byte(hmc6352_device_addr | hmc6352_read);
74     data = i2c_receive_byte();
75     i2c_send_ack();
76     i2c_stop();
77
78     return data;
79 }
```

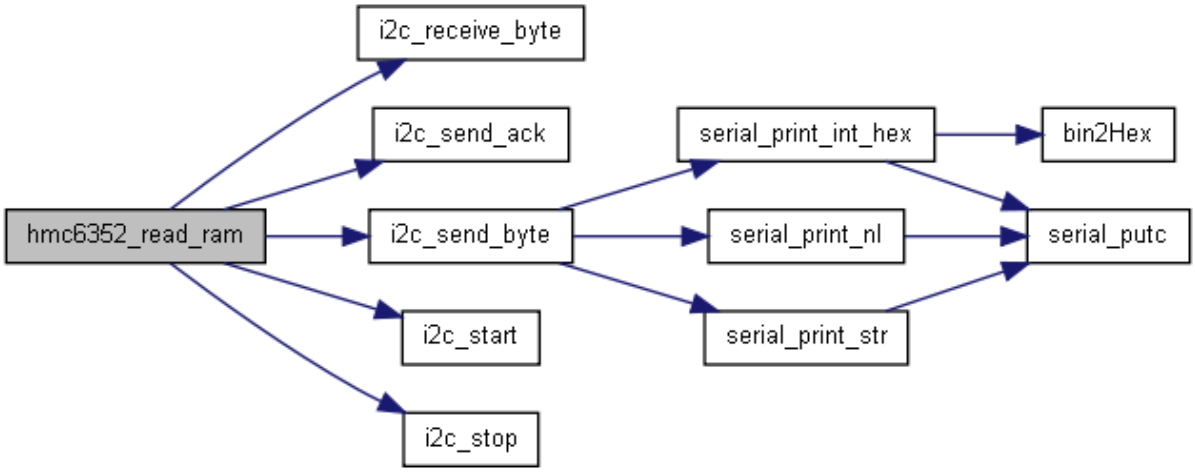
Here is the call graph for this function:



uns8 hmc6352_read_ram (uns8 *addr*)

```
99                                  {
100
101     // Send hmc6352 addr (write)
102     // Send command g (read from ram)
103     // Send ram addr
104
105     // Send hmc6352 addr (read)
106     // Read data
107
108 uns8 data;
109
110     i2c_start();
111     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
112     i2c_send_byte(hmc6352_read_from_ram);
113     i2c_send_byte(addr);
114     i2c_stop();
115
116     i2c_start();
117     i2c_send_byte(hmc6352_device_addr | hmc6352_read);
118     data = i2c_receive_byte();
119     i2c_send_ack();
120     i2c_stop();
121
122     return data;
123 }
```

Here is the call graph for this function:

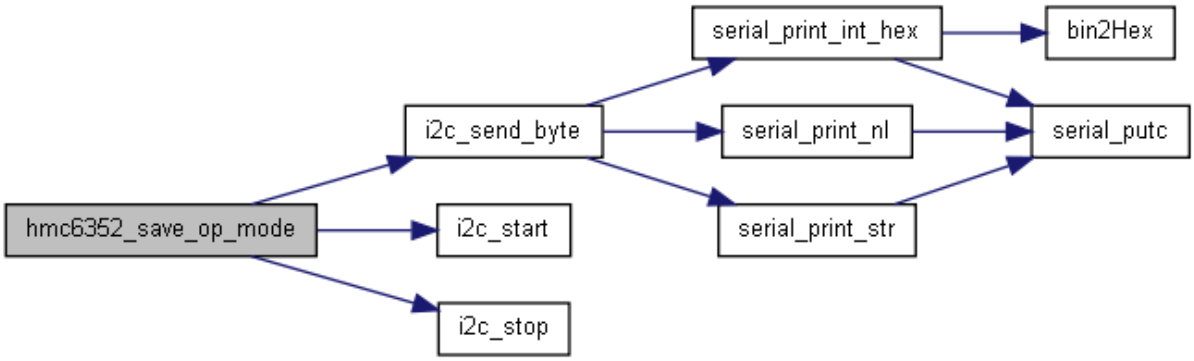


Here is the caller graph for this function:

void hmc6352_save_op_mode ()

```
182                                 {
183     // Send hmc6352 addr (write)
184     // Send command L (Save op mode to eeprom)
185
186     i2c_start();
187     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
188     i2c_send_byte(hmc6352_save_op_mode_cmd);
189     i2c_stop();
190
191 }
```

Here is the call graph for this function:

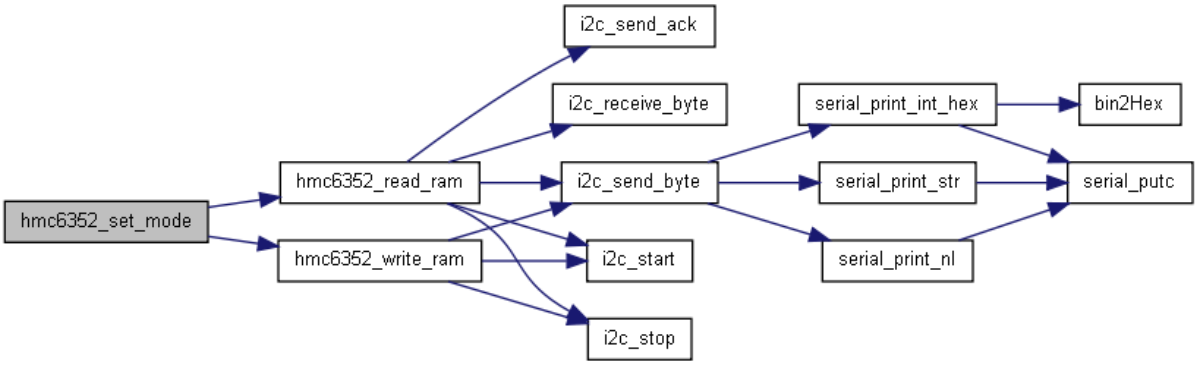


void hmc6352_set_mode (uns8 *mode*)

```
225                                     {
226
227 uns8 data;
228     data = hmc6352_read_ram(0x74);
```

```
229
230     switch (mode) {
231         case hmc6352_mode_standby:
232             data.1 = 0;
233             data.0 = 0;
234             break;
235         case hmc6352_mode_query:
236             data.1 = 0;
237             data.0 = 1;
238             break;
239         case hmc6352_mode_continuous:
240             data.1 = 1;
241             data.0 = 0;
242             break;
243     }
244     hmc6352_write_ram(0x74, data);
245 }
```

Here is the call graph for this function:



void hmc6352_setup_io ()

```
247                         {
248     i2c_setup_io();
```

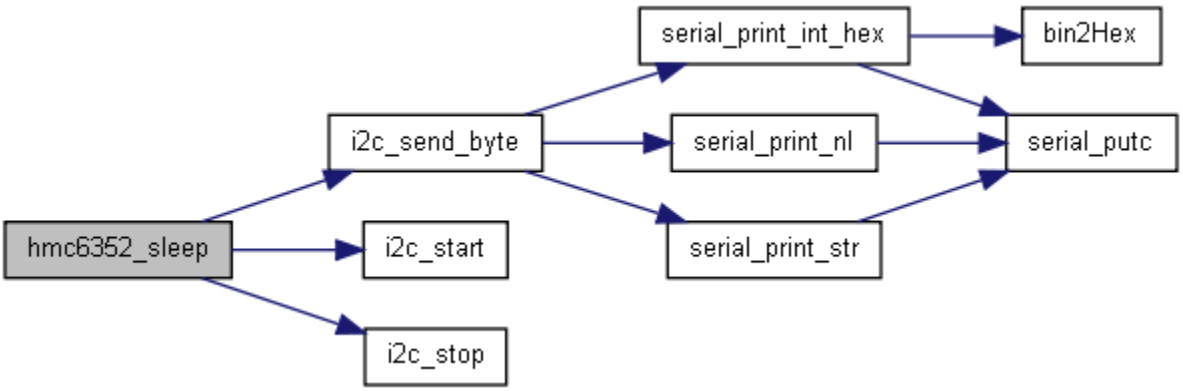
249 }

Here is the call graph for this function:

void hmc6352_sleep ()

```
125                     {
126
127     // Send hmc6352 addr (write)
128     // Send command S (sleep)
129
130     i2c_start();
131     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
132     i2c_send_byte(hmc6352_sleep_cmd);
133     i2c_stop();
134 }
```

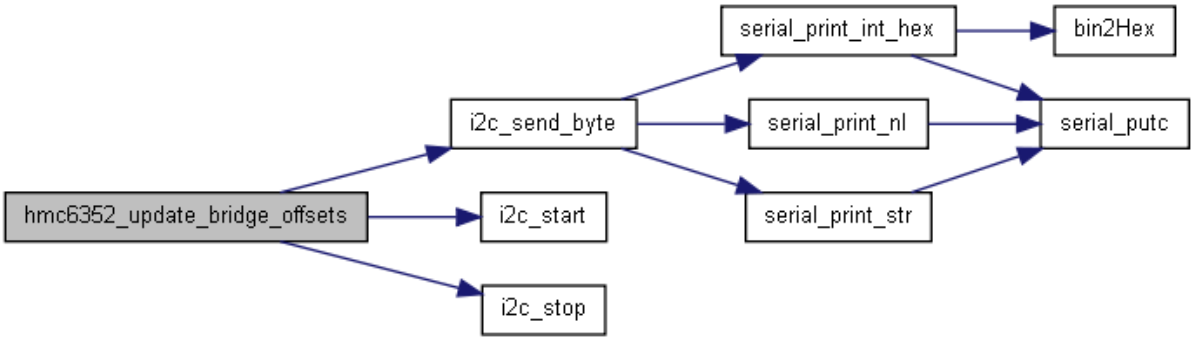
Here is the call graph for this function:



void hmc6352_update_bridge_offsets ()

```
149                                     {
150
151     // Send hmc6352 addr (write)
152     // Send command O (update bridge offsets)
153
154     i2c_start();
155     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
156     i2c_send_byte(hmc6352_update_bridge_cmd);
157     i2c_stop();
158 }
```

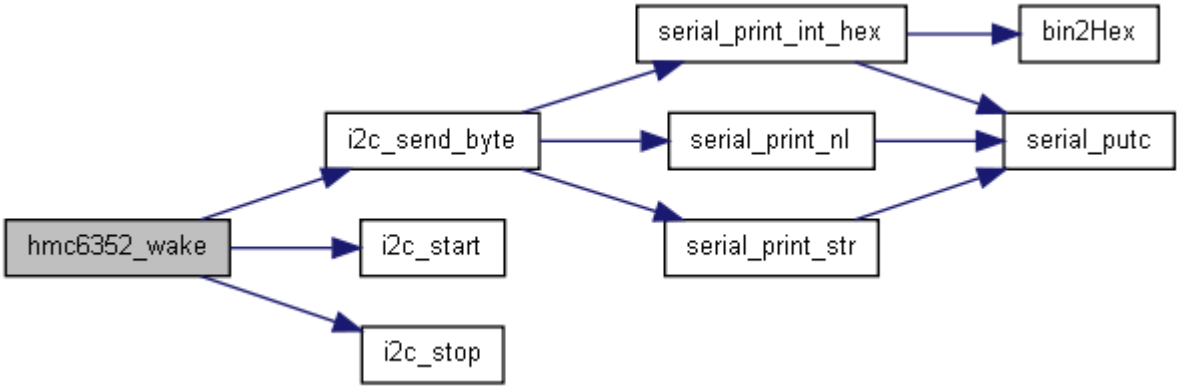
Here is the call graph for this function:



void hmc6352_wake ()

```
137                     {
138
139     // Send hmc6352 addr (write)
140     // Send command W (wake)
141
142     i2c_start();
143     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
144     i2c_send_byte(hmc6352_wake_cmd);
145     i2c_stop();
146
147 }   //37208656
```

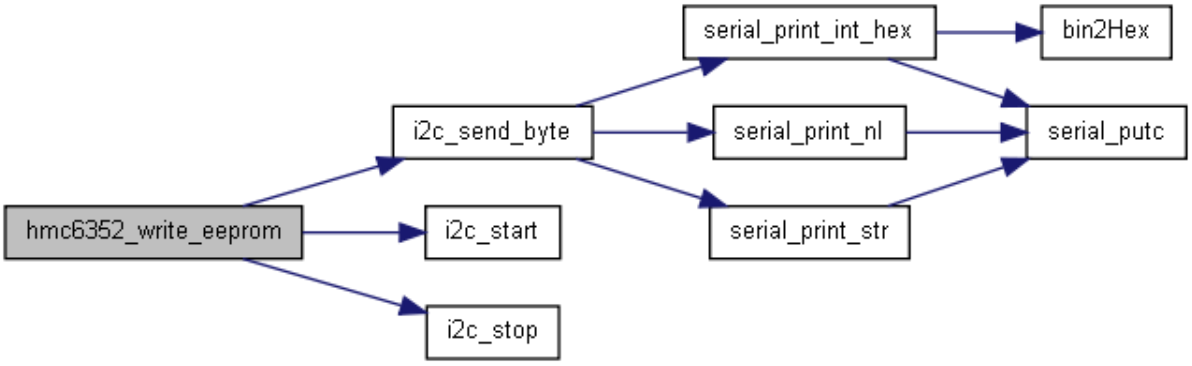
Here is the call graph for this function:



void hmc6352_write_eeprom (uns8 *addr*, uns8 *data*)

```
40                                                  {
41
42      // Send hmc6352 addr (write)
43      // Send command w (write to eeprom)
44      // Send eeprom addr
45      // Send data
46
47      i2c_start();
48      i2c_send_byte(hmc6352_device_addr | hmc6352_write);
49      i2c_send_byte(hmc6352_write_to_eeprom);
50      i2c_send_byte(addr);
51      i2c_send_byte(data);
52      i2c_stop();
53  }
```

Here is the call graph for this function:

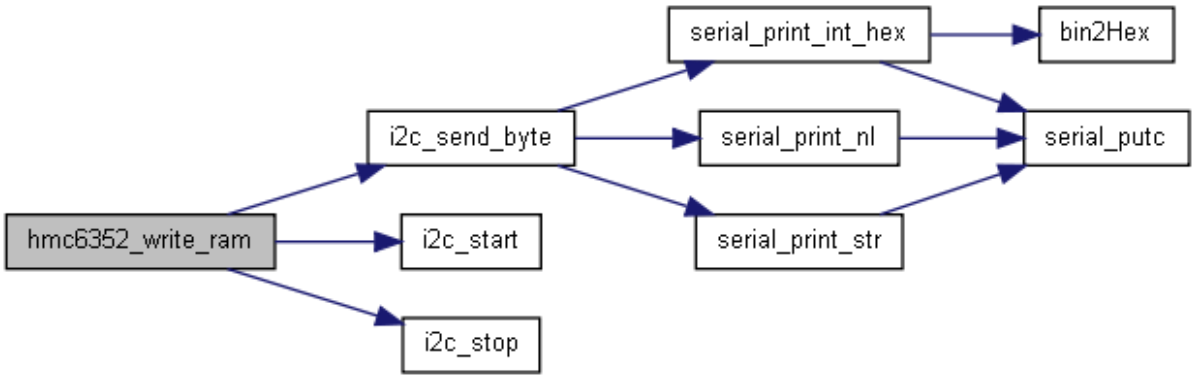


void hmc6352_write_ram (uns8 *addr*, uns8 *data*)

```
82                                                  {
83
84      // Send hmc6352 addr (write)
85      // Send command G (write to ram)
86      // Send ram addr
87      // Send data
88
89      i2c_start();
90      i2c_send_byte(hmc6352_device_addr | hmc6352_write);
91      i2c_send_byte(hmc6352_write_to_ram);
```

```
92      i2c_send_byte(addr);
93      i2c_send_byte(data);
94      i2c_stop();
95
96  }
```

Here is the call graph for this function:

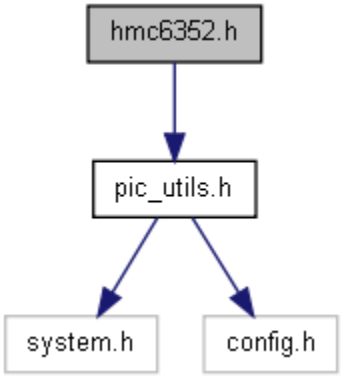


Here is the caller graph for this function:

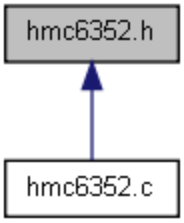
---

hmc6352.h File Reference

Routines for communicating with the hmc6352 digital compass.
Include dependency graph for hmc6352.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define hmc6352_device_addr 0x42
- #define hmc6352_ee_slave_addr 0x00
- #define hmc6352_ee_time_delay 0x05
- #define hmc6352_ee_x_offset_lsb 0x02
- #define hmc6352_ee_x_offset_msb 0x01
- #define hmc6352_ee_y_offset_lsb 0x04
- #define hmc6352_ee_y_offset_msb 0x03
- #define hmc6352_enter_cal_cmd 0x43
- #define hmc6352_exit_cal_cmd 0x45
- #define hmc6352_get_data_cmd 0x41
- #define hmc6352_mode_continuous 0x02
- #define hmc6352_mode_query 0x01
- #define hmc6352_mode_standby 0x00
- #define hmc6352_num_summed 0x06
- #define hmc6352_op_mode 0x08
- #define hmc6352_ram_op_mode_control 0x74
- #define hmc6352_ram_output_data_control 0x4e
- #define hmc6352_read 0x01
- #define hmc6352_read_from_eeprom 0x72
- #define hmc6352_read_from_ram 0x67
- #define hmc6352_save_op_mode_cmd 0x4c
- #define hmc6352_sleep_cmd 0x53
- #define hmc6352_software_ver 0x07
- #define hmc6352_update_bridge_cmd 0x4F
- #define hmc6352_wake_cmd 0x57
- #define hmc6352_write 0x00
- #define hmc6352_write_to_eeprom 0x77
- #define hmc6352_write_to_ram 0x47

Functions

- void hmc6352_enter_cal ()
- void hmc6352_exit_cal ()
- uns16 hmc6352_get_data ()
- uns8 hmc6352_read_eeprom (uns8 addr)
- uns8 hmc6352_read_ram (uns8 addr)
- void hmc6352_save_op_mode ()
- void hmc6352_set_mode (uns8 mode)
- void hmc6352_setup_io ()
- void hmc6352_sleep ()
- void hmc6352_update_bridge_offsets ()
- void hmc6352_wake ()

- void hmc6352_write_eeprom (uns8 addr, uns8 data)
- void hmc6352_write_ram (uns8 addr, uns8 data)

Detailed Description

Define Documentation

#define hmc6352_device_addr 0x42

#define hmc6352_ee_slave_addr 0x00

#define hmc6352_ee_time_delay 0x05

#define hmc6352_ee_x_offset_lsb 0x02

#define hmc6352_ee_x_offset_msb 0x01

#define hmc6352_ee_y_offset_lsb 0x04

#define hmc6352_ee_y_offset_msb 0x03

#define hmc6352_enter_cal_cmd 0x43

#define hmc6352_exit_cal_cmd 0x45

#define hmc6352_get_data_cmd 0x41

#define hmc6352_mode_continuous 0x02

#define hmc6352_mode_query 0x01

#define hmc6352_mode_standby 0x00

#define hmc6352_num_summed 0x06

#define hmc6352_op_mode 0x08

#define hmc6352_ram_op_mode_control 0x74

#define hmc6352_ram_output_data_control 0x4e

#define hmc6352_read 0x01

#define hmc6352_read_from_eeprom 0x72

#define hmc6352_read_from_ram 0x67

#define hmc6352_save_op_mode_cmd 0x4c

#define hmc6352_sleep_cmd 0x53

#define hmc6352_software_ver 0x07

#define hmc6352_update_bridge_cmd 0x4F

#define hmc6352_wake_cmd 0x57

#define hmc6352_write 0x00

#define hmc6352_write_to_eeprom 0x77

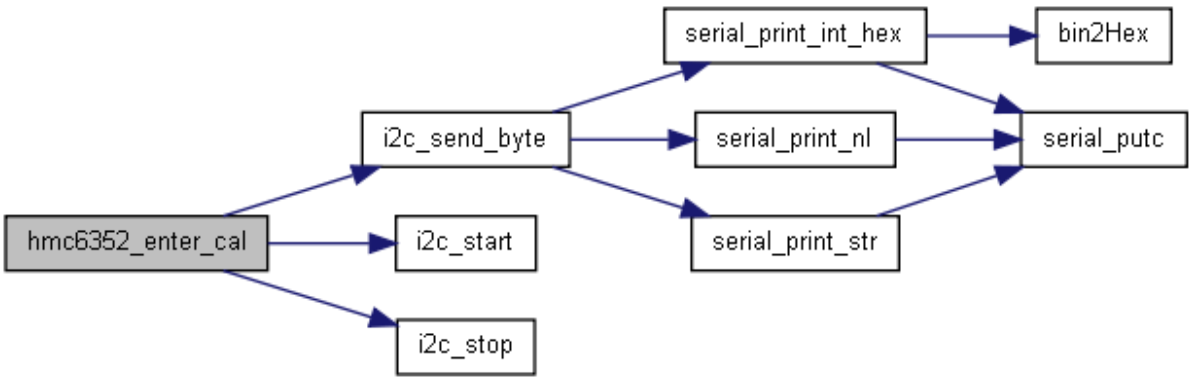
#define hmc6352_write_to_ram 0x47

Function Documentation

void hmc6352_enter_cal ()

```
160                           {
161
162     // Send hmc6352 addr (write)
163     // Send command C (Enter user calibration mode)
164
165     i2c_start();
166     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
167     i2c_send_byte(hmc6352_enter_cal_cmd);
168     i2c_stop();
169 }
```

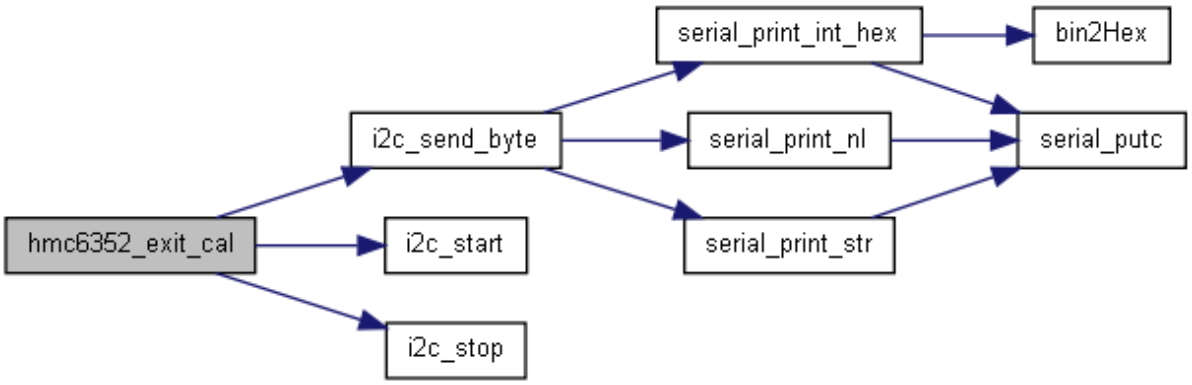
Here is the call graph for this function:



void hmc6352_exit_cal ()

```
171                           {
172
173     // Send hmc6352 addr (write)
174     // Send command E (Exit user calibration mode)
175
176     i2c_start();
177     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
178     i2c_send_byte(hmc6352_exit_cal_cmd);
179     i2c_stop();
180 }
```

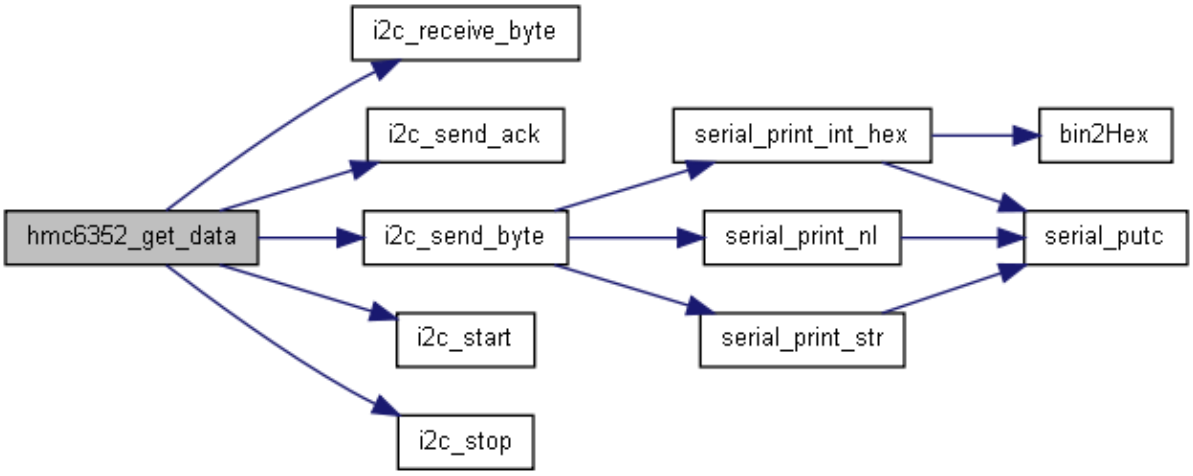
Here is the call graph for this function:



uns16 hmc6352_get_data ()

```
193                         {
194
195     // Send hmc6352 addr (write)
196     // Send command A (get data)
197
198     // Send hmc6352 addr (read)
199     // Read data
200     // Read data
201
202 uns16 data;
203
204     i2c_start();
205     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
206     i2c_send_byte(hmc6352_get_data_cmd);
207     i2c_stop();
208
209     i2c_start();
210     i2c_send_byte(hmc6352_device_addr | hmc6352_read);
211     data = i2c_receive_byte();
212     delay_ms(1);
213     i2c_send_ack();
214     data = data << 8;
215     data = data + i2c_receive_byte();
216     delay_ms(1);
217     i2c_send_ack();
218     i2c_stop();
219
220     return data;
221 }
```

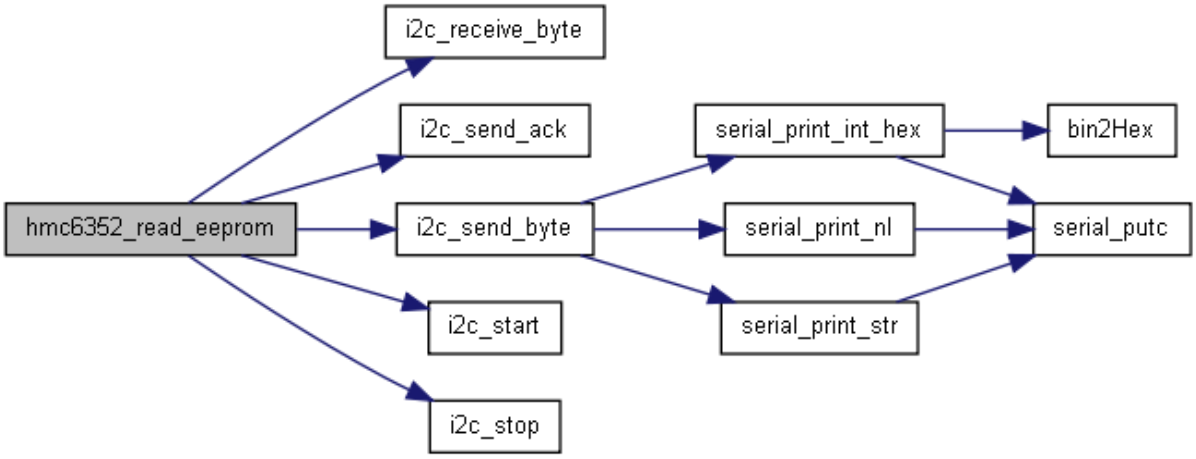
Here is the call graph for this function:



uns8 hmc6352_read_eeprom (uns8 *addr*)

```
55                                      {
56
57      // Send hmc6352 addr (write)
58      // Send command r (read from eeprom)
59      // Send eeprom addr
60
61      // Send hmc6352 addr (read)
62      // Read data
63
64  uns8 data;
65
66      i2c_start();
67      i2c_send_byte(hmc6352_device_addr | hmc6352_write);
68      i2c_send_byte(hmc6352_read_from_eeprom);
69      i2c_send_byte(addr);
70      i2c_stop();
71
72      i2c_start();
73      i2c_send_byte(hmc6352_device_addr | hmc6352_read);
74      data = i2c_receive_byte();
75      i2c_send_ack();
76      i2c_stop();
77
78      return data;
79  }
```

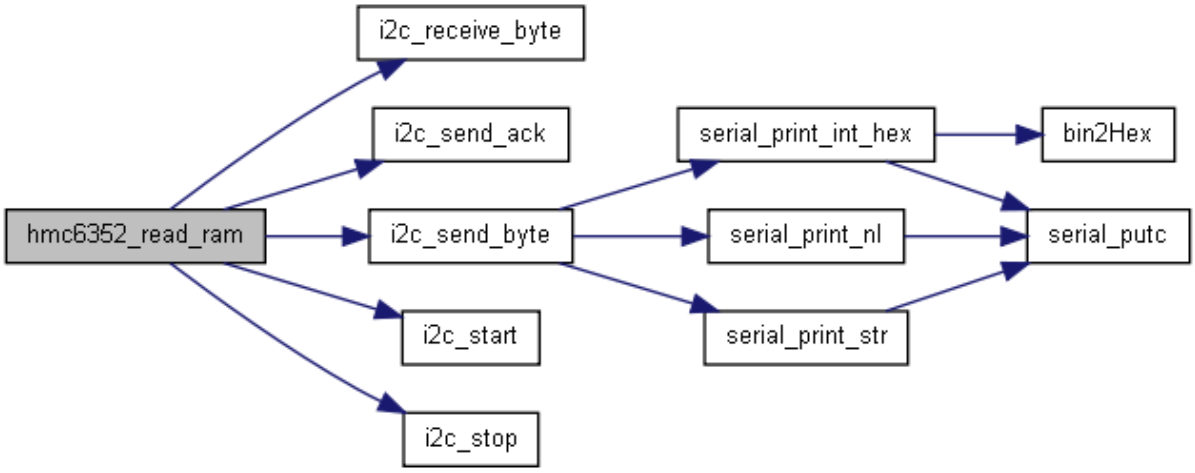
Here is the call graph for this function:



uns8 hmc6352_read_ram (uns8 *addr*)

```
99                                  {
100
101     // Send hmc6352 addr (write)
102     // Send command g (read from ram)
103     // Send ram addr
104
105     // Send hmc6352 addr (read)
106     // Read data
107
108 uns8 data;
109
110     i2c_start();
111     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
112     i2c_send_byte(hmc6352_read_from_ram);
113     i2c_send_byte(addr);
114     i2c_stop();
115
116     i2c_start();
117     i2c_send_byte(hmc6352_device_addr | hmc6352_read);
118     data = i2c_receive_byte();
119     i2c_send_ack();
120     i2c_stop();
121
122     return data;
123 }
```

Here is the call graph for this function:

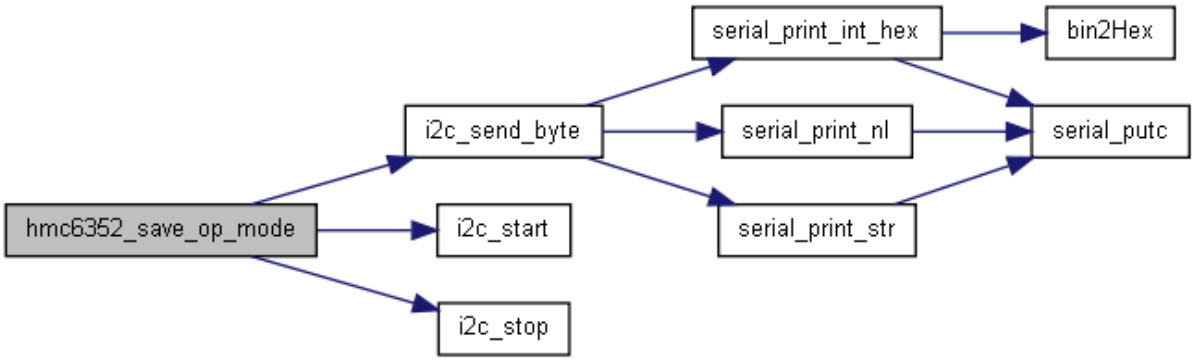


Here is the caller graph for this function:

void hmc6352_save_op_mode ()

```
182                                 {
183     // Send hmc6352 addr (write)
184     // Send command L (Save op mode to eeprom)
185
186     i2c_start();
187     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
188     i2c_send_byte(hmc6352_save_op_mode_cmd);
189     i2c_stop();
190
191 }
```

Here is the call graph for this function:

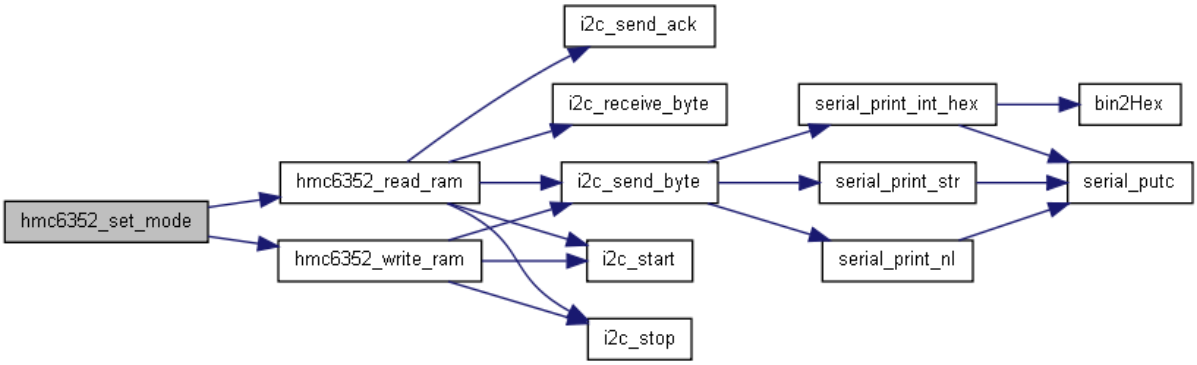


void hmc6352_set_mode (uns8 *mode*)

```
225                                     {
226
227 uns8 data;
228     data = hmc6352_read_ram(0x74);
```

```
229
230     switch (mode) {
231         case hmc6352_mode_standby:
232             data.1 = 0;
233             data.0 = 0;
234             break;
235         case hmc6352_mode_query:
236             data.1 = 0;
237             data.0 = 1;
238             break;
239         case hmc6352_mode_continuous:
240             data.1 = 1;
241             data.0 = 0;
242             break;
243     }
244     hmc6352_write_ram(0x74, data);
245 }
```

Here is the call graph for this function:



void hmc6352_setup_io ()

```
247                         {
248     i2c_setup_io();
```

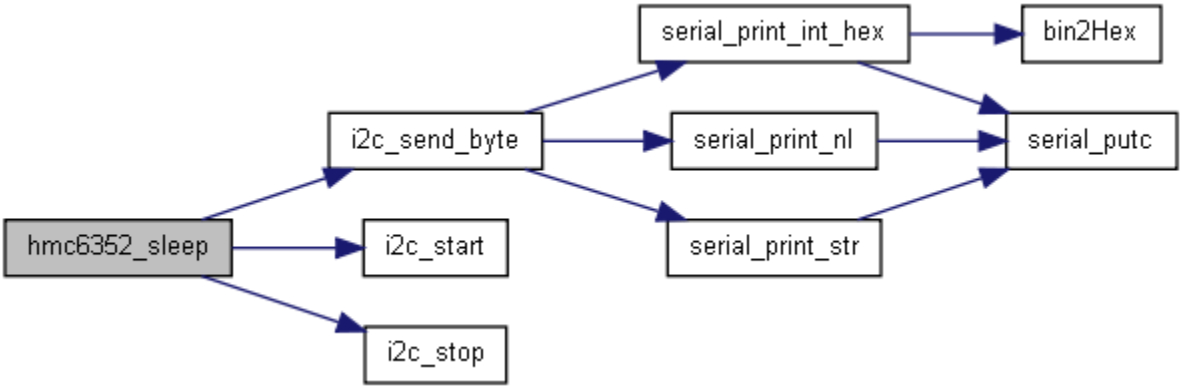
249 }

Here is the call graph for this function:

void hmc6352_sleep ()

```
125                     {
126
127     // Send hmc6352 addr (write)
128     // Send command S (sleep)
129
130     i2c_start();
131     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
132     i2c_send_byte(hmc6352_sleep_cmd);
133     i2c_stop();
134 }
```

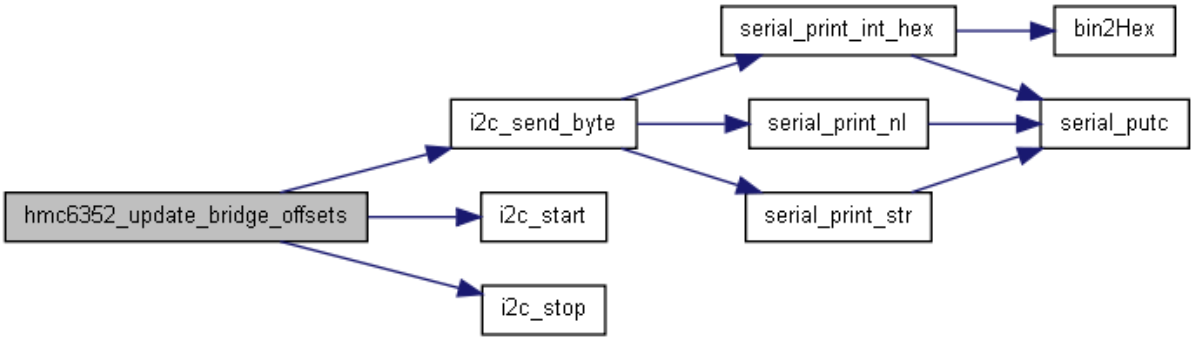
Here is the call graph for this function:



void hmc6352_update_bridge_offsets ()

```
149                                     {
150
151     // Send hmc6352 addr (write)
152     // Send command O (update bridge offsets)
153
154     i2c_start();
155     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
156     i2c_send_byte(hmc6352_update_bridge_cmd);
157     i2c_stop();
158 }
```

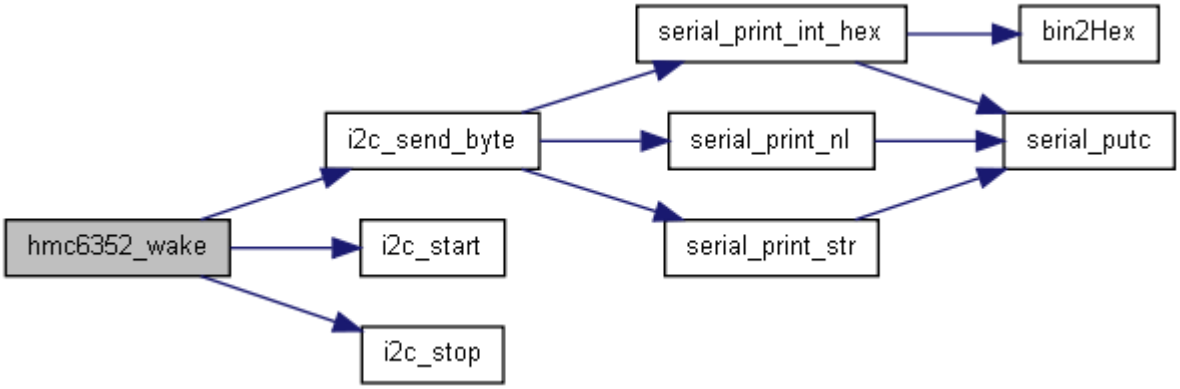
Here is the call graph for this function:



void hmc6352_wake ()

```
137                     {
138
139     // Send hmc6352 addr (write)
140     // Send command W (wake)
141
142     i2c_start();
143     i2c_send_byte(hmc6352_device_addr | hmc6352_write);
144     i2c_send_byte(hmc6352_wake_cmd);
145     i2c_stop();
146
147 }   //37208656
```

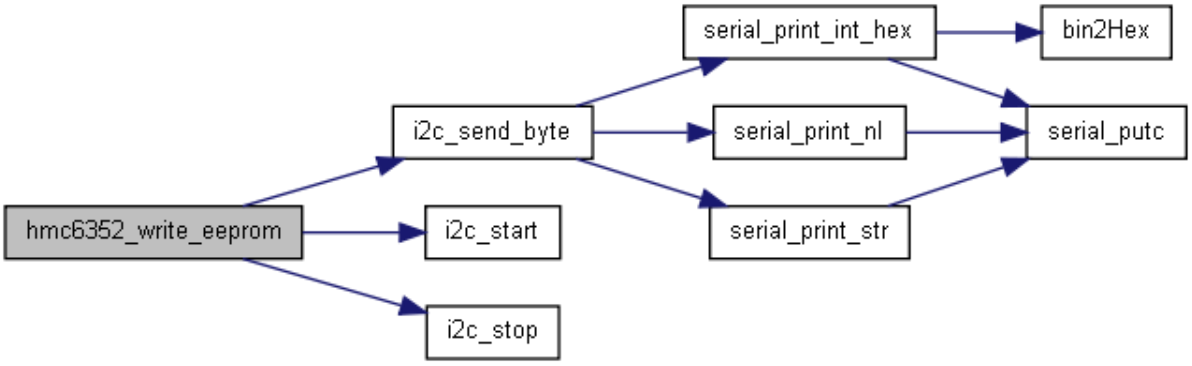
Here is the call graph for this function:



void hmc6352_write_eeprom (uns8 *addr*, uns8 *data*)

```
40                                                  {
41
42      // Send hmc6352 addr (write)
43      // Send command w (write to eeprom)
44      // Send eeprom addr
45      // Send data
46
47      i2c_start();
48      i2c_send_byte(hmc6352_device_addr | hmc6352_write);
49      i2c_send_byte(hmc6352_write_to_eeprom);
50      i2c_send_byte(addr);
51      i2c_send_byte(data);
52      i2c_stop();
53  }
```

Here is the call graph for this function:

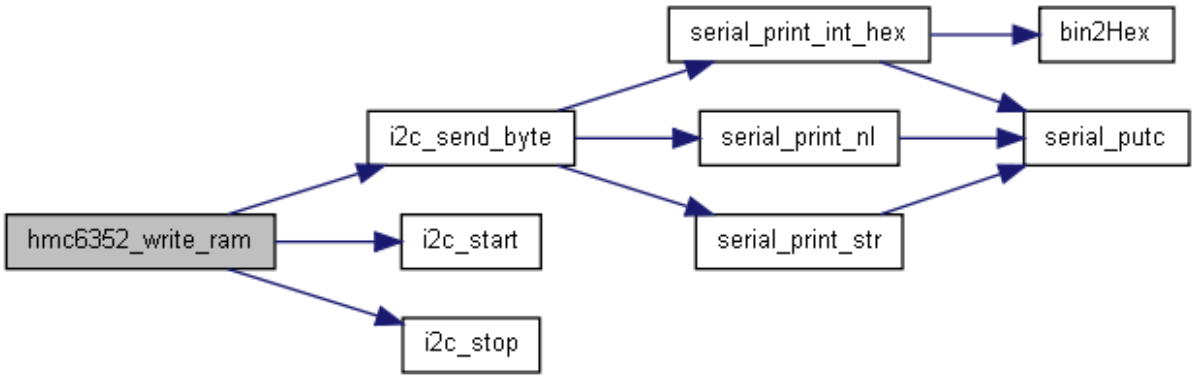


void hmc6352_write_ram (uns8 *addr*, uns8 *data*)

```
82                                                  {
83
84      // Send hmc6352 addr (write)
85      // Send command G (write to ram)
86      // Send ram addr
87      // Send data
88
89      i2c_start();
90      i2c_send_byte(hmc6352_device_addr | hmc6352_write);
91      i2c_send_byte(hmc6352_write_to_ram);
```

```
92      i2c_send_byte(addr);
93      i2c_send_byte(data);
94      i2c_stop();
95
96  }
```

Here is the call graph for this function:

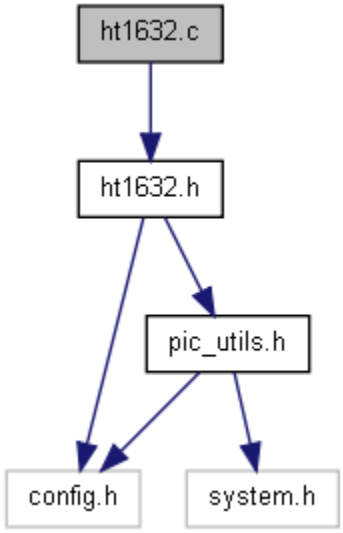


Here is the caller graph for this function:

---

ht1632.c File Reference

Include dependency graph for ht1632.c:



Functions

- void ht1632_fill (uns8 colour)
- void ht1632_fill2 (uns8 colour)

- void ht1632_init (uns8 hw_config)
- void ht1632_send_command (uns8 command)
- void ht1632_set_brightness (uns8 brightness)
- void ht1632_set_pixel (uns8 x, uns8 y, uns8 colour)
- void ht1632_setup_io ()
- void ht1632_write (uns8 mem_addr, uns8 data)

Function Documentation

void ht1632_fill (uns8 *colour*)

```
326                                 {
327     uns8 mem_address;
328     uns8 fill;
329 
330     if (colour) {
331         fill = 0b00001111;
332     } else {
333         fill = 0b00000000;
334     }
335 
336     for(mem_address = 0 ; mem_address < 96 ; mem_address++) {
337         ht1632_write(mem_address, fill);
338     }
339 }
```

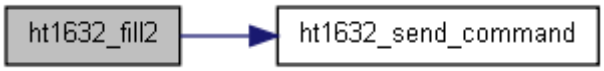
Here is the call graph for this function:

void ht1632_fill2 (uns8 *colour*)

```
341                                 {
342 
343 uns16 count;
344     ht1632_send_command(HT1632_CMD_LEDS_OFF);
345     clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
346 
347     // send WR command
348 
349     // send 1
350     set_pin   (ht1632_data_port, ht1632_data_pin);
351     // pulse wr
352     clear_pin(ht1632_wr_port, ht1632_wr_pin);
353     set_pin   (ht1632_wr_port, ht1632_wr_pin);
354 
355     // send 0
356     clear_pin   (ht1632_data_port, ht1632_data_pin);
357     // pulse wr
358     clear_pin(ht1632_wr_port, ht1632_wr_pin);
359     set_pin   (ht1632_wr_port, ht1632_wr_pin);
360 
361     // send 1
362     set_pin   (ht1632_data_port, ht1632_data_pin);
363     // pulse wr
364     clear_pin(ht1632_wr_port, ht1632_wr_pin);
365     set_pin   (ht1632_wr_port, ht1632_wr_pin);
366 
367     // send mem address of zero
368     clear_pin(ht1632_data_port, ht1632_data_pin);
```

```
369
370     // write mem addr, bits 6 -> 0
371     for(count = 0 ; count < 7 ; count++) {
372
373         //change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(mem_addr, 6));
374         // pulse wr
375         clear_pin(ht1632_wr_port, ht1632_wr_pin);
376         set_pin   (ht1632_wr_port, ht1632_wr_pin);
377         // shift mem addr along
378
379     }
380     if (colour) {
381         set_pin(ht1632_data_port, ht1632_data_pin);
382     } else {
383         clear_pin(ht1632 data port, ht1632 data pin);
384     }
385     // we need to toggle 384 times
386
387     for(count = 0 ; count < 384 ; count++) {
388         // pulse wr
389         clear_pin(ht1632_wr_port, ht1632_wr_pin);
390         set_pin   (ht1632_wr_port, ht1632_wr_pin);
391         // shift mem addr along
392
393     }
394     // reset CS
395
396     set_pin(ht1632 cs1 port, ht1632 cs1 pin);
397     ht1632_send_command(HT1632_CMD_LEDS_ON);
398
399
400 }
```

Here is the call graph for this function:



void ht1632_init (uns8 *hw_config*)

```
75                                  {
76
77     ht1632_send_command(HT1632_CMD_SYS_DISABLE);
78 //  ht1632_send_command(HT1632_CMD_PMOS_16_COMMON); // Correct hardware layout for the board
79 //  ht1632_send_command(HT1632_CMD_PMOS_8_COMMON);  // Correct hardware layout for the board
80     ht1632_send_command(hw_config);
81     ht1632_send_command(HT1632_CMD_CLK_MASTER_MODE);    // We are the master
82     ht1632_send_command(HT1632_CMD_SYS_ENABLE);
83     ht1632_send_command(HT1632_CMD_LEDS_ON); //led on
84 }
```

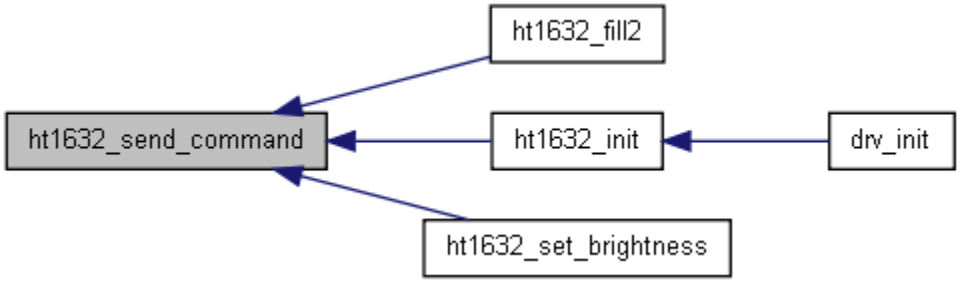
Here is the call graph for this function:

Here is the caller graph for this function:

void ht1632_send_command (uns8 *command*)

```
86                                          {
87
88 uns8 count;
89
90     clear pin(ht1632_cs1_port, ht1632_cs1_pin);
91     clear pin(ht1632_cs2_port, ht1632_cs2_pin);
92     clear pin(ht1632_cs3_port, ht1632_cs3_pin);
93     clear pin(ht1632_cs4_port, ht1632_cs4_pin);
94
95     // send  command
96     // send 1
97     set pin  (ht1632_data_port, ht1632_data_pin);
98     // pulse wr
99     clear pin(ht1632 wr port, ht1632 wr pin);
100     set pin  (ht1632_wr_port, ht1632_wr_pin);
101
102     // send 0
103     clear pin  (ht1632_data_port, ht1632_data_pin);
104     // pulse wr
105     clear pin(ht1632_wr_port, ht1632_wr_pin);
106     set pin  (ht1632_wr_port, ht1632_wr_pin);
107
108     // send 0
109     clear pin  (ht1632_data_port, ht1632_data_pin);
110     // pulse wr
111     clear pin(ht1632_wr_port, ht1632_wr_pin);
112     set pin  (ht1632_wr_port, ht1632_wr_pin);
113
114     // command bits 7 - 0
115     for(count = 0 ; count < 8 ; count++) {
116
117         if (test_bit(command, 7)) {
118             set pin(ht1632_data_port, ht1632_data_pin);
119         } else {
120             clear pin(ht1632_data_port, ht1632_data_pin);
121         }
122
123         // pulse wr
124         clear pin(ht1632_wr_port, ht1632_wr_pin);
125         set pin  (ht1632_wr_port, ht1632_wr_pin);
126         // shift mem addr along
127         command = command << 1;
128     }
129
130     // the don't care pulse
131
132         clear pin(ht1632_wr_port, ht1632_wr_pin);
133         set pin  (ht1632 wr port, ht1632 wr pin);
134
135
136     set pin(ht1632_cs1_port, ht1632_cs1_pin);
137     set pin(ht1632_cs2_port, ht1632_cs2_pin);
138     set pin(ht1632_cs3_port, ht1632_cs3_pin);
139     set pin(ht1632_cs4_port, ht1632_cs4_pin);
140 }
```

Here is the caller graph for this function:



void ht1632_set_brightness (uns8 *brightness*)

```
206                                                 {
207     // allows level 0 - 15
208     ht1632_send_command(0b10100000 | (brightness & 0b00001111));
209 }
```

Here is the call graph for this function:

void ht1632_set_pixel (uns8 *x*, uns8 *y*, uns8 *colour*)

```
211                                                 {
212
213 uns8 common, panel, led_in_panel, inverted_x, out, mem_addr, bit_in_mem_addr, count, data;
214
215     // first calculate memory address
216
217     // y location on panels is top left based
218
219     common = 15 - y;
220
221 /* Previous calculations:
222     panel = x / 8;  // which panel of the three is it that we need to change?
223     led_in_panel = x - (panel * 8);
224     inverted_x = 7 - led_in_panel;
225     out = panel * 8 + led_in_panel; //inverted_x;
226     mem_addr = out * 4 + common / 4;
227 */
228     bit_in_mem_addr = common & 0b00000011;
229
230
231     mem_addr = x * 4 + common / 4;
232
233
234     clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
235
236     // send WR command
237
238     // send 1
239     set_pin   (ht1632_data_port, ht1632_data_pin);
240     // pulse wr
241     clear_pin(ht1632_wr_port, ht1632_wr_pin);
242     set_pin   (ht1632_wr_port, ht1632_wr_pin);
243
244     // send 0
245     clear_pin   (ht1632_data_port, ht1632_data_pin);
246     // pulse wr
247     clear_pin(ht1632_wr_port, ht1632_wr_pin);
```

```
248     set pin   (ht1632_wr_port, ht1632_wr_pin);
249
250     // send 1
251     set pin   (ht1632_data_port, ht1632_data_pin);
252     // pulse wr
253     clear pin(ht1632_wr_port, ht1632_wr_pin);
254     set pin   (ht1632_wr_port, ht1632_wr_pin);
255
256     // write mem addr, bits 6 -> 0
257     for(count = 0 ; count < 7 ; count++) {
258         if (test bit(mem addr, 6)) {
259             set pin(ht1632_data_port, ht1632_data_pin);
260         } else {
261             clear pin(ht1632_data_port, ht1632_data_pin);
262         }
263
264         //change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(mem_addr, 6));
265         // pulse rd
266         clear pin(ht1632_wr_port, ht1632_wr_pin);
267         set pin   (ht1632_wr_port, ht1632_wr_pin);
268
269         // shift mem addr along
270         mem_addr = mem_addr << 1;
271     }
272
273     // Retrieve 4 bits
274     // read clocked out on falling edge of RD
275
276     make input(ht1632_data_port, ht1632_data_pin);
277
278     for(count = 0 ; count < 4 ; count++) {
279         // pulse rd
280         clear pin(ht1632_rd_port, ht1632_rd_pin);
281
282         data = data >> 1;
283         data.3 = test pin(ht1632_data_port, ht1632_data_pin);
284
285         set pin   (ht1632_rd_port, ht1632_rd_pin);
286
287     }
288
289     make output(ht1632_data_port, ht1632_data_pin);
290
291     // now we have the data, we need to change the bit
292     if (colour) {
293         set_bit(data, bit_in_mem_addr);
294     } else {
295         clear_bit(data, bit_in_mem_addr);
296     }
297
298     // Now write it back out again
299
300     // write data, bits 0 -> 3 (different from mem addr format)
301     for(count = 0 ; count < 4 ; count++) {
302         //change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(data, 0));
303         if (test_bit(data, 0)) {
304             set pin(ht1632_data_port, ht1632_data_pin);
305         } else {
306             clear pin(ht1632_data_port, ht1632_data_pin);
307         }
308
309         // pulse wr
310         clear pin(ht1632_wr_port, ht1632_wr_pin);
311         set pin   (ht1632_wr_port, ht1632_wr_pin);
312         // shift data along
313         data = data >> 1;
314     }
315
316
317     // reset CS
318     // don't think this is necessary
```

```
319 //  set_pin(ht1632_cs1_port, ht1632_cs1_pin);
320 //  clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
321     set_pin   (ht1632_cs1_port, ht1632_cs1_pin);
322
323
324 }
```

void ht1632_setup_io ()

```
41                          {
42
43     make_output(ht1632_cs1_port, ht1632_cs1_pin);
44
45     #if ht1632_displays > 1
46         make_output(ht1632_cs2_port, ht1632_cs2_pin);
47     #endif
48     #if ht1632_displays > 2
49         make_output(ht1632_cs3_port, ht1632_cs3_pin);
50     #endif
51     #if ht1632_displays > 3
52         make_output(ht1632_cs4_port, ht1632_cs4_pin);
53     #endif
54
55
56     make_output(ht1632_data_port, ht1632_data_pin);
57     make_output(ht1632_wr_port, ht1632_wr_pin);
58     make_output(ht1632_rd_port, ht1632_rd_pin);
59
60     set_pin(ht1632_wr_port, ht1632_wr_pin);
61     set_pin(ht1632_rd_port, ht1632_rd_pin);
62     set_pin(ht1632_cs1_port, ht1632_cs1_pin);
63     #if ht1632_displays > 1
64         set_pin(ht1632_cs2_port, ht1632_cs2_pin);
65     #endif
66     #if ht1632_displays > 2
67         set_pin(ht1632_cs3_port, ht1632_cs3_pin);
68     #endif
69     #if ht1632_displays > 3
70         set_pin(ht1632_cs4_port, ht1632_cs4_pin);
71     #endif
72
73 }
```

Here is the caller graph for this function:

void ht1632_write (uns8 *mem_addr*, uns8 *data*)

```
142                                             {
143
144 uns8 count;
145
146     clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
147
148     // send WR command
149
150     // send 1
151     set_pin   (ht1632_data_port, ht1632_data_pin);
152     // pulse wr
153     clear_pin(ht1632_wr_port, ht1632_wr_pin);
154     set_pin   (ht1632_wr_port, ht1632_wr_pin);
155
```

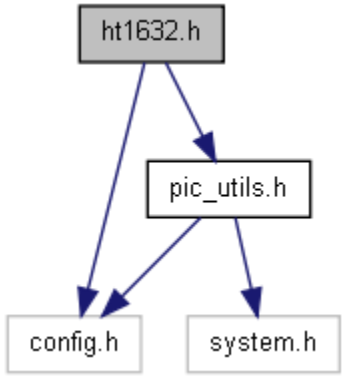
```
156     // send 0
157     clear pin  (ht1632_data_port, ht1632_data_pin);
158     // pulse wr
159     clear pin(ht1632_wr_port, ht1632_wr_pin);
160     set pin  (ht1632_wr_port, ht1632_wr_pin);
161
162     // send 1
163     set pin  (ht1632_data_port, ht1632_data_pin);
164     // pulse wr
165     clear pin(ht1632_wr_port, ht1632_wr_pin);
166     set pin  (ht1632_wr_port, ht1632_wr_pin);
167
168     // write mem addr, bits 6 -> 0
169     for(count = 0 ; count < 7 ; count++) {
170         if (test bit(mem addr, 6)) {
171             set pin(ht1632_data_port, ht1632_data_pin);
172         } else {
173             clear pin(ht1632_data_port, ht1632_data_pin);
174         }
175
176         //change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(mem_addr, 6));
177         // pulse wr
178         clear pin(ht1632_wr_port, ht1632_wr_pin);
179         set pin  (ht1632_wr_port, ht1632_wr_pin);
180         // shift mem addr along
181         mem_addr = mem_addr << 1;
182     }
183
184     // write data, bits 0 -> 3 (different from mem addr format)
185     for(count = 0 ; count < 4 ; count++) {
186         change pin var(ht1632_data_port, ht1632_data_pin, test_bit(data, 0));
187         // pulse wr
188         clear pin(ht1632_wr_port, ht1632_wr_pin);
189         set pin  (ht1632_wr_port, ht1632_wr_pin);
190         // shift mem addr along
191         data = data >> 1;
192     }
193     // reset CS
194
195 //  set pin(ht1632_cs1_port, ht1632_cs1_pin);
196 //  delay_ms(1);
197 //  clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
198     set pin(ht1632_cs1_port, ht1632_cs1_pin);
199
200
201 }
```

Here is the caller graph for this function:

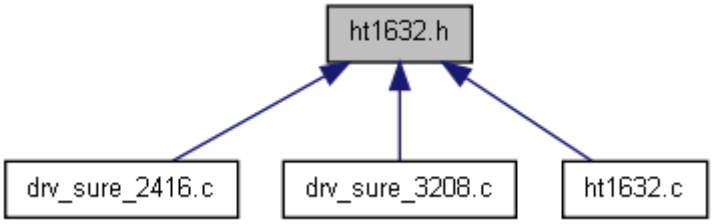
---

ht1632.h File Reference

Holtek LED matrix display routines, used in Sure 2416 display and others.
Include dependency graph for ht1632.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define HT1632_CMD_BLINK_OFF 0b00001000
- #define HT1632_CMD_BLINK_ON 0b00001001
- #define HT1632_CMD_CLK_MASTER_MODE 0b00010100
- #define HT1632_CMD_CLK_SLAVE_MODE 0b00010000
- #define HT1632_CMD_CLK_SOURCE_EXT 0b00011100
- #define HT1632_CMD_CLK_SOURCE_INT_RC 0b00011000
- #define HT1632_CMD_LEDS_OFF 0b00000010
- #define HT1632_CMD_LEDS_ON 0b00000011
- #define HT1632_CMD_NMOS_16_COMMON 0b00100100
- #define HT1632_CMD_NMOS_8_COMMON 0b00100000
- #define HT1632_CMD_PMOS_16_COMMON 0b00101100
- #define HT1632_CMD_PMOS_8_COMMON 0b00101000
- #define HT1632_CMD_SYS_DISABLE 0b00000000
- #define HT1632_CMD_SYS_ENABLE 0b00000001

Functions

- void ht1632_clear ()
- void ht1632_fill (uns8 colour)
- void ht1632_fill2 (uns8 colour)
- uns8 ht1632_get_pixel (uns8 x, uns8 y)
- void ht1632_horizontal_line (uns8 x, uns8 y, uns8 length, uns8 colour)
- void ht1632_init (uns8 hw_config)
- void ht1632_send_command (uns8 command)
- void ht1632_set_brightness (uns8 brightness)
- void ht1632_set_pixel (uns8 x, uns8 y, uns8 colour)
- void ht1632_setup_io ()
- void ht1632_vertical_line (uns8 x, uns8 y, uns8 length, uns8 colour)

- void ht1632_write (uns8 mem_addr, uns8 data)

Detailed Description

Routines to communicate with Holtek HT1632 led matrix display chip

---

Define Documentation

#define HT1632_CMD_BLINK_OFF 0b00001000

#define HT1632_CMD_BLINK_ON 0b00001001

#define HT1632_CMD_CLK_MASTER_MODE 0b00010100

#define HT1632_CMD_CLK_SLAVE_MODE 0b00010000

#define HT1632_CMD_CLK_SOURCE_EXT 0b00011100

#define HT1632_CMD_CLK_SOURCE_INT_RC 0b00011000

#define HT1632_CMD_LEDS_OFF 0b00000010

#define HT1632_CMD_LEDS_ON 0b00000011

#define HT1632_CMD_NMOS_16_COMMON 0b00100100

#define HT1632_CMD_NMOS_8_COMMON 0b00100000

#define HT1632_CMD_PMOS_16_COMMON 0b00101100

#define HT1632_CMD_PMOS_8_COMMON 0b00101000

#define HT1632_CMD_SYS_DISABLE 0b00000000

#define HT1632_CMD_SYS_ENABLE 0b00000001

Function Documentation

void ht1632_clear ()

void ht1632_fill (uns8 *colour*)

```
326                                 {
327     uns8 mem_address;
328     uns8 fill;
329 
330     if (colour) {
331         fill = 0b00001111;
332     } else {
333         fill = 0b00000000;
334     }
335 
336     for(mem_address = 0 ; mem_address < 96 ; mem_address++) {
```

```
337         ht1632_write(mem_address, fill);
338     }
339 }
```

Here is the call graph for this function:

void ht1632_fill2 (uns8 *colour*)

```
341                                 {
342 
343 uns16 count;
344     ht1632_send_command(HT1632_CMD_LEDS_OFF);
345     clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
346 
347     // send WR command
348 
349     // send 1
350     set_pin   (ht1632_data_port, ht1632_data_pin);
351     // pulse wr
352     clear_pin(ht1632_wr_port, ht1632_wr_pin);
353     set_pin   (ht1632_wr_port, ht1632_wr_pin);
354 
355     // send 0
356     clear_pin   (ht1632_data_port, ht1632_data_pin);
357     // pulse wr
358     clear_pin(ht1632_wr_port, ht1632_wr_pin);
359     set_pin   (ht1632_wr_port, ht1632_wr_pin);
360 
361     // send 1
362     set_pin   (ht1632_data_port, ht1632_data_pin);
363     // pulse wr
364     clear_pin(ht1632_wr_port, ht1632_wr_pin);
365     set_pin   (ht1632_wr_port, ht1632_wr_pin);
366 
367     // send mem address of zero
368     clear_pin(ht1632_data_port, ht1632_data_pin);
369 
370     // write mem addr, bits 6 -> 0
371     for(count = 0 ; count < 7 ; count++) {
372 
373         //change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(mem_addr, 6));
374         // pulse wr
375         clear_pin(ht1632_wr_port, ht1632_wr_pin);
376         set_pin   (ht1632_wr_port, ht1632_wr_pin);
377         // shift mem addr along
378 
379     }
380     if (colour) {
381         set_pin(ht1632_data_port, ht1632_data_pin);
382     } else {
383         clear_pin(ht1632_data_port, ht1632_data_pin);
384     }
385     // we need to toggle 384 times
386 
387     for(count = 0 ; count < 384 ; count++) {
388         // pulse wr
389         clear_pin(ht1632_wr_port, ht1632_wr_pin);
390         set_pin   (ht1632_wr_port, ht1632_wr_pin);
391         // shift mem addr along
392 
393     }
394     // reset CS
395 
396     set_pin(ht1632_cs1_port, ht1632_cs1_pin);
```

```
397     ht1632_send_command(HT1632_CMD_LEDS_ON);
398
399
400 }
```

Here is the call graph for this function:

uns8 ht1632_get_pixel (uns8 *x*, uns8 *y*)

void ht1632_horizontal_line (uns8 *x*, uns8 *y*, uns8 *length*, uns8 *colour*)

void ht1632_init (uns8 *hw_config*)

```
75                                  {
76
77     ht1632_send_command(HT1632_CMD_SYS_DISABLE);
78 //  ht1632_send_command(HT1632_CMD_PMOS_16_COMMON); // Correct hardware layout for the board
79 //  ht1632_send_command(HT1632_CMD_PMOS_8_COMMON);  // Correct hardware layout for the board
80     ht1632_send_command(hw_config);
81     ht1632_send_command(HT1632_CMD_CLK_MASTER_MODE);    // We are the master
82     ht1632_send_command(HT1632_CMD_SYS_ENABLE);
83     ht1632_send_command(HT1632_CMD_LEDS_ON); //led on
84 }
```

Here is the call graph for this function:

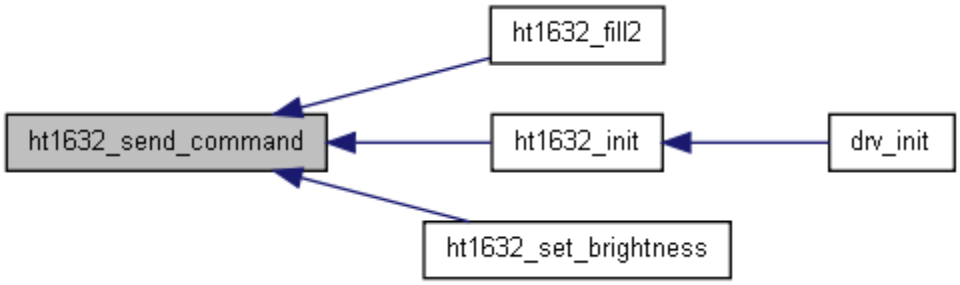
Here is the caller graph for this function:

void ht1632_send_command (uns8 *command*)

```
86                                      {
87
88 uns8 count;
89
90     clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
91     clear_pin(ht1632_cs2_port, ht1632_cs2_pin);
92     clear_pin(ht1632_cs3_port, ht1632_cs3_pin);
93     clear_pin(ht1632_cs4_port, ht1632_cs4_pin);
94
95     // send  command
96     // send 1
97     set_pin   (ht1632_data_port, ht1632_data_pin);
98     // pulse wr
99     clear_pin(ht1632_wr_port, ht1632_wr_pin);
100     set_pin   (ht1632_wr_port, ht1632_wr_pin);
101
102     // send 0
103     clear_pin   (ht1632_data_port, ht1632_data_pin);
104     // pulse wr
105     clear_pin(ht1632_wr_port, ht1632_wr_pin);
106     set_pin   (ht1632_wr_port, ht1632_wr_pin);
```

```
107
108     // send 0
109     clear pin  (ht1632 data port, ht1632 data pin);
110     // pulse wr
111     clear pin(ht1632_wr_port, ht1632_wr_pin);
112     set pin  (ht1632_wr_port, ht1632_wr_pin);
113
114     // command bits 7 - 0
115     for(count = 0 ; count < 8 ; count++) {
116
117         if (test bit(command, 7)) {
118             set pin(ht1632_data_port, ht1632_data_pin);
119         } else {
120             clear pin(ht1632_data_port, ht1632_data_pin);
121         }
122
123         // pulse wr
124         clear pin(ht1632_wr_port, ht1632_wr_pin);
125         set pin  (ht1632_wr_port, ht1632_wr_pin);
126         // shift mem addr along
127         command = command << 1;
128     }
129
130     // the don't care pulse
131
132         clear pin(ht1632_wr_port, ht1632_wr_pin);
133         set pin  (ht1632_wr_port, ht1632_wr_pin);
134
135
136     set pin(ht1632_cs1_port, ht1632_cs1_pin);
137     set pin(ht1632_cs2_port, ht1632_cs2_pin);
138     set pin(ht1632 cs3 port, ht1632 cs3 pin);
139     set pin(ht1632_cs4_port, ht1632_cs4_pin);
140 }
```

Here is the caller graph for this function:



void ht1632_set_brightness (uns8 *brightness*)

```
206                                             {
207     // allows level 0 - 15
208     ht1632 send command(0b10100000 | (brightness & 0b00001111));
209 }
```

Here is the call graph for this function:

void ht1632_set_pixel (uns8 *x*, uns8 *y*, uns8 *colour*)

```
211                                                 {
212
213 uns8 common, panel, led in panel, inverted x, out, mem addr, bit in mem addr, count, data;
214
215     // first calculate memory address
216
217     // y location on panels is top left based
218
219     common = 15 - y;
220
221 /* Previous calculations:
222     panel = x / 8;  // which panel of the three is it that we need to change?
223     led_in_panel = x - (panel * 8);
224     inverted_x = 7 - led_in_panel;
225     out = panel * 8 + led in panel; //inverted x;
226     mem_addr = out * 4 + common / 4;
227 */
228     bit_in_mem_addr = common & 0b00000011;
229
230
231     mem_addr = x * 4 + common / 4;
232
233
234     clear pin(ht1632_cs1_port, ht1632_cs1_pin);
235
236     // send WR command
237
238     // send 1
239     set pin   (ht1632_data_port, ht1632_data_pin);
240     // pulse wr
241     clear pin(ht1632_wr_port, ht1632_wr_pin);
242     set pin   (ht1632_wr_port, ht1632_wr_pin);
243
244     // send 0
245     clear pin   (ht1632_data_port, ht1632_data_pin);
246     // pulse wr
247     clear pin(ht1632_wr_port, ht1632_wr_pin);
248     set pin   (ht1632_wr_port, ht1632_wr_pin);
249
250     // send 1
251     set pin   (ht1632_data_port, ht1632_data_pin);
252     // pulse wr
253     clear pin(ht1632_wr_port, ht1632_wr_pin);
254     set pin   (ht1632_wr_port, ht1632_wr_pin);
255
256     // write mem addr, bits 6 -> 0
257     for(count = 0 ; count < 7 ; count++) {
258         if (test_bit(mem_addr, 6)) {
259             set pin(ht1632_data_port, ht1632_data_pin);
260         } else {
261             clear pin(ht1632_data_port, ht1632_data_pin);
262         }
263
264         //change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(mem_addr, 6));
265         // pulse rd
266         clear pin(ht1632_wr_port, ht1632_wr_pin);
267         set pin   (ht1632_wr_port, ht1632_wr_pin);
268
269         // shift mem addr along
270         mem_addr = mem_addr << 1;
271     }
272
273     // Retrieve 4 bits
274     // read clocked out on falling edge of RD
275
276     make input(ht1632_data_port, ht1632_data_pin);
277
278     for(count = 0 ; count < 4 ; count++) {
279         // pulse rd
280         clear pin(ht1632_rd_port, ht1632_rd_pin);
281
```

```
282         data = data >> 1;
283         data.3 = test pin(ht1632_data_port, ht1632_data_pin);
284 
285         set pin   (ht1632_rd_port, ht1632_rd_pin);
286 
287     }
288 
289     make output(ht1632_data_port, ht1632_data_pin);
290 
291     // now we have the data, we need to change the bit
292     if (colour) {
293         set_bit(data, bit_in_mem_addr);
294     } else {
295         clear_bit(data, bit_in_mem_addr);
296     }
297 
298     // Now write it back out again
299 
300     // write data, bits 0 -> 3 (different from mem addr format)
301     for(count = 0 ; count < 4 ; count++) {
302         //change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(data, 0));
303         if (test_bit(data, 0)) {
304             set pin(ht1632_data_port, ht1632_data_pin);
305         } else {
306             clear pin(ht1632_data_port, ht1632_data_pin);
307         }
308 
309         // pulse wr
310         clear pin(ht1632_wr_port, ht1632_wr_pin);
311         set pin   (ht1632_wr_port, ht1632_wr_pin);
312         // shift data along
313         data = data >> 1;
314     }
315 
316 
317     // reset CS
318     // don't think this is necessary
319 //  set_pin(ht1632_cs1_port, ht1632_cs1_pin);
320 //  clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
321     set pin   (ht1632_cs1_port, ht1632_cs1_pin);
322 
323 
324 }
```

void ht1632_setup_io ()

```
41                          {
42 
43     make output(ht1632_cs1_port, ht1632_cs1_pin);
44 
45     #if ht1632_displays > 1
46         make output(ht1632_cs2_port, ht1632_cs2_pin);
47     #endif
48     #if ht1632_displays > 2
49         make output(ht1632_cs3_port, ht1632_cs3_pin);
50     #endif
51     #if ht1632_displays > 3
52         make output(ht1632_cs4_port, ht1632_cs4_pin);
53     #endif
54 
55 
56     make output(ht1632_data_port, ht1632_data_pin);
57     make output(ht1632_wr_port, ht1632_wr_pin);
58     make output(ht1632_rd_port, ht1632_rd_pin);
59 
60     set pin(ht1632_wr_port, ht1632_wr_pin);
61     set pin(ht1632_rd_port, ht1632_rd_pin);
62     set pin(ht1632_cs1_port, ht1632_cs1_pin);
```

```
63      #if ht1632_displays > 1
64          set_pin(ht1632_cs2_port, ht1632_cs2_pin);
65      #endif
66      #if ht1632_displays > 2
67          set_pin(ht1632_cs3_port, ht1632_cs3_pin);
68      #endif
69      #if ht1632_displays > 3
70          set_pin(ht1632_cs4_port, ht1632_cs4_pin);
71      #endif
72
73  }
```

Here is the caller graph for this function:

void ht1632_vertical_line (uns8 *x*, uns8 *y*, uns8 *length*, uns8 *colour*)

void ht1632_write (uns8 *mem_addr*, uns8 *data*)

```
142                                             {
143
144 uns8 count;
145
146     clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
147
148     // send WR command
149
150     // send 1
151     set_pin   (ht1632_data_port, ht1632_data_pin);
152     // pulse wr
153     clear_pin(ht1632_wr_port, ht1632_wr_pin);
154     set_pin   (ht1632_wr_port, ht1632_wr_pin);
155
156     // send 0
157     clear_pin   (ht1632_data_port, ht1632_data_pin);
158     // pulse wr
159     clear_pin(ht1632_wr_port, ht1632_wr_pin);
160     set_pin   (ht1632_wr_port, ht1632_wr_pin);
161
162     // send 1
163     set_pin   (ht1632_data_port, ht1632_data_pin);
164     // pulse wr
165     clear_pin(ht1632_wr_port, ht1632_wr_pin);
166     set_pin   (ht1632_wr_port, ht1632_wr_pin);
167
168     // write mem addr, bits 6 -> 0
169     for(count = 0 ; count < 7 ; count++) {
170         if (test_bit(mem_addr, 6)) {
171             set_pin(ht1632_data_port, ht1632_data_pin);
172         } else {
173             clear_pin(ht1632_data_port, ht1632_data_pin);
174         }
175
176         //change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(mem_addr, 6));
177         // pulse wr
178         clear_pin(ht1632_wr_port, ht1632_wr_pin);
179         set_pin   (ht1632_wr_port, ht1632_wr_pin);
180         // shift mem addr along
181         mem_addr = mem_addr << 1;
182     }
183
184     // write data, bits 0 -> 3 (different from mem addr format)
185     for(count = 0 ; count < 4 ; count++) {
186         change_pin_var(ht1632_data_port, ht1632_data_pin, test_bit(data, 0));
```

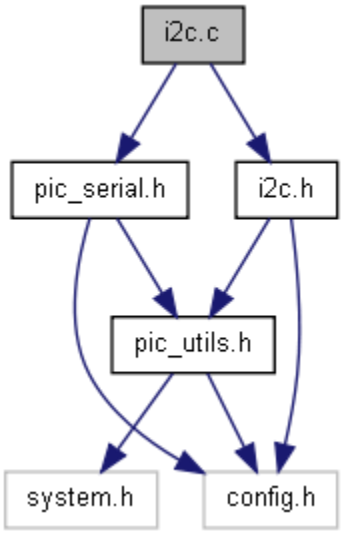
```
187         // pulse wr
188         clear_pin(ht1632_wr_port, ht1632_wr_pin);
189         set_pin  (ht1632_wr_port, ht1632_wr_pin);
190         // shift mem addr along
191         data = data >> 1;
192     }
193     // reset CS
194
195 //  set_pin(ht1632_cs1_port, ht1632_cs1_pin);
196 //  delay_ms(1);
197 //  clear_pin(ht1632_cs1_port, ht1632_cs1_pin);
198     set_pin(ht1632_cs1_port, ht1632_cs1_pin);
199
200
201 }
```

Here is the caller graph for this function:

---

i2c.c File Reference

Include dependency graph for i2c.c:



Defines

- #define DELAY_AMOUNT 100

Functions

- void i2c_ack_polling (uns8 device_address)
- uns8 i2c_read_eeprom (uns8 device_address, uns8 mem_address)
- *Read an 8 bit byte over I2C buss.* uns16 i2c_read_eeprom_16bit (uns8 device_address, uns8 mem_address)
- *Read 16 bits of data over I2C buss.* uns8 i2c_receive_byte (void)
- *Receive byte from I2C buss.* void i2c_send_ack (void)

- *Send an ACK.* void i2c_send_byte (uns8 data)
- *Send a byte to I2C buss.* void i2c_setup_io ()
- *Setup ports and pins for I2C communication.* void i2c_start (void)
- *Send start signal to I2C buss.* void i2c_stop (void)
- *Send stop signal to I2C buss.* void i2c_write_eeprom (uns8 device_address, uns8 mem_address, uns8 data)
- *Write an 8 bit byte ove I2C buss.* void i2c_write_eeprom_16bit (uns8 device_address, uns8 mem_address, uns16 data)

Write a 16 bit value over I2C buss.

Define Documentation

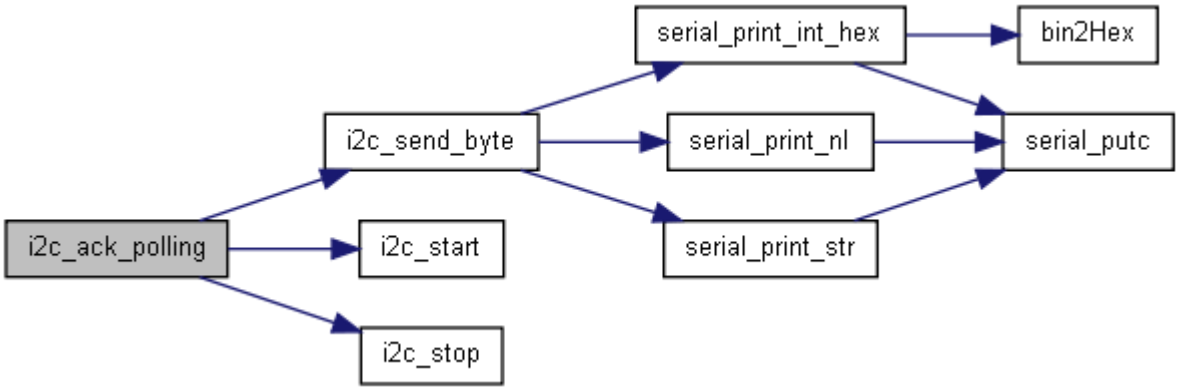
#define DELAY_AMOUNT 100

Function Documentation

void i2c_ack_polling (uns8 *device_address*)

```
42 {
43     while(test_pin(i2c_sda_port, i2c_sda_pin)!= 0)
44     {
45         i2c_start();
46         i2c_send_byte(device_address);
47     }
48     i2c_stop();
49 }
```

Here is the call graph for this function:



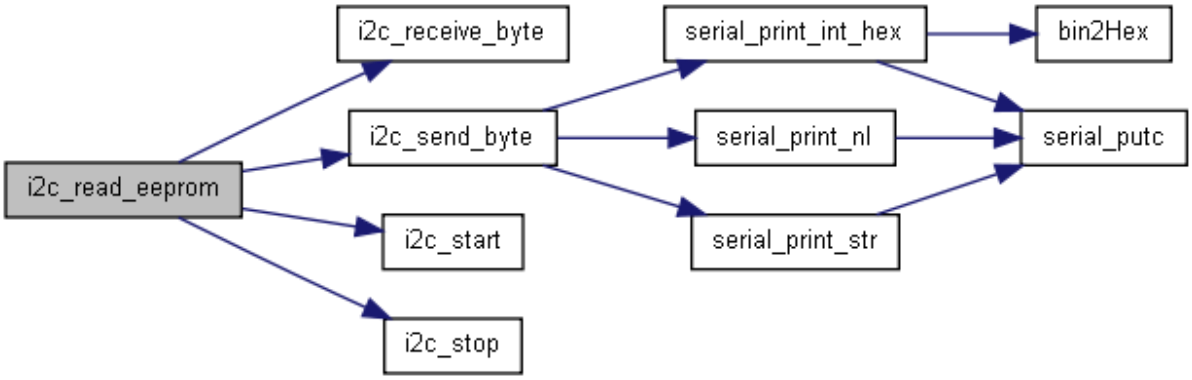
uns8 i2c_read_eeprom (uns8 *device_address*, uns8 *mem_address*)

Read an 8 bit byte from the specified device at the memory address

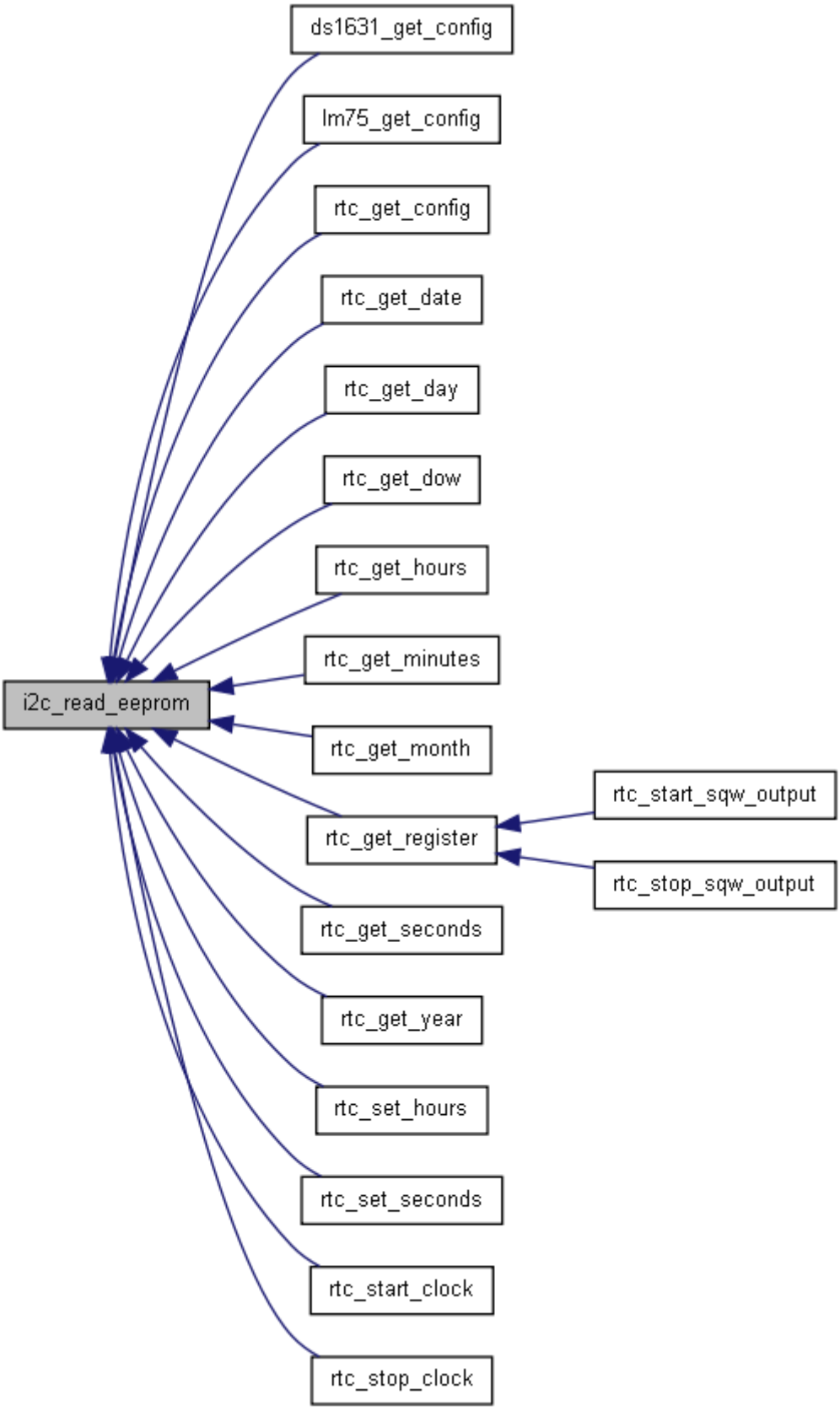
```
77 {
78 uns8 data;
79     //i2c_ack_polling(device_address);
80 
81     i2c_start();
82     i2c_send_byte(device_address);
83     //send_byte(address.high8); // Upper Address - Needed for >= 32k EEProms
84     i2c_send_byte(mem_address);
85     i2c_stop();
86 
87     i2c_start();
88     i2c_send_byte(device_address | 0b00000001); // Read bit must be set
```

```
89      data = i2c receive byte(); // reuse local variable
90
91      // write an nack
92      clear pin(i2c_scl_port, i2c_scl_pin);
93      i2c write sda();
94      delay_us(DELAY AMOUNT);
95      set pin(i2c_sda_port, i2c_sda_pin);
96      set pin(i2c_scl_port, i2c_scl_pin);
97      delay_us(DELAY AMOUNT);
98
99      i2c stop();
100      return(data);
101 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

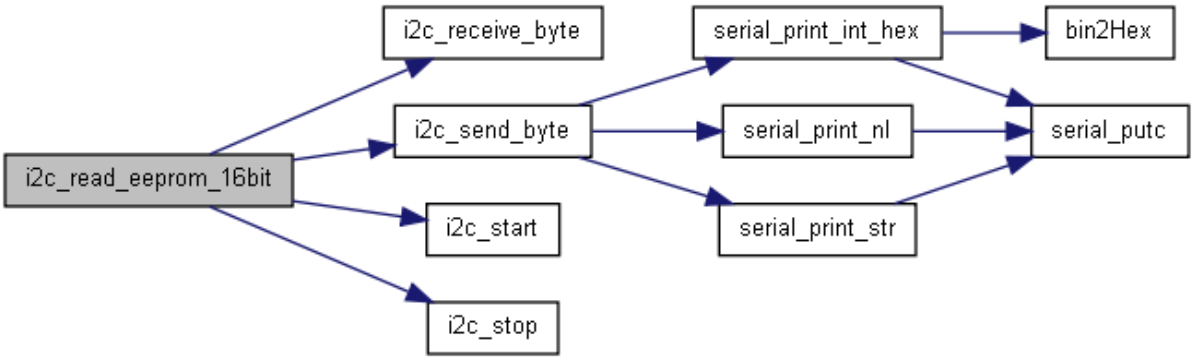

ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
rtc_get_dow
rtc_get_hours
rtc_get_minutes
i2c_read_eeprom
rtc_get_month
rtc_start_sqw_output
rtc_get_register
rtc_stop_sqw_output
rtc_get_seconds
rtc_get_year
rtc_set_hours
rtc_set_seconds
rtc_start_clock
rtc_stop_clock

uns16 i2c_read_eeprom_16bit (uns8 *device_address*, uns8 *mem_address*)

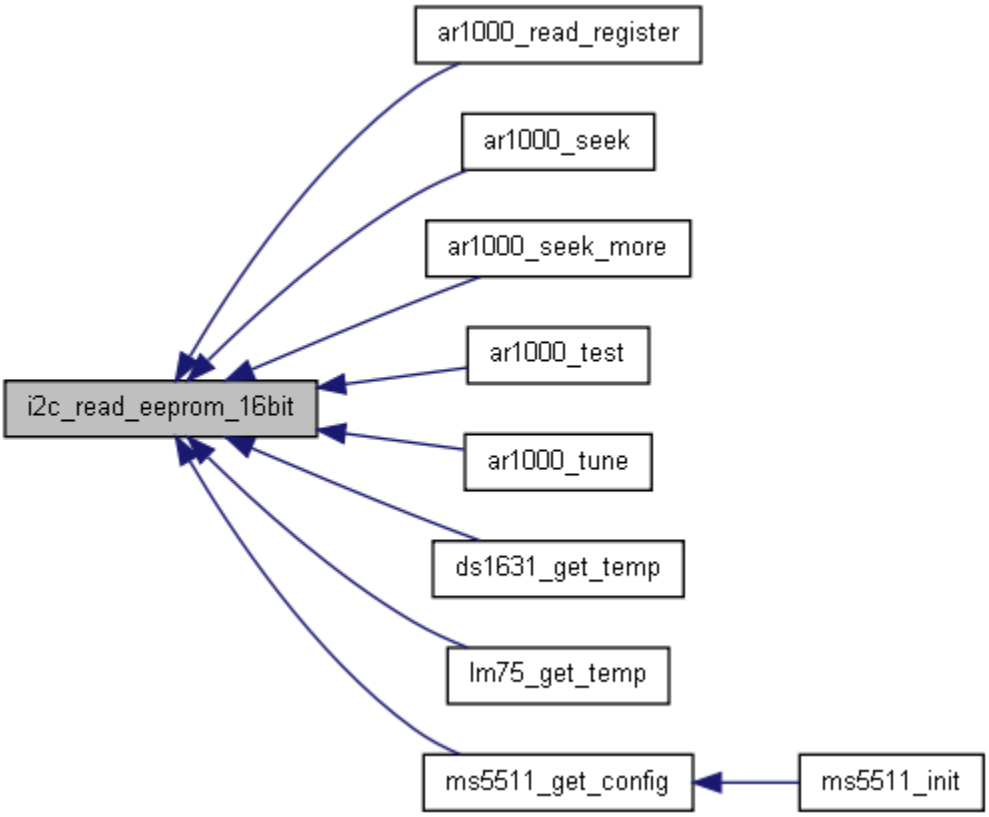
Read a 16 bit chunk of data from the given device and memory address

```
104 {
105     uns16 data;
106 
107     //i2c_ack_polling(device_address);
108     i2c_start();
109     i2c_send_byte(device_address);
110     //send_byte(address.high8); // Upper Address - Needed for >= 32k EEProms
111     i2c_send_byte(mem_address);
112 
113     i2c_start();
114     i2c_send_byte(device_address | 0b00000001); // Read bit must be set
115     data = i2c_receive_byte(); // reuse local variable
116     i2c_write_sda();
117 
118     clear_pin(i2c_scl_port, i2c_scl_pin);
119     delay_us(DELAY_AMOUNT);
120     clear_pin(i2c_sda_port, i2c_sda_pin); //ack
121     delay_us(DELAY_AMOUNT);
122     set_pin(i2c_scl_port, i2c_scl_pin);
123     delay_us(DELAY_AMOUNT);
124 
125     data = data << 8 | i2c_receive_byte(); // reuse local variable
126 
127     // write an nack
128     clear_pin(i2c_scl_port, i2c_scl_pin);
129     i2c_write_sda();
130     delay_us(DELAY_AMOUNT);
131     set_pin(i2c_sda_port, i2c_sda_pin);
132     set_pin(i2c_scl_port, i2c_scl_pin);
133     delay_us(DELAY_AMOUNT);
134 
135     i2c_stop();
136 
137     return(data);
138 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

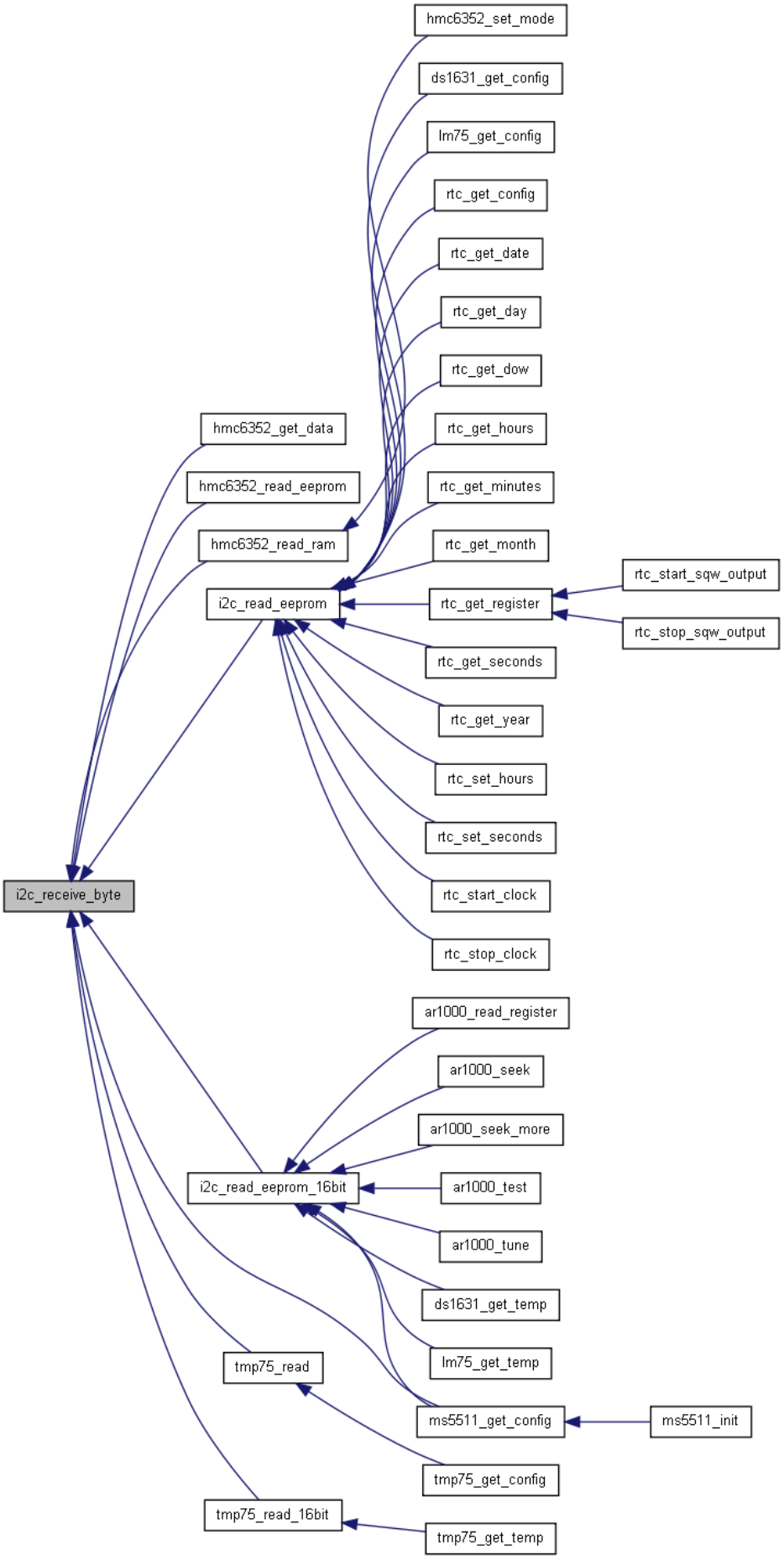


uns8 i2c_receive_byte ()

Clock in a byte over I2C lines. Note that an acknowledge is not sent/received

```
187 {
188    uns8 count, in_byte;
189 
190     clear_pin(i2c_scl_port, i2c_scl_pin);
191 
192     i2c_read_sda();
193     for(count = 0 ; count < 8 ; count++)
194     {
195         clear_pin(i2c_scl_port, i2c_scl_pin);
196         delay_us(DELAY_AMOUNT);
197         set_pin(i2c_scl_port, i2c_scl_pin);
198         delay_us(DELAY_AMOUNT);
199 
200         in_byte = in_byte << 1;
201         in_byte.0 = test_pin(i2c_sda_port, i2c_sda_pin);
202     }
203     return(in_byte);
204 }
```

Here is the caller graph for this function:


hmc6352_set_mode
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
rtc_get_dow
rtc_get_hours
rtc_get_minutes
rtc_get_month
rtc_get_register
rtc_start_sqw_output
rtc_stop_sqw_output
rtc_get_seconds
rtc_get_year
rtc_set_hours
rtc_set_seconds
rtc_start_clock
rtc_stop_clock
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_read_ram
i2c_read_eeprom
i2c_receive_byte
ar1000_read_register
ar1000_seek
ar1000_seek_more
ar1000_test
ar1000_tune
i2c_read_eeprom_16bit
ds1631_get_temp
lm75_get_temp
ms5511_get_config
ms5511_init
tmp75_read
tmp75_get_config
tmp75_read_16bit
tmp75_get_temp

void i2c_send_ack (void)

Sends an I2C acknowledge (bit)

```
140                         {
141
142     i2c_write_sda();
143
144     clear_pin(i2c_scl_port, i2c_scl_pin);
145     delay_us(DELAY_AMOUNT);
146     clear_pin(i2c_sda_port, i2c_sda_pin); //ack
147     delay_us(DELAY_AMOUNT);
148     set_pin(i2c_scl_port, i2c_scl_pin);
149     delay_us(DELAY_AMOUNT);
150
151
152 }
```

Here is the caller graph for this function:

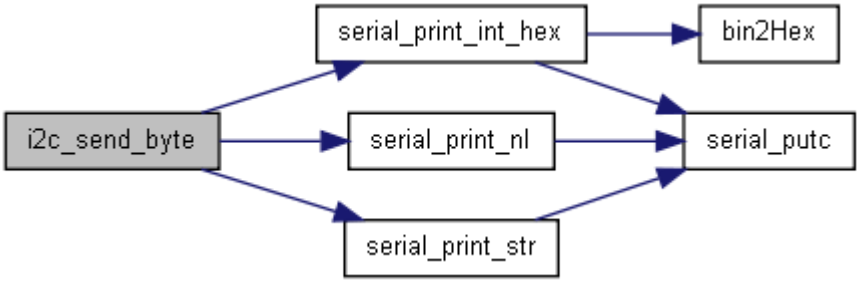
void i2c_send_byte (uns8 *data*)

Clock out a byte from the I2C buss. An acknowledge is "received" (although ignored).

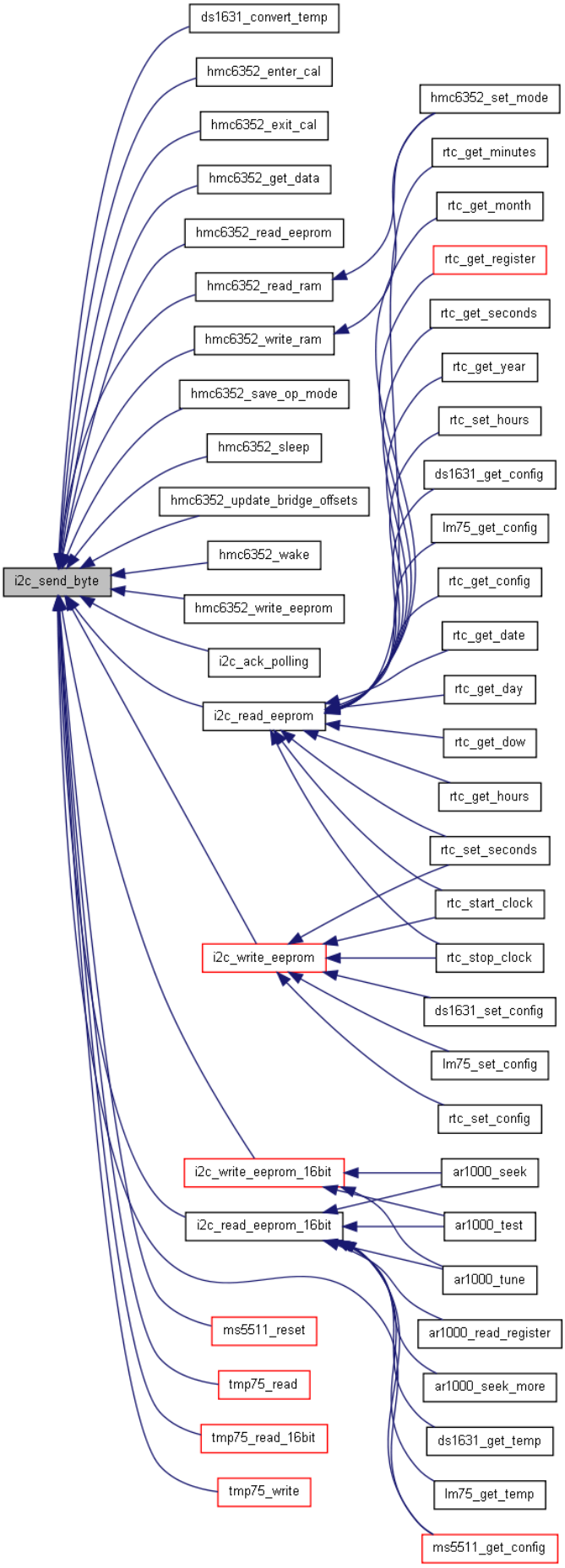
```
207 {
208    uns8 count;
209 serial_print_str("Sending: ");
210 serial_print_int_hex(data);
211 serial_print_nl();
212     clear_pin(i2c_scl_port, i2c_scl_pin);
213     i2c_write_sda();
214
215     for( count = 0 ; count < 8 ; count++ )
216     {
217         clear_pin(i2c_scl_port, i2c_scl_pin);
218         change_pin(i2c_sda_port, i2c_sda_pin, data.7);
219         data = data << 1;
220         delay_us(DELAY_AMOUNT);
221         set_pin(i2c_scl_port, i2c_scl_pin);
222         delay_us(DELAY_AMOUNT);
223     }
224
225     // read ack (and ignore)
226     clear_pin(i2c_scl_port, i2c_scl_pin);
227     i2c_read_sda();
228     delay_us(DELAY_AMOUNT);
229     set_pin(i2c_scl_port, i2c_scl_pin);
230     // read at this point
231
232     delay_us(DELAY_AMOUNT);
```

```
233 }
```

Here is the call graph for this function:



Here is the caller graph for this function:


ds1631_convert_temp
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_read_ram
hmc6352_write_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_update_bridge_offsets
hmc6352_wake
hmc6352_write_eeprom
i2c_ack_polling
i2c_send_byte
i2c_read_eeprom
i2c_write_eeprom
i2c_write_eeprom_16bit
i2c_read_eeprom_16bit
ms5511_reset
tmp75_read
tmp75_read_16bit
tmp75_write
hmc6352_set_mode
rtc_get_minutes
rtc_get_month
rtc_get_register
rtc_get_seconds
rtc_get_year
rtc_set_hours
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
rtc_get_dow
rtc_get_hours
rtc_set_seconds
rtc_start_clock
rtc_stop_clock
ds1631_set_config
lm75_set_config
rtc_set_config
ar1000_seek
ar1000_test
ar1000_tune
ar1000_read_register
ar1000_seek_more
ds1631_get_temp
lm75_get_temp
ms5511_get_config

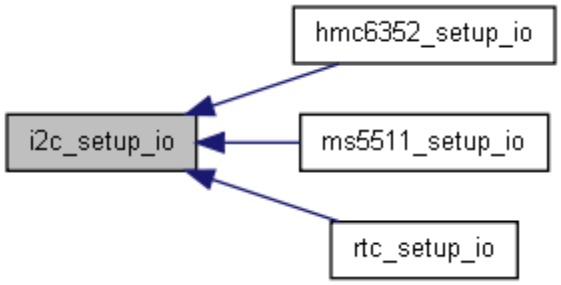
void i2c_setup_io ()

Setup ports and pins for i2c output.

Set port and pins correctly for I2C communication

```
235                     {
236     make_output(i2c_scl_port, i2c_scl_pin);
237     make_input(i2c_sda_port, i2c_sda_pin);
238 }
```

Here is the caller graph for this function:

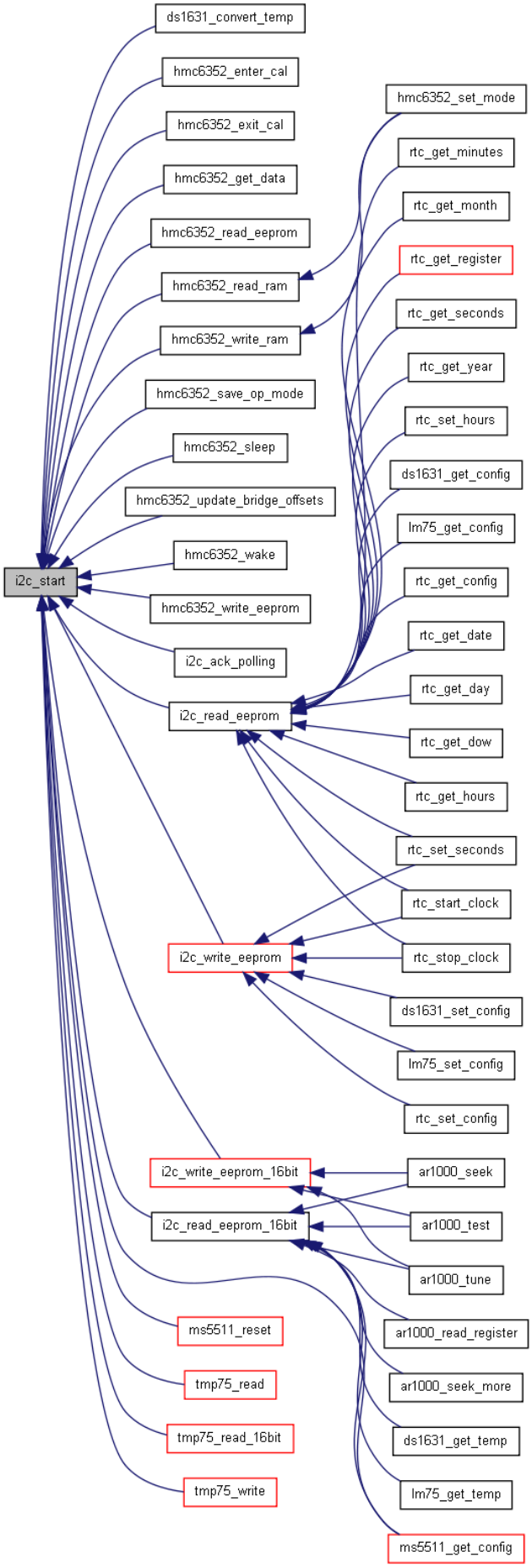


void i2c_start (void)

Signals a start condition on the I2C buss.

```
156 {
157
158     clear_pin(i2c_scl_port, i2c_scl_pin);
159     delay_us(DELAY_AMOUNT);
160
161     i2c_write_sda();
162
163     set_pin(i2c_sda_port, i2c_sda_pin);
164     delay_us(DELAY_AMOUNT);
165     set_pin(i2c_scl_port, i2c_scl_pin);
166     delay_us(DELAY_AMOUNT);
167     clear_pin(i2c_sda_port, i2c_sda_pin);
168     delay_us(DELAY_AMOUNT);
169 }
```

Here is the caller graph for this function:

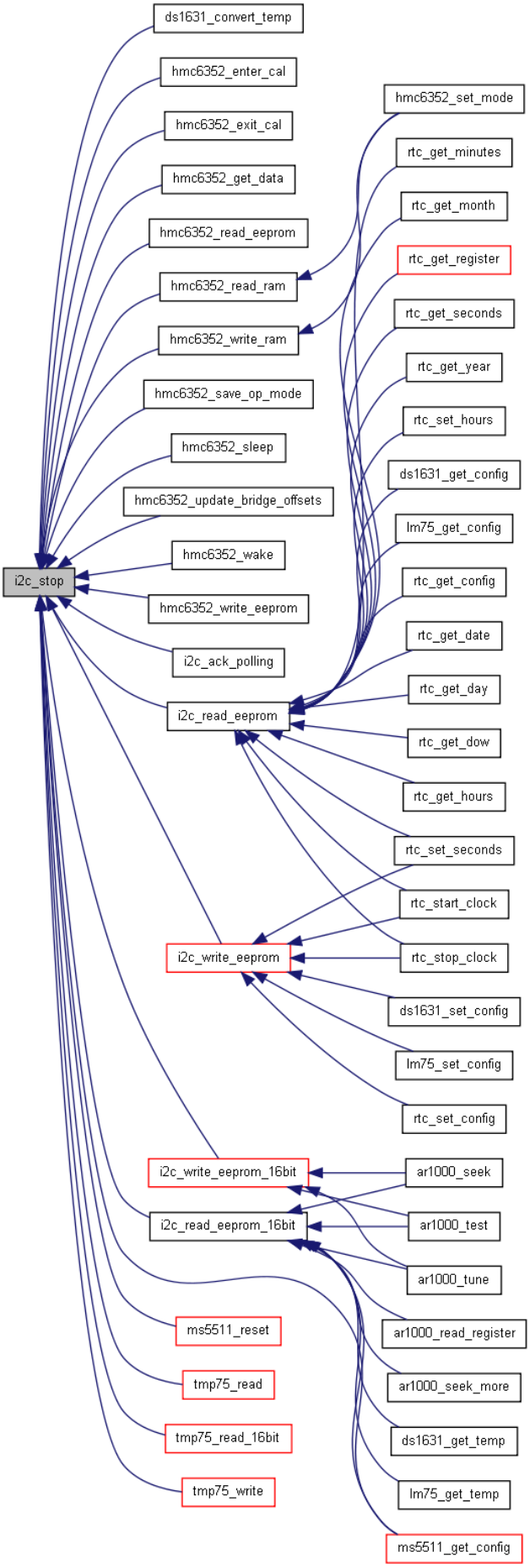

ds1631_convert_temp
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_read_ram
hmc6352_write_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_update_bridge_offsets
hmc6352_wake
hmc6352_write_eeprom
i2c_ack_polling
i2c_start
i2c_read_eeprom
i2c_write_eeprom
i2c_write_eeprom_16bit
i2c_read_eeprom_16bit
ms5511_reset
tmp75_read
tmp75_read_16bit
tmp75_write
hmc6352_set_mode
rtc_get_minutes
rtc_get_month
rtc_get_register
rtc_get_seconds
rtc_get_year
rtc_set_hours
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
rtc_get_dow
rtc_get_hours
rtc_set_seconds
rtc_start_clock
rtc_stop_clock
ds1631_set_config
lm75_set_config
rtc_set_config
ar1000_seek
ar1000_test
ar1000_tune
ar1000_read_register
ar1000_seek_more
ds1631_get_temp
lm75_get_temp
ms5511_get_config

void i2c_stop (void)

Signals a stop condition on the I2C buss.

```
172 {
173     clear_pin(i2c_scl_port, i2c_scl_pin);
174     delay_us(DELAY_AMOUNT);
175 
176     i2c_write_sda();
177 
178     clear_pin(i2c_sda_port, i2c_sda_pin);
179     delay_us(DELAY_AMOUNT);
180     set_pin(i2c_scl_port, i2c_scl_pin);
181     delay_us(DELAY_AMOUNT);
182     set_pin(i2c_sda_port, i2c_sda_pin);
183     delay_us(DELAY_AMOUNT);
184 }
```

Here is the caller graph for this function:

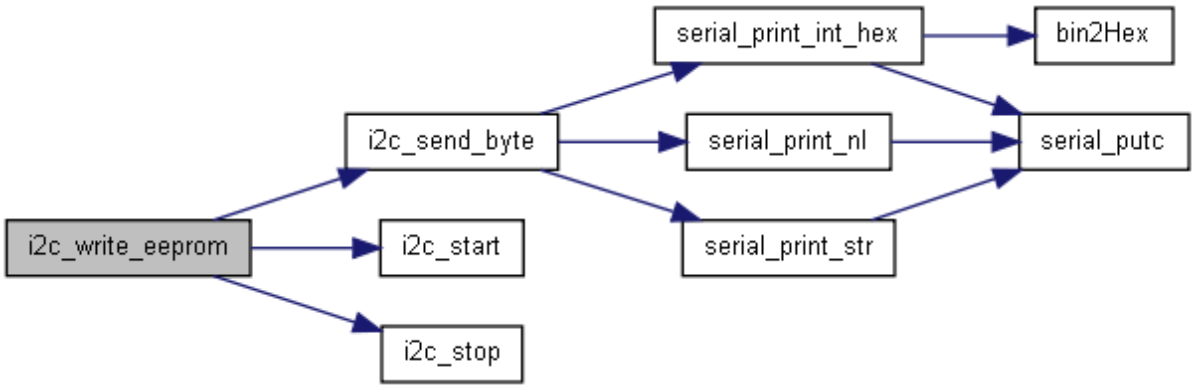

ds1631_convert_temp
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_read_ram
hmc6352_write_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_update_bridge_offsets
hmc6352_wake
hmc6352_write_eeprom
i2c_ack_polling
i2c_stop
i2c_read_eeprom
i2c_write_eeprom
i2c_write_eeprom_16bit
i2c_read_eeprom_16bit
ms5511_reset
tmp75_read
tmp75_read_16bit
tmp75_write
hmc6352_set_mode
rtc_get_minutes
rtc_get_month
rtc_get_register
rtc_get_seconds
rtc_get_year
rtc_set_hours
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
rtc_get_dow
rtc_get_hours
rtc_set_seconds
rtc_start_clock
rtc_stop_clock
ds1631_set_config
lm75_set_config
rtc_set_config
ar1000_seek
ar1000_test
ar1000_tune
ar1000_read_register
ar1000_seek_more
ds1631_get_temp
lm75_get_temp
ms5511_get_config

void i2c_write_eeprom (uns8 *device_address*, uns8 *mem_address*, uns8 *data*)

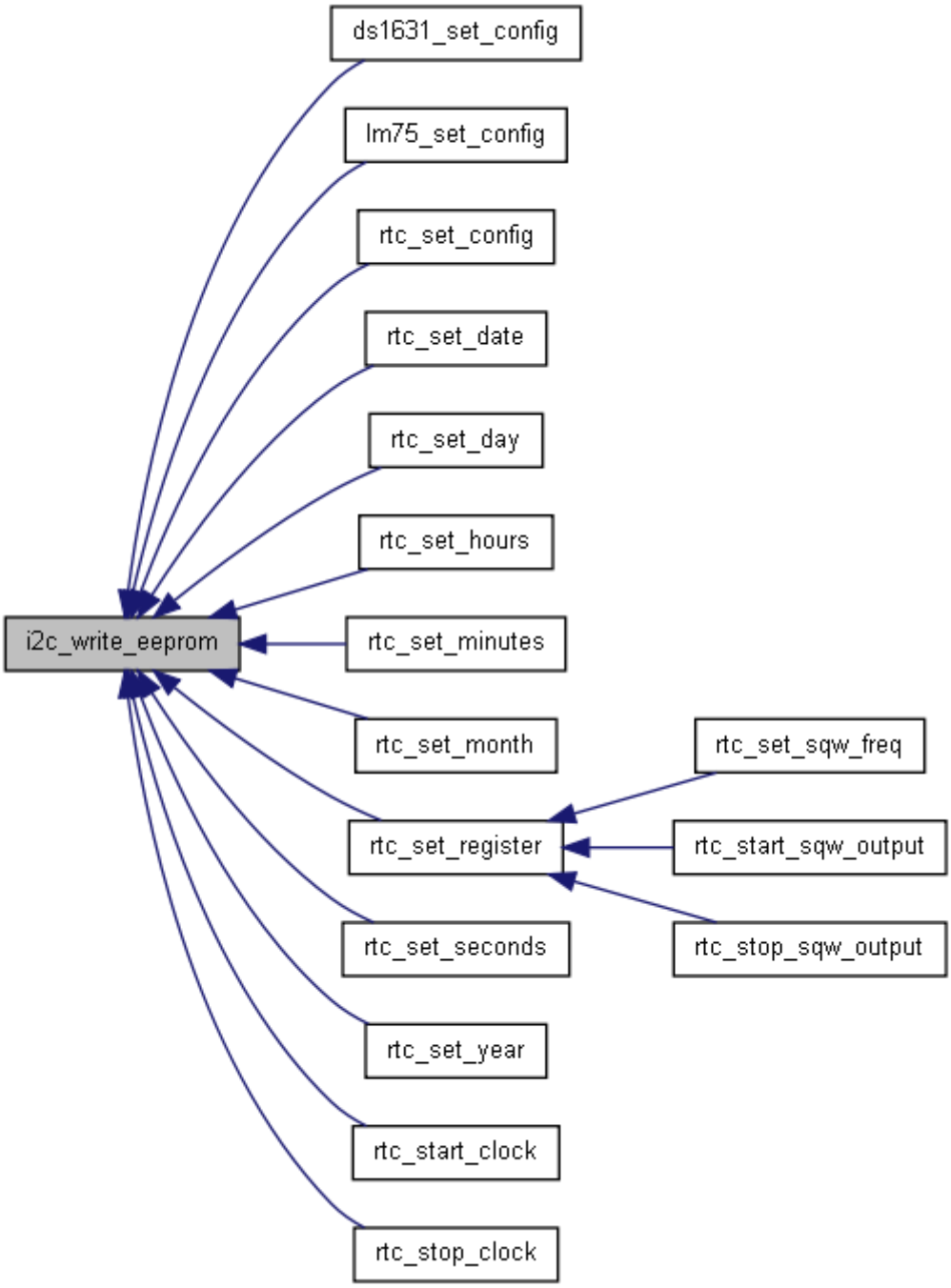
Write a byte to a given device address at the memory address

```
52 {
53     //i2c_ack_polling(device_address);
54     i2c_start();
55     i2c_send_byte(device_address);
56 
57     //send_byte(address.high8); //Uppder Address - Needed for >= 32k EEProms
58     i2c_send_byte(mem_address);
59     i2c_send_byte(data);
60     i2c_stop();
61 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



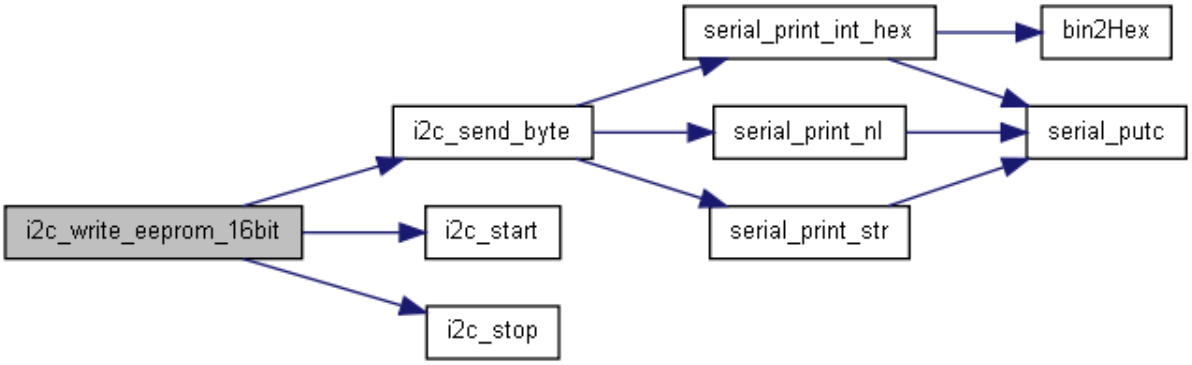
void i2c_write_eeprom_16bit (uns8 *device_address*, uns8 *mem_address*, uns16 *data*)

Write a byte to a given device address at the memory address

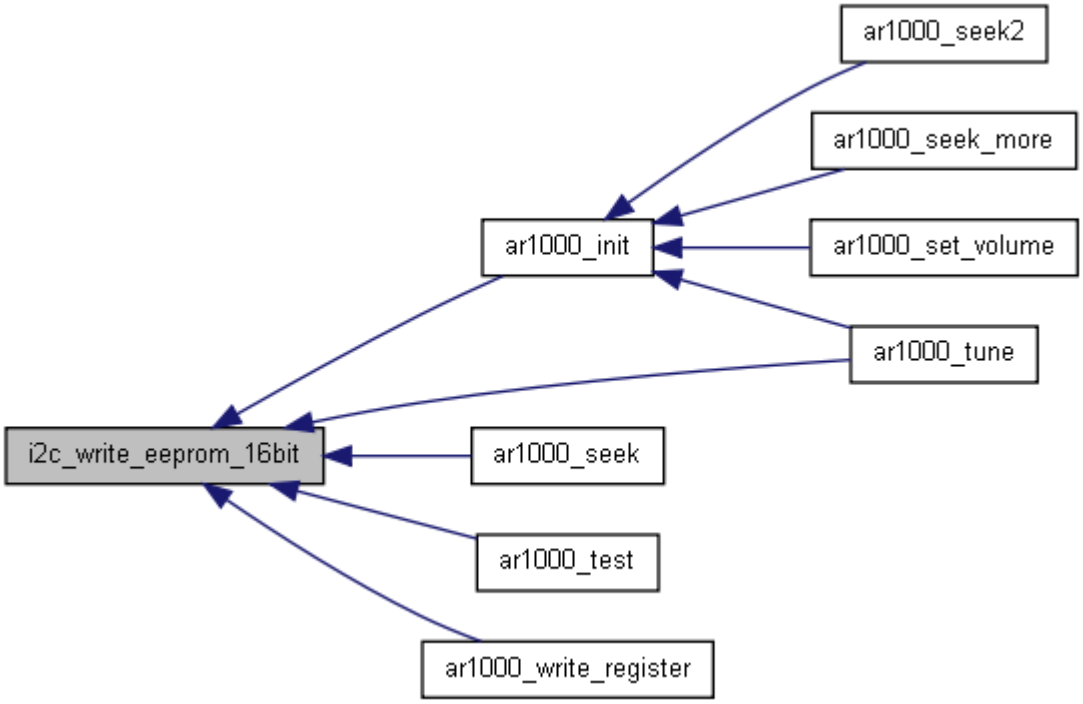
```
64 {
65     //i2c_ack_polling(device_address);
66     i2c_start();
67     i2c_send_byte(device_address);
68 
69     //send_byte(address.high8); //Uppder Address - Needed for >= 32k EEProms
70     i2c_send_byte(mem_address);
71     i2c_send_byte(data >> 8);
72     i2c_send_byte(data & 0x00ff);
```

```
73      i2c_stop();
74  }
```

Here is the call graph for this function:



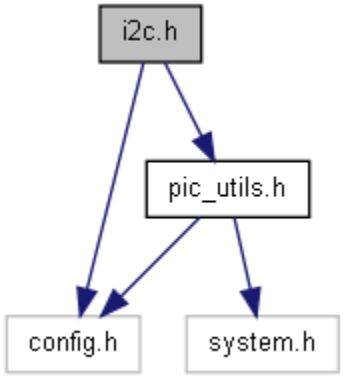
Here is the caller graph for this function:



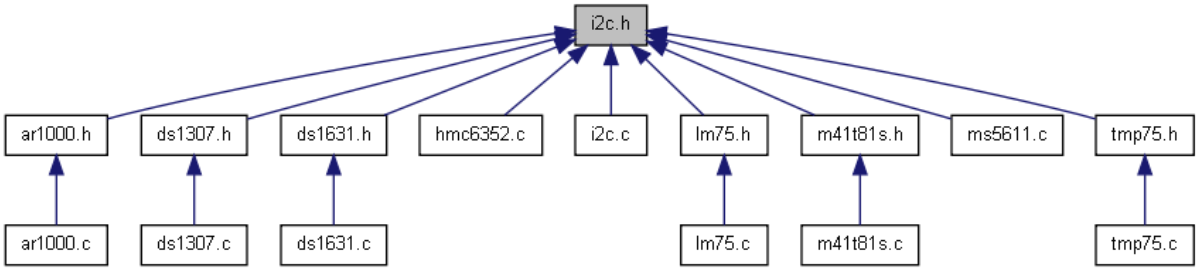
---

i2c.h File Reference

I2C software routines.
Include dependency graph for i2c.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define __i2c_h defined
- #define i2c_read_sda() set_bit(tris_array[i2c_sda_port - PORTA], i2c_sda_pin);
- #define i2c_setup() i2c_setup_io()
- #define i2c_write_sda() clear_bit(tris_array[i2c_sda_port - PORTA], i2c_sda_pin);

Functions

- uns8 i2c_read_eeprom (uns8 device_address, uns8 mem_address)
- *Read an 8 bit byte over I2C buss.* uns16 i2c_read_eeprom_16bit (uns8 device_address, uns8 mem_address)
- *Read 16 bits of data over I2C buss.* uns8 i2c_receive_byte ()
- *Receive byte from I2C buss.* void i2c_send_ack (void)
- *Send an ACK.* void i2c_send_byte (uns8 data)
- *Send a byte to I2C buss.* void i2c_setup_io ()
- *Setup ports and pins for I2C communication.* void i2c_start (void)
- *Send start signal to I2C buss.* void i2c_stop (void)
- *Send stop signal to I2C buss.* void i2c_write_eeprom (uns8 device_address, uns8 mem_address, uns8 data)
- *Write an 8 bit byte ove I2C buss.* void i2c_write_eeprom_16bit (uns8 device_address, uns8 mem_address, uns16 data)

Write a 16 bit value over I2C buss.

Detailed Description

I2C communication routines. Although all standard functions are provided, you should only need to use i2c_setup, i2c_read_eeprom and i2c_write_eeprom

Define Documentation

#define __i2c_h defined

#define i2c_read_sda() set_bit(tris_array[i2c_sda_port - PORTA], i2c_sda_pin);

Change SDA line to read mode

#define i2c_setup() i2c_setup_io()

#define i2c_write_sda() clear_bit(tris_array[i2c_sda_port - PORTA], i2c_sda_pin);

Change SDA line to write mode

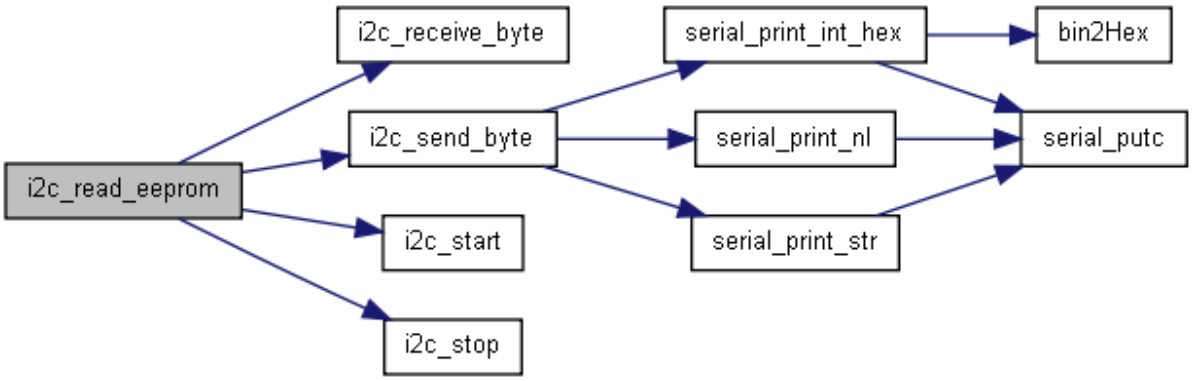
Function Documentation

uns8 i2c_read_eeprom (uns8 *device_address*, uns8 *mem_address*)

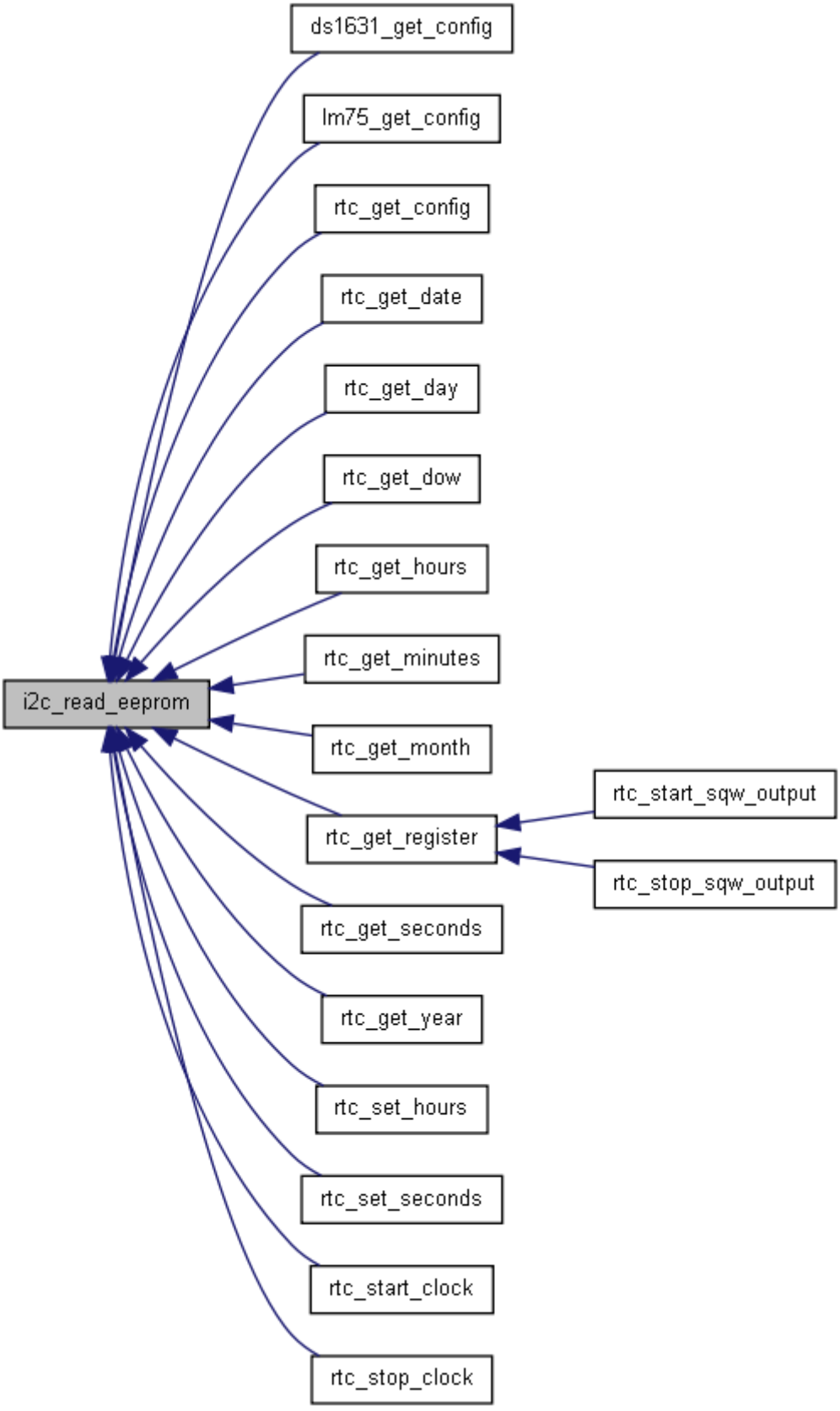
Read an 8 bit byte from the specified device at the memory address

```
77 {
78 uns8 data;
79     //i2c_ack_polling(device_address);
80 
81     i2c_start();
82     i2c_send_byte(device_address);
83     //send_byte(address.high8); // Upper Address - Needed for >= 32k EEProms
84     i2c_send_byte(mem_address);
85     i2c_stop();
86 
87     i2c_start();
88     i2c_send_byte(device_address | 0b00000001); // Read bit must be set
89     data = i2c_receive_byte(); // reuse local variable
90 
91     // write an nack
92     clear_pin(i2c_scl_port, i2c_scl_pin);
93     i2c_write_sda();
94     delay_us(DELAY_AMOUNT);
95     set_pin(i2c_sda_port, i2c_sda_pin);
96     set_pin(i2c_scl_port, i2c_scl_pin);
97     delay_us(DELAY_AMOUNT);
98 
99     i2c_stop();
100     return(data);
101 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

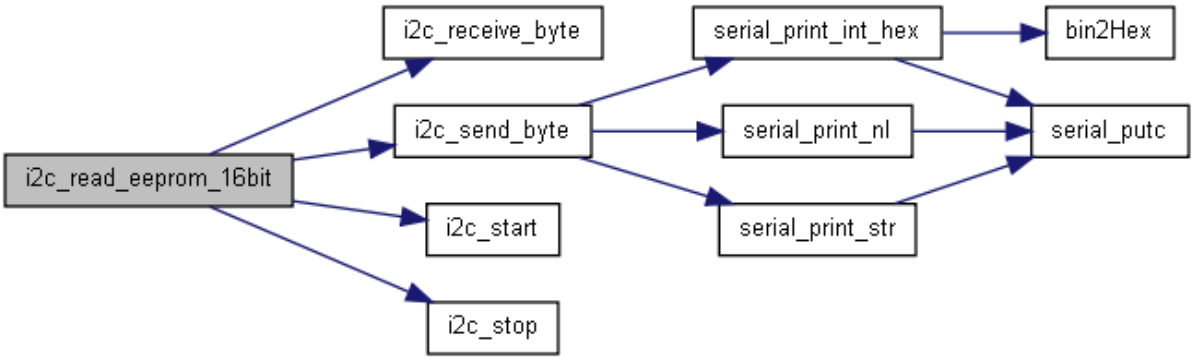


uns16 i2c_read_eeprom_16bit (uns8 *device_address*, uns8 *mem_address*)

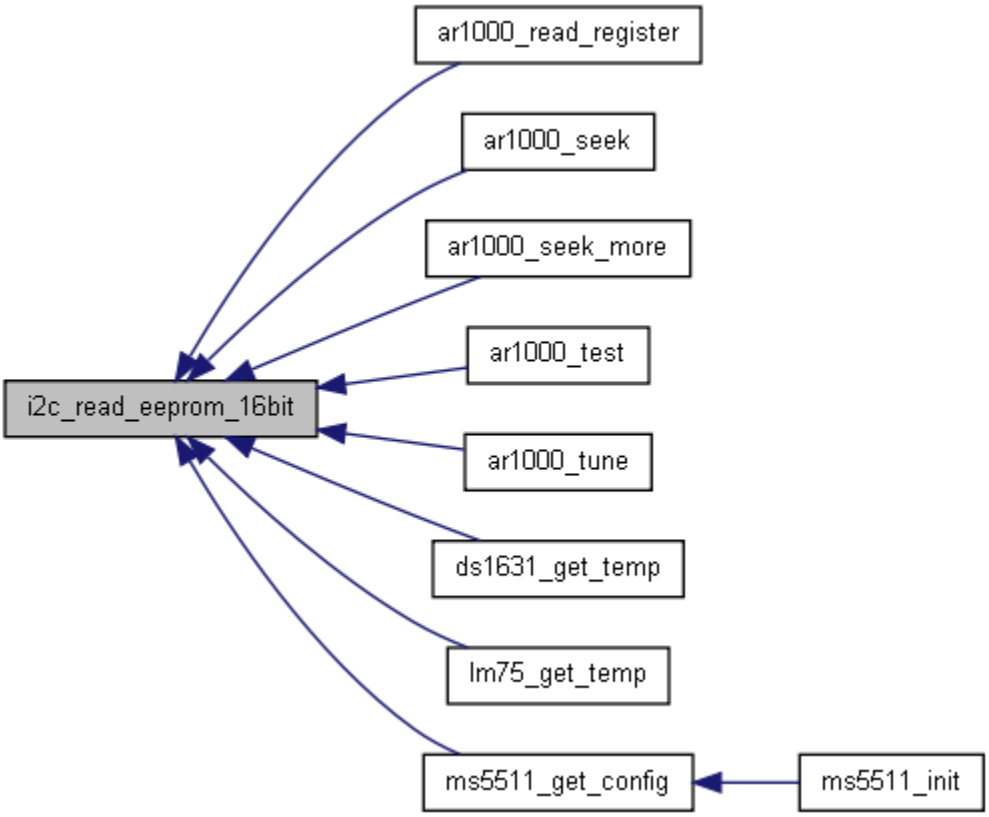
Read a 16 bit chunk of data from the given device and memory address

```
104 {
105     uns16 data;
106 
107     //i2c_ack_polling(device_address);
108     i2c_start();
109     i2c_send_byte(device_address);
110     //send_byte(address.high8); // Upper Address - Needed for >= 32k EEProms
111     i2c_send_byte(mem_address);
112 
113     i2c_start();
114     i2c_send_byte(device_address | 0b00000001); // Read bit must be set
115     data = i2c_receive_byte(); // reuse local variable
116     i2c_write_sda();
117 
118     clear_pin(i2c_scl_port, i2c_scl_pin);
119     delay_us(DELAY_AMOUNT);
120     clear_pin(i2c_sda_port, i2c_sda_pin); //ack
121     delay_us(DELAY_AMOUNT);
122     set_pin(i2c_scl_port, i2c_scl_pin);
123     delay_us(DELAY_AMOUNT);
124 
125     data = data << 8 | i2c_receive_byte(); // reuse local variable
126 
127     // write an nack
128     clear_pin(i2c_scl_port, i2c_scl_pin);
129     i2c_write_sda();
130     delay_us(DELAY_AMOUNT);
131     set_pin(i2c_sda_port, i2c_sda_pin);
132     set_pin(i2c_scl_port, i2c_scl_pin);
133     delay_us(DELAY_AMOUNT);
134 
135     i2c_stop();
136 
137     return(data);
138 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

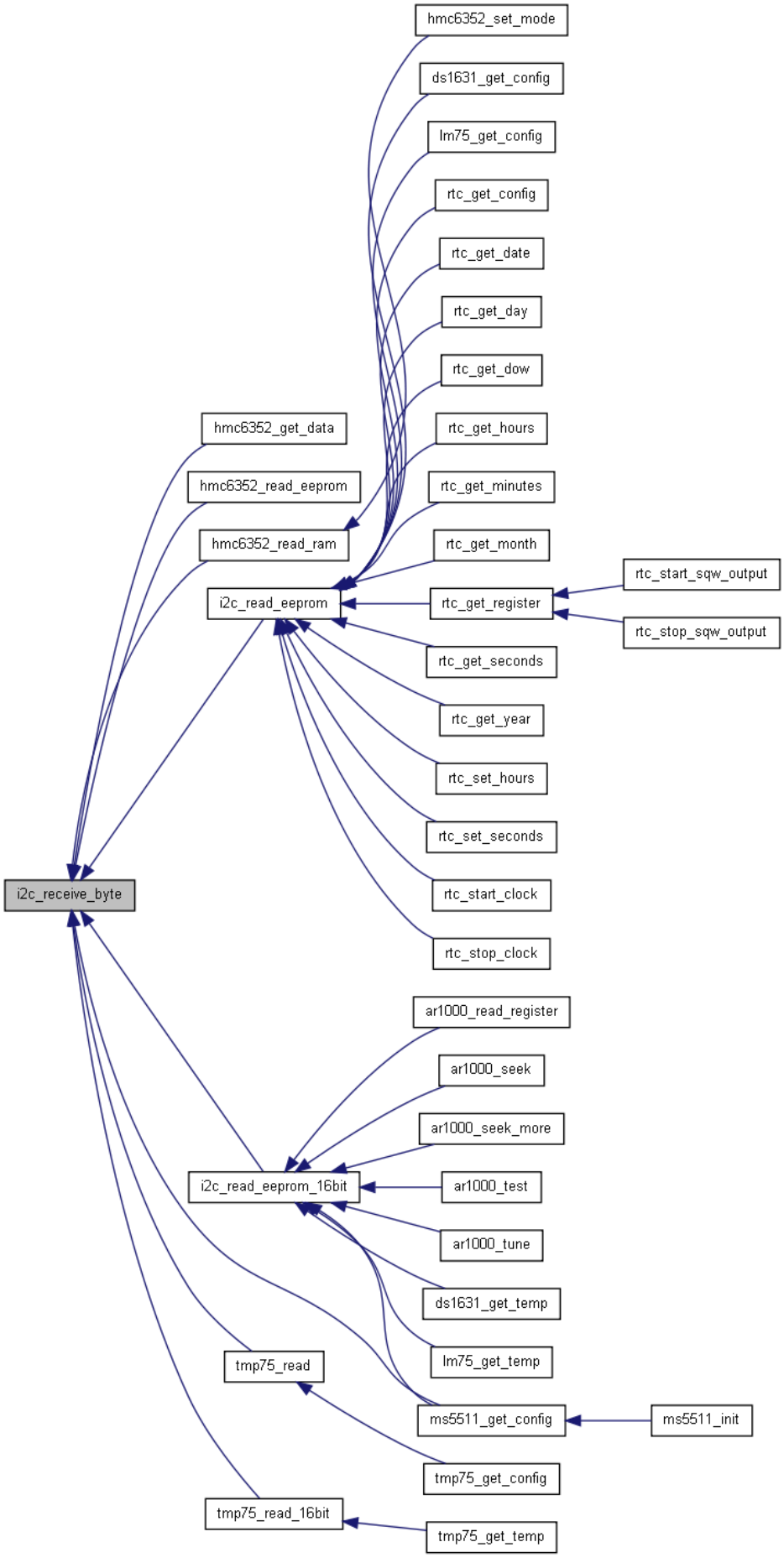


uns8 i2c_receive_byte ()

Clock in a byte over I2C lines. Note that an acknowledge is not sent/received

```
187 {
188    uns8 count, in_byte;
189 
190     clear_pin(i2c_scl_port, i2c_scl_pin);
191 
192     i2c_read_sda();
193     for(count = 0 ; count < 8 ; count++)
194     {
195         clear_pin(i2c_scl_port, i2c_scl_pin);
196         delay_us(DELAY_AMOUNT);
197         set_pin(i2c_scl_port, i2c_scl_pin);
198         delay_us(DELAY_AMOUNT);
199 
200         in_byte = in_byte << 1;
201         in_byte.0 = test_pin(i2c_sda_port, i2c_sda_pin);
202     }
203     return(in_byte);
204 }
```

Here is the caller graph for this function:


hmc6352_set_mode
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
rtc_get_dow
rtc_get_hours
rtc_get_minutes
rtc_get_month
rtc_get_register
rtc_start_sqw_output
rtc_stop_sqw_output
rtc_get_seconds
rtc_get_year
rtc_set_hours
rtc_set_seconds
rtc_start_clock
rtc_stop_clock
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_read_ram
i2c_read_eeprom
i2c_receive_byte
ar1000_read_register
ar1000_seek
ar1000_seek_more
ar1000_test
ar1000_tune
i2c_read_eeprom_16bit
ds1631_get_temp
lm75_get_temp
ms5511_get_config
ms5511_init
tmp75_read
tmp75_get_config
tmp75_read_16bit
tmp75_get_temp

void i2c_send_ack (void)

Sends an I2C acknowledge (bit)

```
140                         {
141
142     i2c_write_sda();
143
144     clear_pin(i2c_scl_port, i2c_scl_pin);
145     delay_us(DELAY_AMOUNT);
146     clear_pin(i2c_sda_port, i2c_sda_pin); //ack
147     delay_us(DELAY_AMOUNT);
148     set_pin(i2c_scl_port, i2c_scl_pin);
149     delay_us(DELAY_AMOUNT);
150
151
152 }
```

Here is the caller graph for this function:

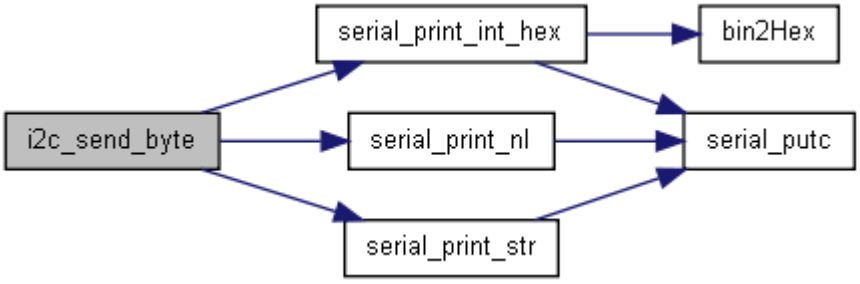
void i2c_send_byte (uns8 *data*)

Clock out a byte from the I2C buss. An acknowledge is "received" (although ignored).

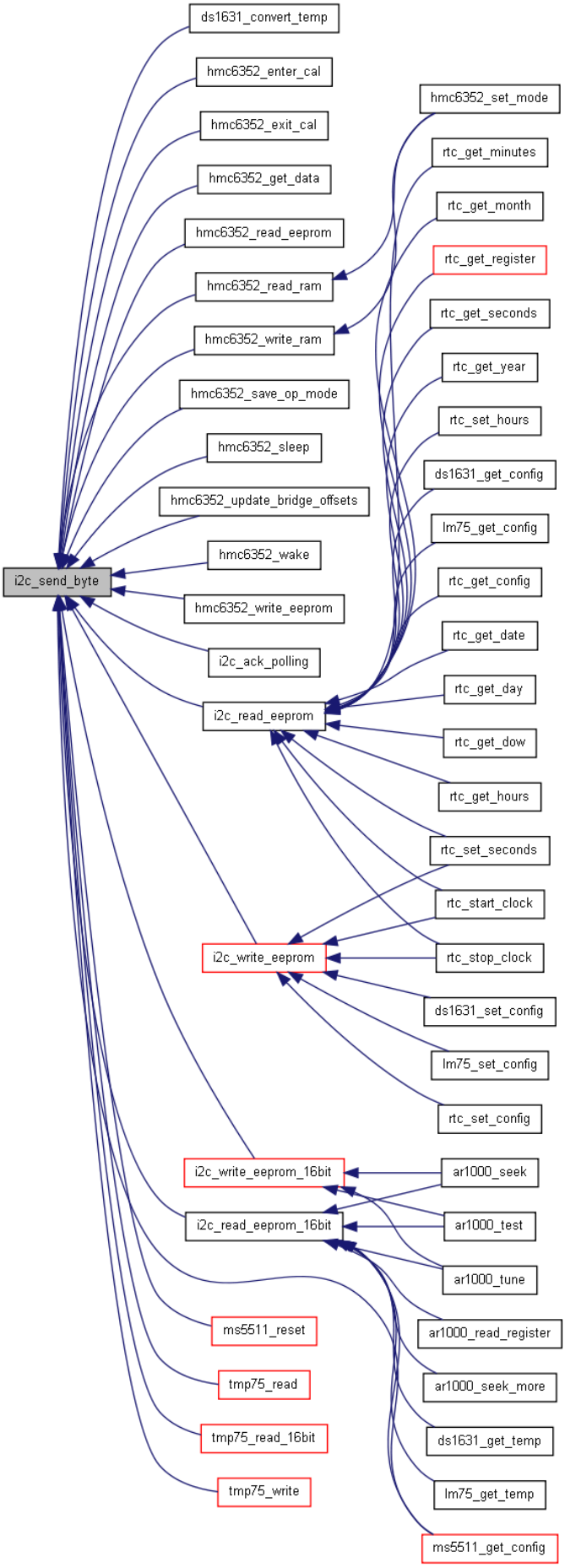
```
207 {
208    uns8 count;
209 serial_print_str("Sending: ");
210 serial_print_int_hex(data);
211 serial_print_nl();
212     clear_pin(i2c_scl_port, i2c_scl_pin);
213     i2c_write_sda();
214
215     for( count = 0 ; count < 8 ; count++ )
216     {
217         clear_pin(i2c_scl_port, i2c_scl_pin);
218         change_pin(i2c_sda_port, i2c_sda_pin, data.7);
219         data = data << 1;
220         delay_us(DELAY_AMOUNT);
221         set_pin(i2c_scl_port, i2c_scl_pin);
222         delay_us(DELAY_AMOUNT);
223     }
224
225     // read ack (and ignore)
226     clear_pin(i2c_scl_port, i2c_scl_pin);
227     i2c_read_sda();
228     delay_us(DELAY_AMOUNT);
229     set_pin(i2c_scl_port, i2c_scl_pin);
230     // read at this point
231
232     delay_us(DELAY_AMOUNT);
```

```
233 }
```

Here is the call graph for this function:



Here is the caller graph for this function:


ds1631_convert_temp
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_read_ram
hmc6352_write_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_update_bridge_offsets
hmc6352_wake
hmc6352_write_eeprom
i2c_ack_polling
i2c_send_byte
i2c_read_eeprom
i2c_write_eeprom
i2c_write_eeprom_16bit
i2c_read_eeprom_16bit
ms5511_reset
tmp75_read
tmp75_read_16bit
tmp75_write
hmc6352_set_mode
rtc_get_minutes
rtc_get_month
rtc_get_register
rtc_get_seconds
rtc_get_year
rtc_set_hours
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
rtc_get_dow
rtc_get_hours
rtc_set_seconds
rtc_start_clock
rtc_stop_clock
ds1631_set_config
lm75_set_config
rtc_set_config
ar1000_seek
ar1000_test
ar1000_tune
ar1000_read_register
ar1000_seek_more
ds1631_get_temp
lm75_get_temp
ms5511_get_config

void i2c_setup_io ()

Set port and pins correctly for I2C communication

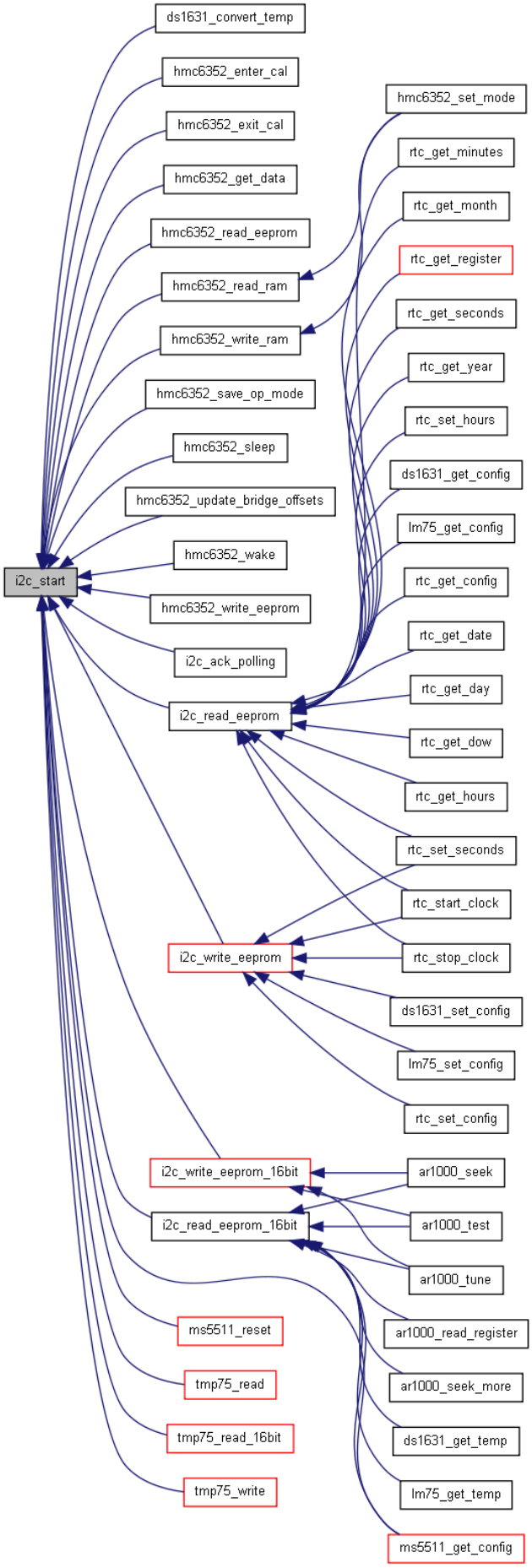
```
235                     {
236     make_output(i2c_scl_port, i2c_scl_pin);
237     make_input(i2c_sda_port, i2c_sda_pin);
238 }
```

void i2c_start (void)

Signals a start condition on the I2C buss.

```
156 {
157 
158     clear_pin(i2c_scl_port, i2c_scl_pin);
159     delay_us(DELAY_AMOUNT);
160 
161     i2c_write_sda();
162 
163     set_pin(i2c_sda_port, i2c_sda_pin);
164     delay_us(DELAY_AMOUNT);
165     set_pin(i2c_scl_port, i2c_scl_pin);
166     delay_us(DELAY_AMOUNT);
167     clear_pin(i2c_sda_port, i2c_sda_pin);
168     delay_us(DELAY_AMOUNT);
169 }
```

Here is the caller graph for this function:

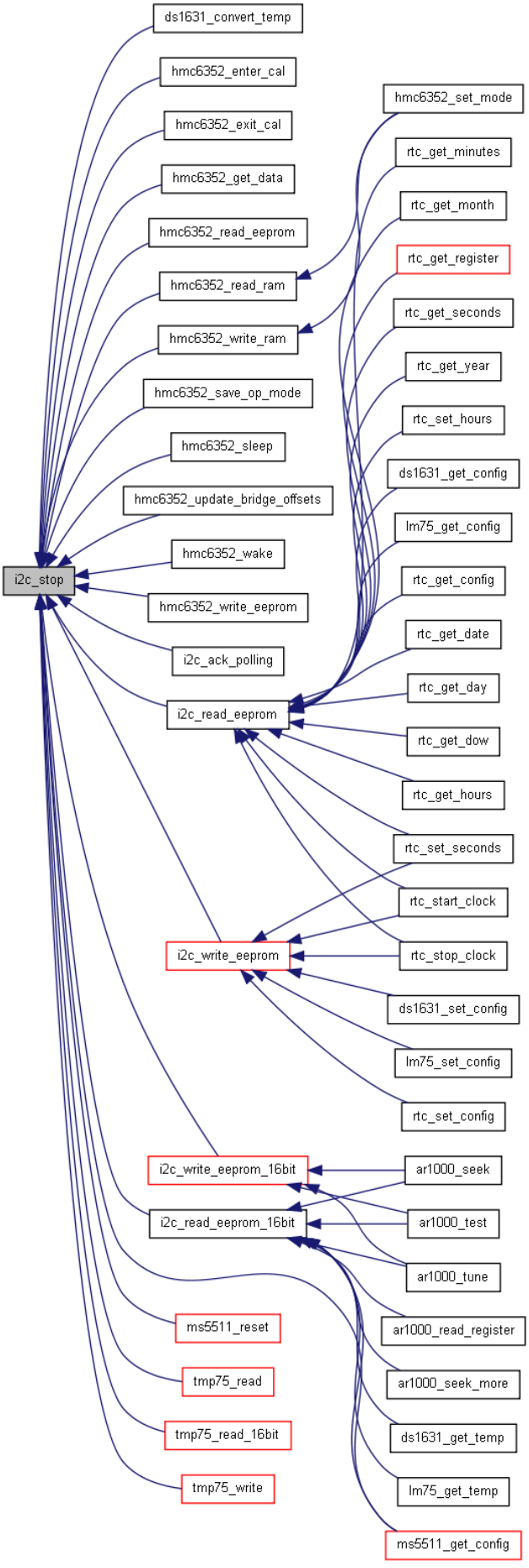

ds1631_convert_temp
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_read_ram
hmc6352_write_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_update_bridge_offsets
hmc6352_wake
hmc6352_write_eeprom
i2c_ack_polling
i2c_start
i2c_read_eeprom
i2c_write_eeprom
i2c_write_eeprom_16bit
i2c_read_eeprom_16bit
ms5511_reset
tmp75_read
tmp75_read_16bit
tmp75_write
hmc6352_set_mode
rtc_get_minutes
rtc_get_month
rtc_get_register
rtc_get_seconds
rtc_get_year
rtc_set_hours
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
rtc_get_dow
rtc_get_hours
rtc_set_seconds
rtc_start_clock
rtc_stop_clock
ds1631_set_config
lm75_set_config
rtc_set_config
ar1000_seek
ar1000_test
ar1000_tune
ar1000_read_register
ar1000_seek_more
ds1631_get_temp
lm75_get_temp
ms5511_get_config

void i2c_stop (void)

Signals a stop condition on the I2C buss.

```
172 {
173     clear_pin(i2c_scl_port, i2c_scl_pin);
174     delay_us(DELAY_AMOUNT);
175 
176     i2c_write_sda();
177 
178     clear_pin(i2c_sda_port, i2c_sda_pin);
179     delay_us(DELAY_AMOUNT);
180     set_pin(i2c_scl_port, i2c_scl_pin);
181     delay_us(DELAY_AMOUNT);
182     set_pin(i2c_sda_port, i2c_sda_pin);
183     delay_us(DELAY_AMOUNT);
184 }
```

Here is the caller graph for this function:

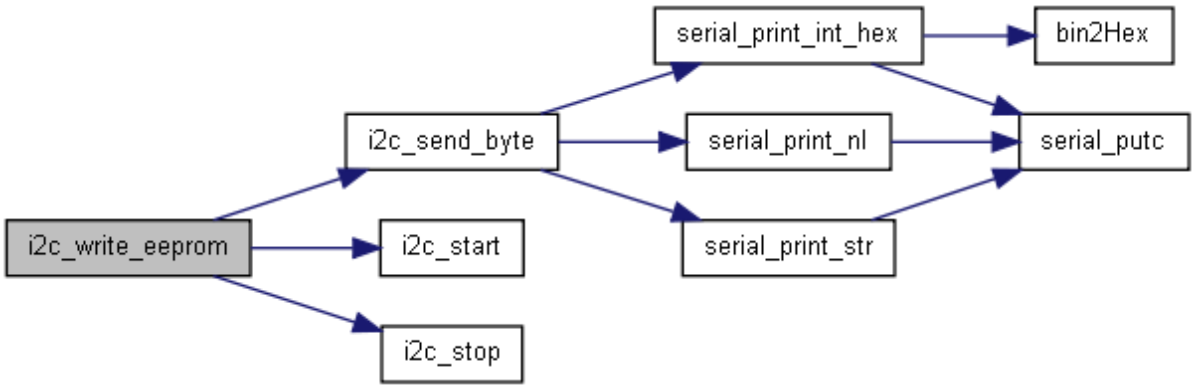

ds1631_convert_temp
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_read_ram
hmc6352_write_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_update_bridge_offsets
hmc6352_wake
hmc6352_write_eeprom
i2c_ack_polling
i2c_stop
i2c_read_eeprom
i2c_write_eeprom
i2c_write_eeprom_16bit
i2c_read_eeprom_16bit
ms5511_reset
tmp75_read
tmp75_read_16bit
tmp75_write
hmc6352_set_mode
rtc_get_minutes
rtc_get_month
rtc_get_register
rtc_get_seconds
rtc_get_year
rtc_set_hours
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
rtc_get_dow
rtc_get_hours
rtc_set_seconds
rtc_start_clock
rtc_stop_clock
ds1631_set_config
lm75_set_config
rtc_set_config
ar1000_seek
ar1000_test
ar1000_tune
ar1000_read_register
ar1000_seek_more
ds1631_get_temp
lm75_get_temp
ms5511_get_config

void i2c_write_eeprom (uns8 *device_address*, uns8 *mem_address*, uns8 *data*)

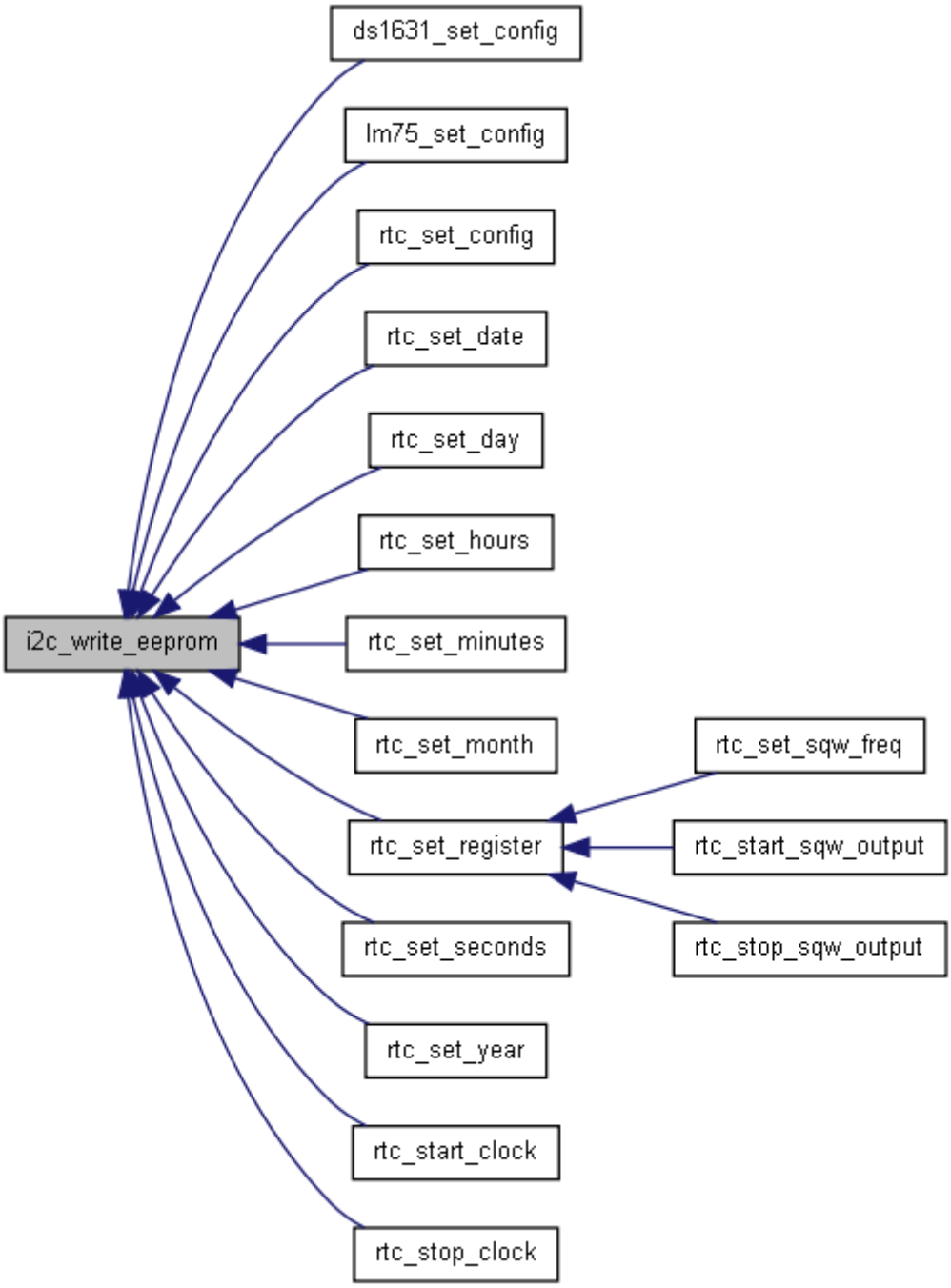
Write a byte to a given device address at the memory address

```
52 {
53     //i2c_ack_polling(device_address);
54     i2c_start();
55     i2c_send_byte(device_address);
56 
57     //send_byte(address.high8); //Uppder Address - Needed for >= 32k EEProms
58     i2c_send_byte(mem_address);
59     i2c_send_byte(data);
60     i2c_stop();
61 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



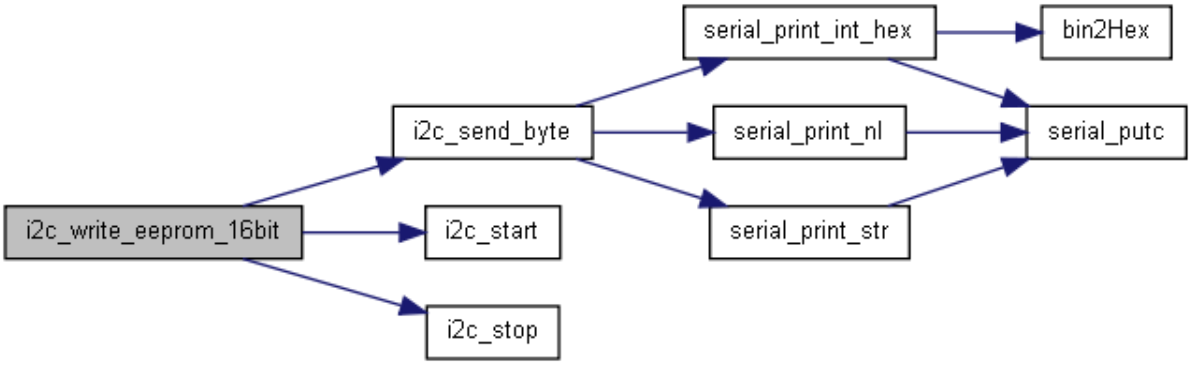
void i2c_write_eeprom_16bit (uns8 *device_address*, uns8 *mem_address*, uns16 *data*)

Write a byte to a given device address at the memory address

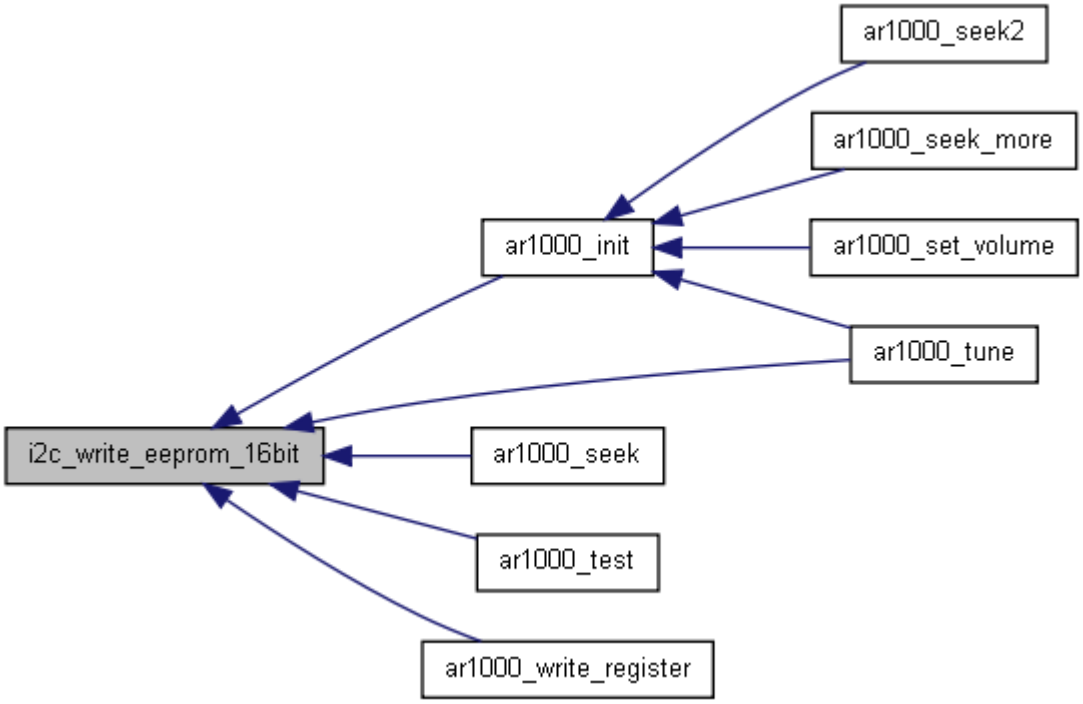
```
64 {
65     //i2c_ack_polling(device_address);
66     i2c_start();
67     i2c_send_byte(device_address);
68 
69     //send_byte(address.high8); //Uppder Address - Needed for >= 32k EEProms
70     i2c_send_byte(mem_address);
71     i2c_send_byte(data >> 8);
72     i2c_send_byte(data & 0x00ff);
```

```
73      i2c_stop();
74  }
```

Here is the call graph for this function:



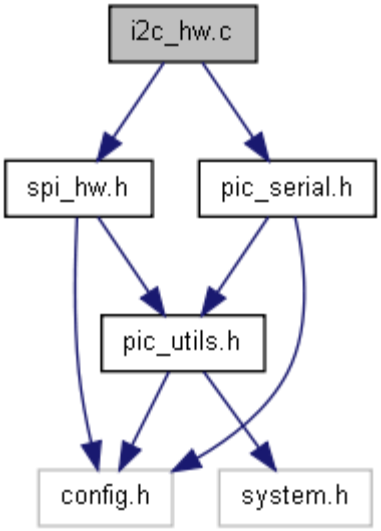
Here is the caller graph for this function:



---

i2c_hw.c File Reference

Include dependency graph for i2c_hw.c:



Functions

- void spi_hw_init ()
- uns8 spi_hw_receive ()
- void spi_hw_setup_io ()
- void spi_hw_transmit (uns8 data)

Function Documentation

void spi_hw_init ()

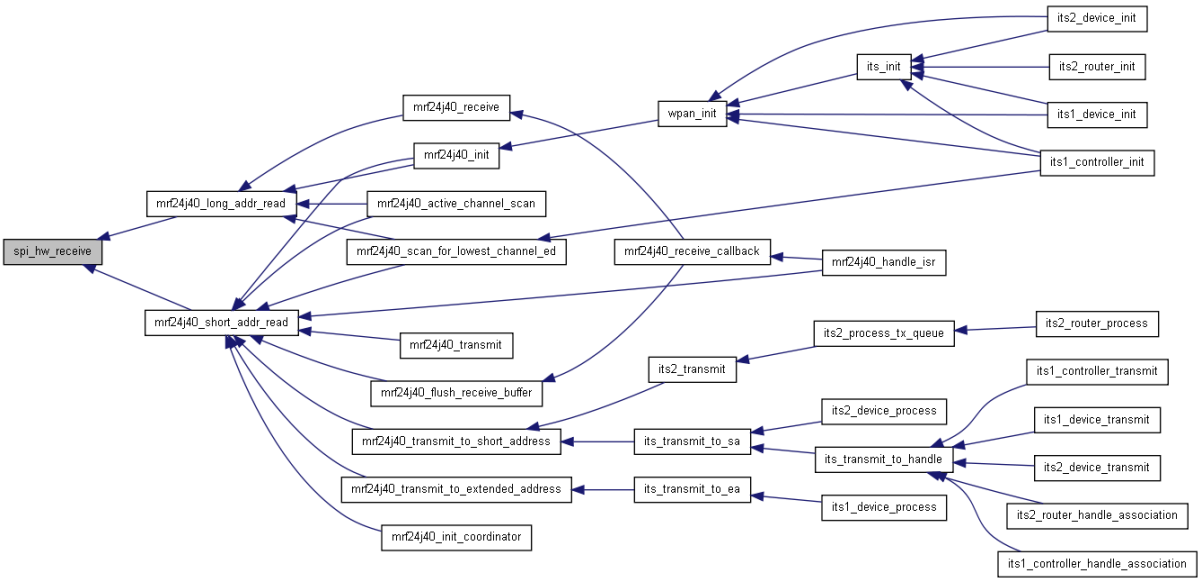
```
82                      {
83     // for 0,0 mode
84     clear_bit(sspcon1, CKP);
85
86     set_bit(sspstat, CKE);
87     set_bit(sspstat, SMP);  // from tim
88     set_bit(sspcon1, SSPEN);    // enable mssp serial port
89 }
```

uns8 spi_hw_receive ()

```
106                         {
107
108     spi_hw_transmit(0x00);  // dummy transmit to get something back
109     return sspbuf;
110
111 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



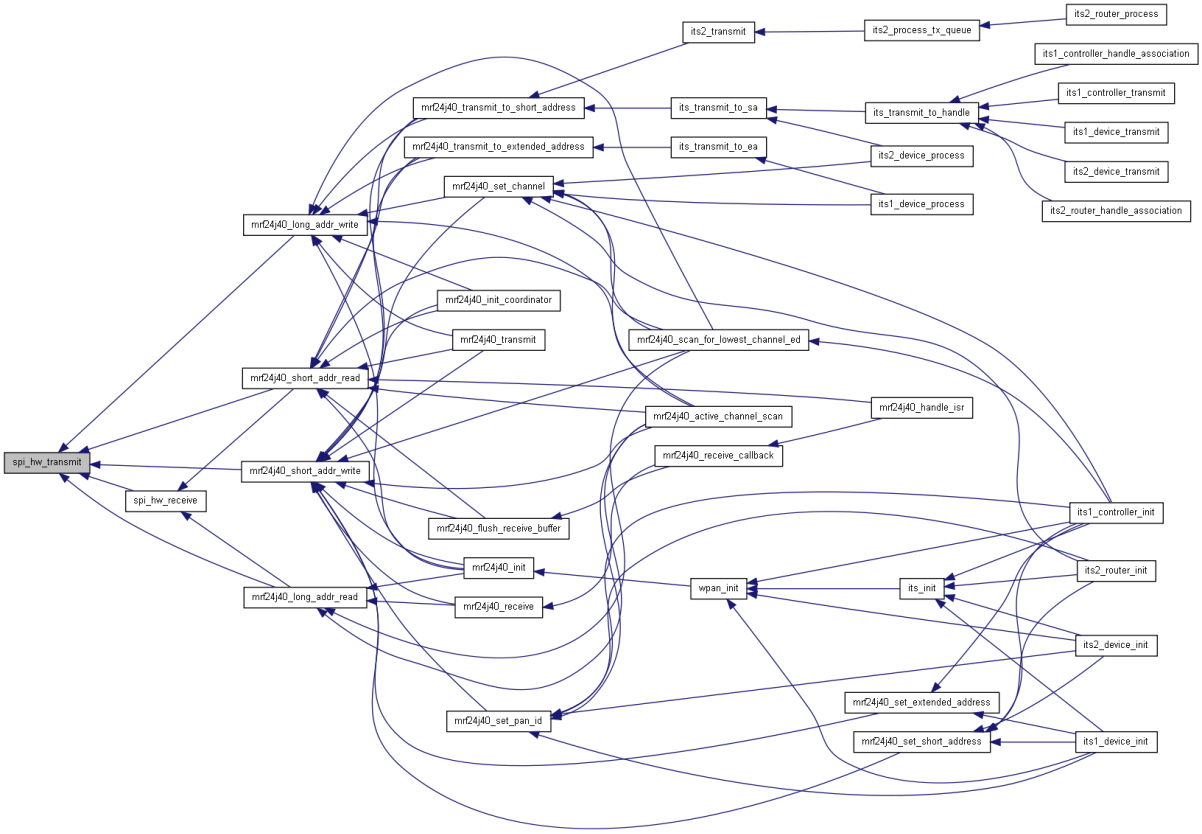
void spi_hw_setup_io ()

```
40                      {
41
42 make_output(PORTC, 5);
43 make_input(PORTC, 4);
44
45 #ifdef SPI_HW_MASTER_MODE
46     make_output(PORTC, 3);
47     clear_bit(sspcon1, 3);
48     clear_bit(sspcon1, 2);
49     #ifdef SPI_HW_MASTER_CLOCK_TMR2_DIV_2
50         set_bit(sspcon1, 1);
51         set_bit(sspcon1, 0);
52     #endif
53     #ifdef SPI_HW_MASTER_CLOCK_FOSC_DIV_64
54         set_bit  (sspcon1, 1);
55         clear_bit(sspcon1, 0);
56     #endif
57     #ifdef SPI_HW_MASTER_CLOCK_FOSC_DIV_16
58         clear_bit(sspcon1, 1);
59         set_bit  (sspcon1, 0);
60     #endif
61     #ifdef SPI_HW_MASTER_CLOCK_FOSC_DIV_4
62         clear_bit(sspcon1, 1);
63         clear_bit(sspcon1, 0);
64     #endif
65 #else
66     // Slave mode
67     make_input(PORTC, 3);   // sck
68     clear_bit(sspcon1, 3);
69     set_bit  (sspcon1, 2);
70     clear_bit(sspcon1, 1);
71     #ifdef SPI_HW_USE_SS
72         make_input(PORTA, 5);   // SS
73         clear_bit(sspcon1, 0);
74     #else
75         set_bit(sspcon1, 0);
76     #endif
77 #endif
78
79 }
```

void spi_hw_transmit (uns8 *data*)

```
91                                  {
92
93      clear_bit(pir1, SSPIF);
94      // Try and send, if we have something already sending, try again
95      do {
96          clear_bit(sspcon1, WCOL);
97          sspbuf = data;
98      } while (test_bit(sspcon1, WCOL));
99
100      // Wait here while we transmit
101      // Note that we will hang here if we don't have SPI setup properly
102      // or we're running under the simulator
103      while (!test_bit(pir1, SSPIF)) {}
104 }
```

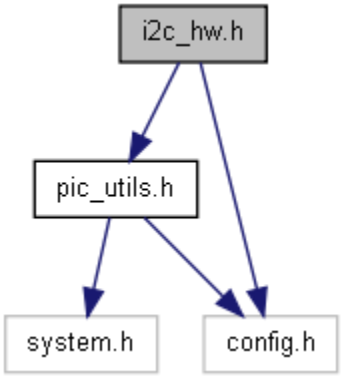
Here is the caller graph for this function:



---

i2c_hw.h File Reference

Outputs i2c interfaces (clock+data).
Include dependency graph for i2c_hw.h:



Functions

- void i2c_pulse_0 ()
- *i2c test routine* void i2c_pulse_1 ()
- *i2c test routine* void i2c_setup_io ()
- *Setup ports and pins for i2c output.* void i2c_write (uns8 data)
- *Send a byte of data using software i2c.* void i2c_write_lsb (uns8 data)

Send a byte of data using software i2c.

Detailed Description

Covers standard i2c-like interfaces (clock + data) and Sure Electronics displays which are a little different

---

Function Documentation

void i2c_pulse_0 ()

void i2c_pulse_1 ()

void i2c_setup_io ()

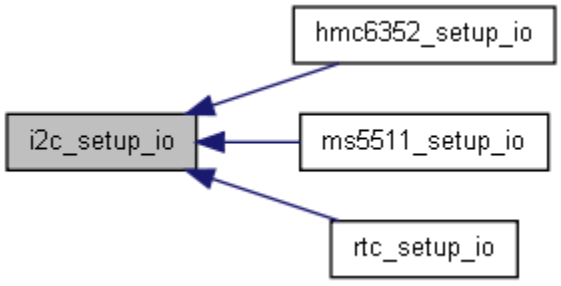
Setup ports and pins for i2c output

Setup ports and pins for i2c output.

Set port and pins correctly for I2C communication

```
235                     {
236     make_output(i2c_scl_port, i2c_scl_pin);
237     make_input(i2c_sda_port, i2c_sda_pin);
238 }
```

Here is the caller graph for this function:



void i2c_write (uns8 *data*)

Sends a byte of data MSB first, data only changes on clock low

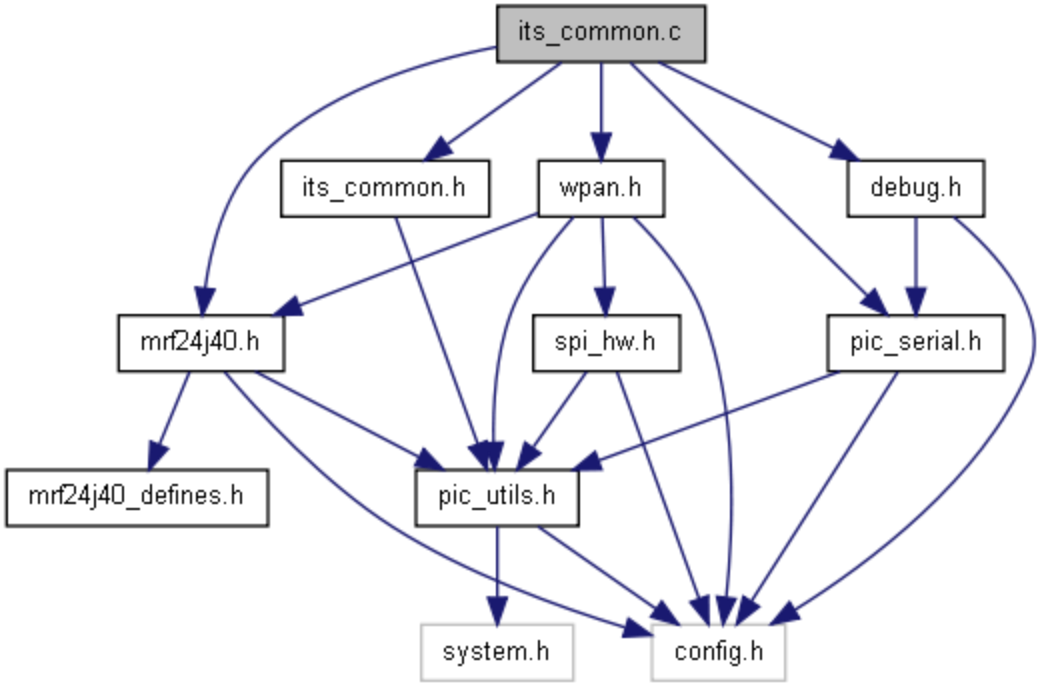
void i2c_write_lsb (uns8 *data*)

Sends a byte of data LSB first, data only changes on clock low

---

its_common.c File Reference

Include dependency graph for its_common.c:



Functions

- its_device_handle its_add_local_device (uns16 device_id, uns16 pan_id, uns16 short_address)
- its_device_handle its_add_net_device (uns16 device_id, uns16 previous_hop)
- its_device_handle its_get_device_handle (uns16 device_id)
- uns16 its_get_device_id ()
- its_device_info * its_get_device_info (uns8 handle)
- uns16 its_get_network_id ()
- *Retrieve the current network ID.* uns8 its_get_next_sequence ()
- void its_init ()
- *Initialise ITS and lower layers.* void its_print_devices ()
- *Print devices currently known to this one.* void its_set_device_id (uns16 device_id)
- *Set the ITS device ID.* void its_set_network_id (uns16 network_id)
- *Set the ITS network ID.* void its_transmit_to_ea (uns8 *dest_ea, uns16 dest_its_device_id, uns8 packet_type, uns8 *data, uns8 data_length)

- void its_transmit_to_handle (its_device_handle handle, uns8 packet_type, uns8 *data, uns8 data_length)
- void its_transmit_to_sa (uns16 dest_pan_id, uns16 dest_sa, uns16 dest_device_id, uns8 packet_type, uns8 *data, uns8 data_length)

Variables

- uns16 its_device_id
- its_device_info its_devices [ITS_MAX_KNOWN_DEVICES]
- uns16 its_network_id
- uns8 its_sequence = 0

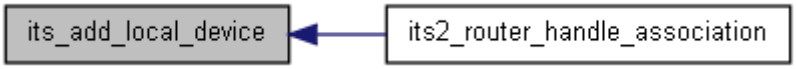
Function Documentation

its_device_handle its_add_local_device (uns16 *device_id*, uns16 *pan_id*, uns16 *short_address*)

```
127
{
128
129 bit found_slot = 0;
130     debug_str("Adding device_id (");
131     debug_int_hex_16bit(device_id);
132     debug_str(" (pan_id ");
133     debug_int_hex_16bit(pan_id);
134     debug_str(" short_addr ");
135     debug_int_hex_16bit(short_address);
136     debug_str(")\n");
137
138     for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
139         if (its_devices[count].its_device_id == 0xffff) {
140             found_slot = 1;
141             break;
142         }
143     }
144     if (found_slot) {
145         its_devices[count].its_device_id = device_id;
146         its_devices[count].addr.local.pan_id = pan_id;
147         its_devices[count].addr.local.short_address = short_address;
148     } else {
149         count = ITS_DEVICE_NONE;
150     }
151
152     its_print_devices();
153
154     return count;
155
156 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



its_device_handle its_add_net_device (uns16 *device_id*, uns16 *previous_hop*)

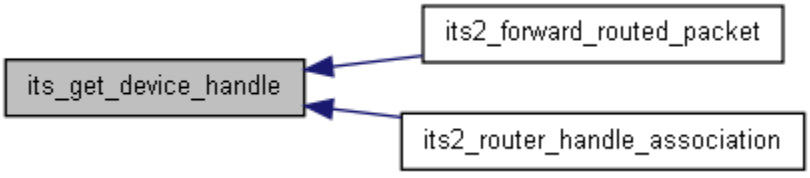
```
158                                                                                 {
159
160 bit found_slot = 0;
161     debug_str(" Add ");
162     debug_int_hex_16bit(device_id);
163     debug_str(" (via  ");
164     debug_int_hex_16bit(previous_hop);
165     debug_str(")\n");
166
167     for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
168         if (its_devices[count].its_device_id == 0xffff) {
169             found_slot = 1;
170             break;
171         }
172     }
173     if (found_slot) {
174         its_devices[count].its_device_id = device_id;
175         its_devices[count].addr.remote.remote_indicator = 0xffff;
176         its_devices[count].addr.remote.prior_device_id = previous hop;
177     } else {
178         count = ITS_DEVICE_NONE;
179     }
180     its_print_devices();
181     return count;
182 }
```

Here is the call graph for this function:

its_device_handle its_get_device_handle (uns16 *device_id*)

```
108                                                         {
109
110 bit found = 0;
111     if (device_id == 0xffff) {
112         return ITS_DEVICE_NONE;
113     }
114     for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
115         if (its_devices[count].its_device_id == device_id) {
116             found = 1;
117             break;
118         }
119     }
120     if (found) {
121         return count;
122     } else {
123         return ITS_DEVICE_NONE;
124     }
125 }
```

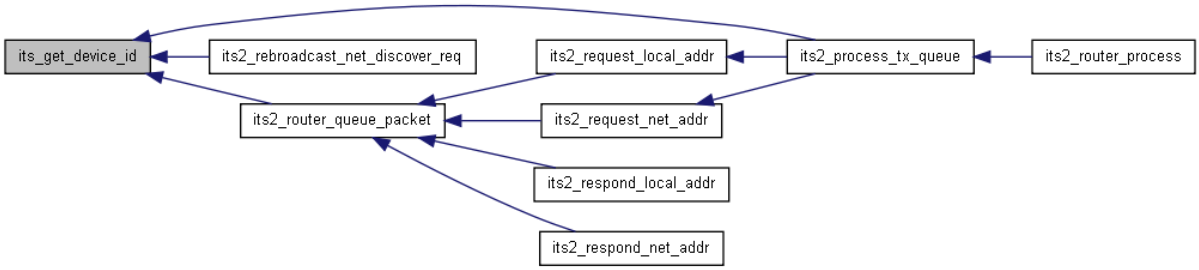
Here is the caller graph for this function:



uns16 its_get_device_id ()

```
89                              {
90      return its_device_id;
91  }
```

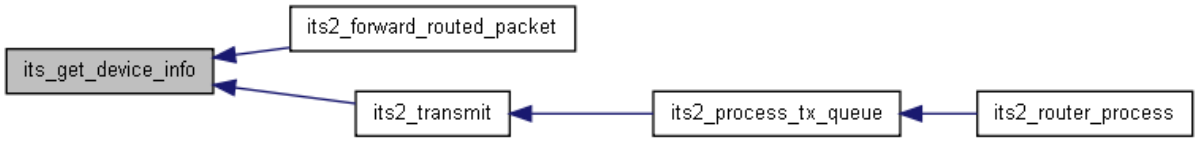
Here is the caller graph for this function:



its_device_info* its_get_device_info (uns8 *handle*)

```
100                                                 {
101     if (handle < ITS_MAX_KNOWN_DEVICES) {
102         return &its_devices[handle];
103     } else {
104         return NULL;
105     }
106 }
```

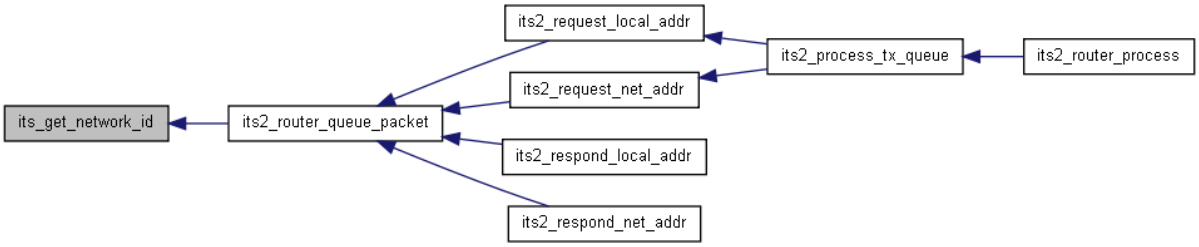
Here is the caller graph for this function:



uns16 its_get_network_id ()

```
81                              {
82      return its_network_id;
83  }
```

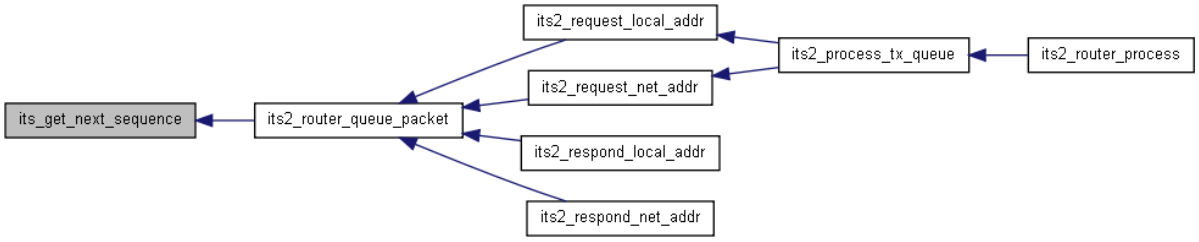
Here is the caller graph for this function:



uns8 its_get_next_sequence ()

```
72                                  {
73      return its_sequence++;
74  }
```

Here is the caller graph for this function:

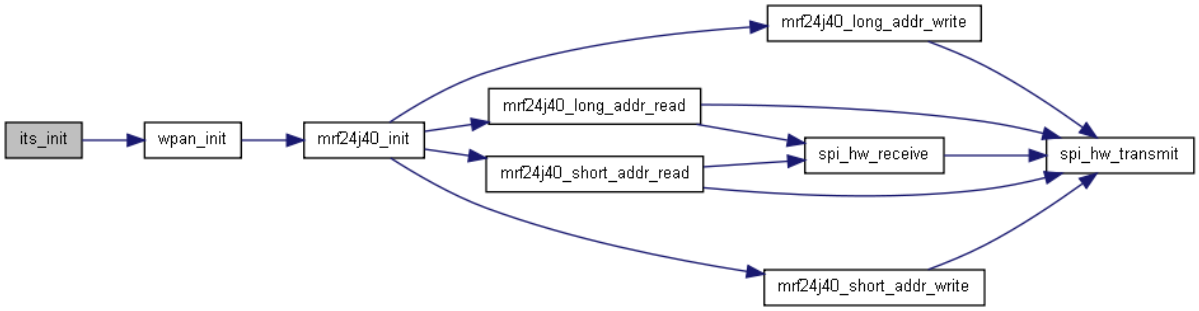


void its_init ()

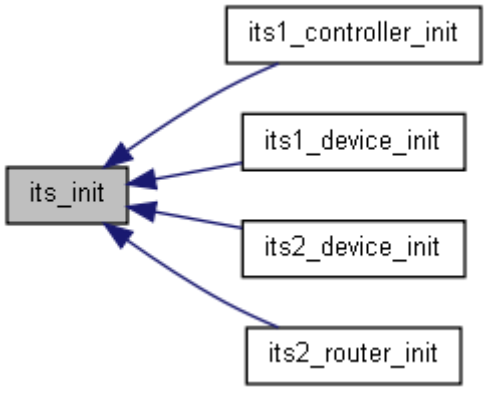
Call before using any ITS functionality

```
93                  {
94      for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
95          its_devices[count].its_device_id = 0xffff;    // 0xffff = blank entry
96      }
97      wpan_init();
98  }
```

Here is the call graph for this function:



Here is the caller graph for this function:

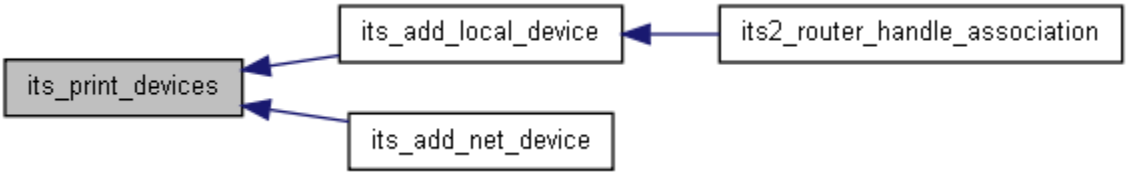


void its_print_devices ()

```
51                          {
```

```
52      debug_nl();
53      for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
54          if (its_devices[count].its_device_id != 0xffff) {    // 0xffff = blank entry
55              debug_str(" dev= ");
56              debug_int_hex_16bit(its_devices[count].its_device_id);
57              if (its_devices[count].addr.remote.remote_indicator != 0xffff) {
58                  debug_str(" pan= ");
59                  debug_int_hex_16bit(its_devices[count].addr.local.pan_id);
60                  debug_str(" sa= ");
61                  debug_int_hex_16bit(its_devices[count].addr.local.short_address);
62              } else {
63                  debug_str(" via= ");
64                  debug_int_hex_16bit(its_devices[count].addr.remote.prior_device_id);
65              }
66              debug_nl();
67
68          }
69      }
70  }
```

Here is the caller graph for this function:

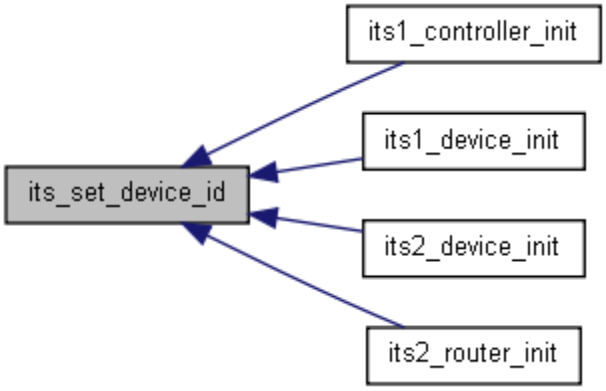


void its_set_device_id (uns16 *device_id*)

Set the ITS device ID before sending packets. See also: its_set_network_id(uns16 network_id);

```
85                                          {
86      its_device_id = device_id;
87  }
```

Here is the caller graph for this function:

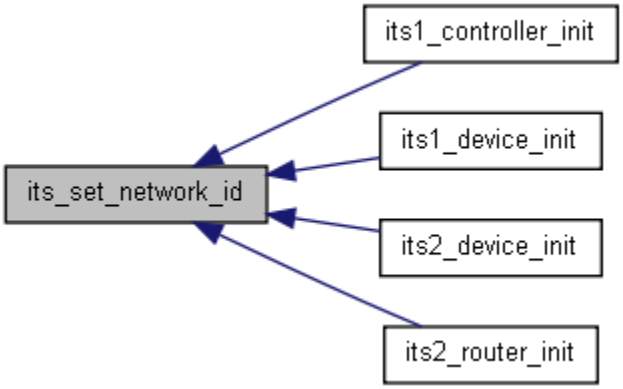


void its_set_network_id (uns16 *network_id*)

Set the ITS network ID before sending packets See also: its_set_device_id(uns16 device_id);

```
76                                          {
77      its_network_id = network_id;
78  }
```

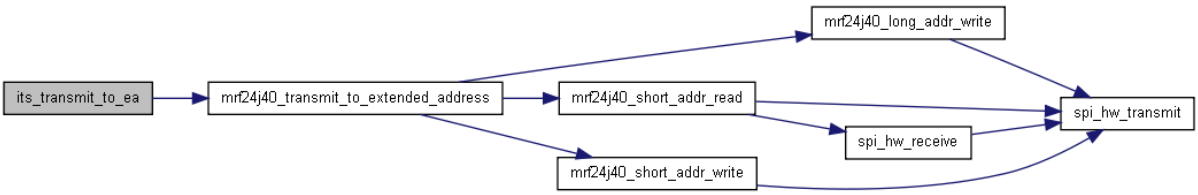
Here is the caller graph for this function:



void its_transmit_to_ea (uns8 * *dest_ea*, uns16 *dest_its_device_id*, uns8 *packet_type*, uns8 * *data*, uns8 *data_length*)

```
232
{
233
234 // must do this more efficiently when we have it working!!!!
235
236 uns8 buffer[20];
237
238 uns8 count;
239
240     // should do some checks on handle to see if its valid, but hey...
241
242     buffer[0] = 'I';
243     buffer[1] = 'T';
244     buffer[2] = 11;     // counted by hand from below
245     buffer[3] = packet_type;
246     buffer[4] = its_sequence++;
247     // network
248     buffer[5] = its_network_id & 0xff;  // lsb
249     buffer[6] = its_network_id >> 8;    // msb
250     // source
251     buffer[7] = its_device_id & 0xff;   // lsb
252     buffer[8] = its_device_id >> 8; // msb
253     // dest
254
255     buffer[9] = dest_its_device_id & 0xff;  // lsb
256     buffer[10] = dest_its_device_id >> 8;   // msb
257     // routing
258     // in mode 2 we would go and look up our routing table here
259     buffer[11] = 1; // max hop count
260     buffer[12] = 0; // hop count
261     buffer[13] = 0; // num routes is zero
262                     // so no routes list
263     buffer[14] = data_length;   // length of data
264     for (count = 0; count < data_length; count++) {
265         buffer[15+count] = data[count];
266     }   // copy data in
267     mrf24j40_transmit_to_extended_address(FRAME_TYPE_DATA, /* pan */ 0xffff,
268                                           dest_ea,  &buffer,
269                                           data_length + 15, MRF_NO_ACK);
270 }
```

Here is the call graph for this function:

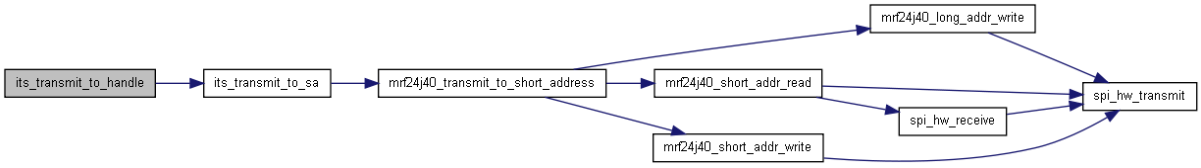


Here is the caller graph for this function:

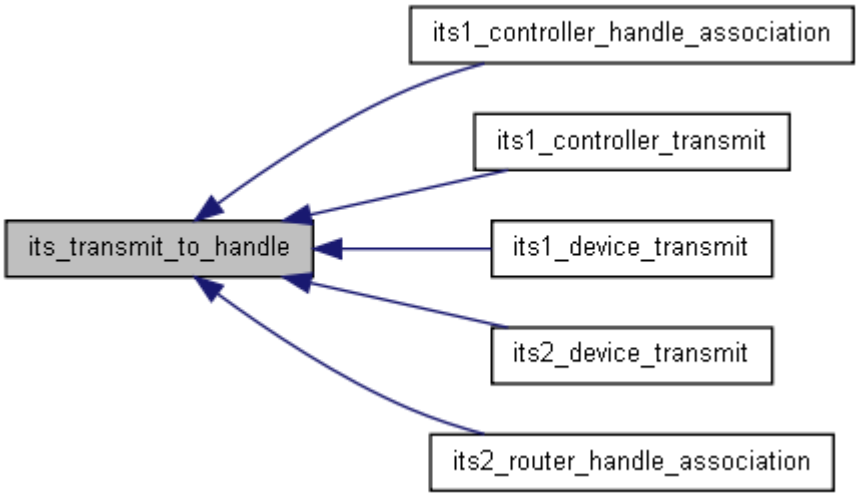
void its_transmit_to_handle (its_device_handle *handle*, uns8 *packet_type*, uns8 * *data*, uns8 *data_length*)

```
185
{
186
187     // !! should do some checks on handle to see if its valid, but hey...
188
189     its_transmit_to_sa(its_devices[handle].addr.local.pan_id,
190                        its_devices[handle].addr.local.short_address,
191                        its_devices[handle].its_device_id,
192                        packet_type, data, data_length);
193 }
```

Here is the call graph for this function:



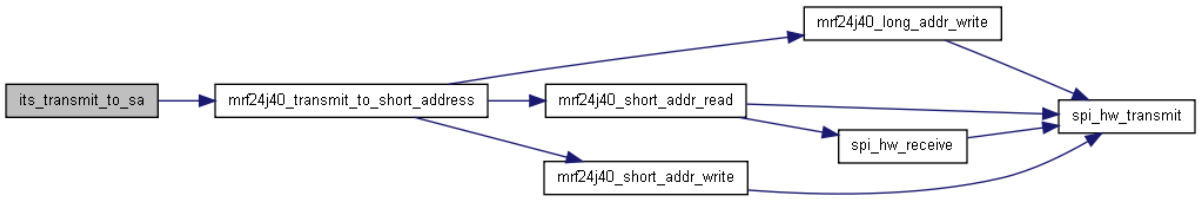
Here is the caller graph for this function:



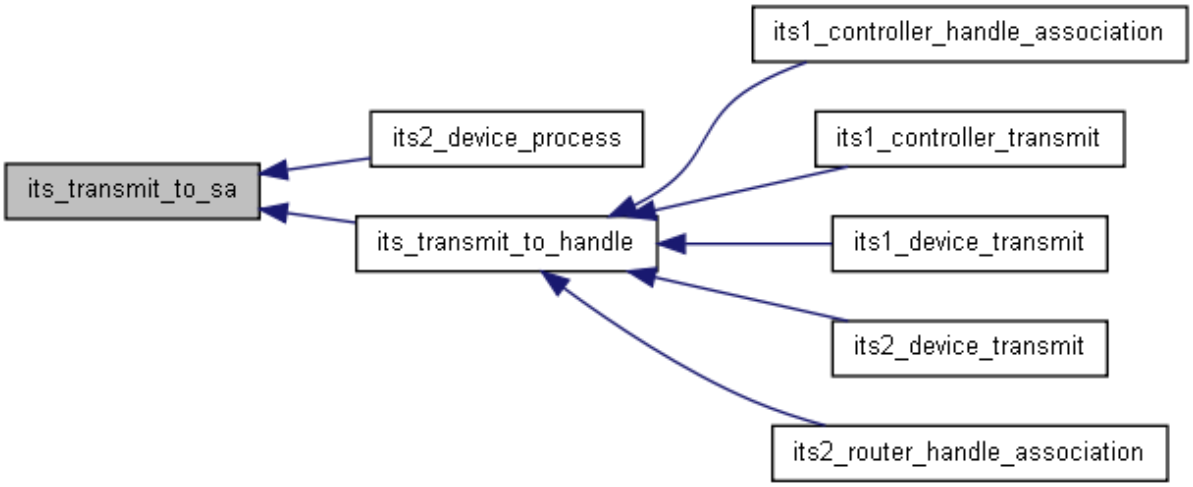
void its_transmit_to_sa (uns16 *dest_pan_id*, uns16 *dest_sa*, uns16 *dest_device_id*, uns8 *packet_type*, uns8 * *data*, uns8 *data_length*)

```
195
{
196
197 uns8 buffer[20];
198 uns16 temp;
199 uns8 count;
200
201     buffer[0] = 'I';
202     buffer[1] = 'T';
203     buffer[2] = 11;     // counted by hand from below
204     buffer[3] = packet_type;
205     buffer[4] = its_sequence++;
206     // network
207     buffer[5] = its_network_id & 0xff;  // lsb
208     buffer[6] = its_network_id >> 8;    // msb
209     // source
210     buffer[7] = its_device_id & 0xff;   // lsb
211     buffer[8] = its_device_id >> 8; // msb
212     // dest
213     temp = dest_device_id;
214     buffer[9] = temp & 0xff;    // lsb
215     buffer[10] = temp >> 8; // msb
216     // routing
217     // in mode 2 we would go and look up our routing table here
218     buffer[11] = 1; // max hop count
219     buffer[12] = 0; // hop count
220     buffer[13] = 0; // num routes is zero
221                     // so no routes list
222     buffer[14] = data_length;   // length of data
223     for (count = 0; count < data_length; count++) {
224         buffer[15+count] = data[count];
225     }   // copy data in
226     mrf24j40_transmit_to_short_address(FRAME_TYPE_DATA, dest_pan_id,
227                                        dest_sa, &buffer,
228                                        data_length + 15, MRF_ACK);
229 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



Variable Documentation

uns16 its_device_id

its_device_info its_devices[ITS_MAX_KNOWN_DEVICES]

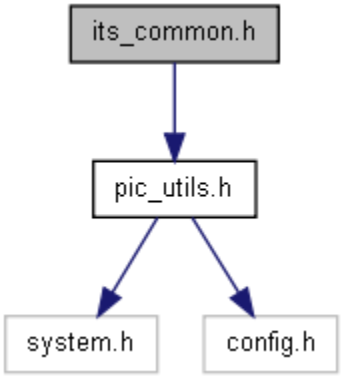
uns16 its_network_id

uns8 its_sequence = 0

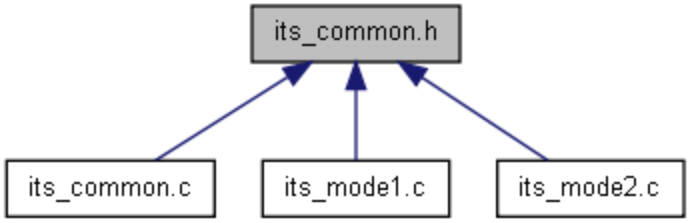
---

its_common.h File Reference

Include dependency graph for its_common.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- union its_address
- struct its_device_info
- struct local_address
- struct remote_address

Defines

- #define ITS_ACK 0x03
- #define ITS_APP_DATA 0x02
- #define ITS_ASSOC_REQ 0x00
- #define ITS_ASSOC_RES 0x01
- #define ITS_DEVICE_NONE 0xff
- #define ITS_ENDPOINT_DATA 0x0c
- #define ITS_ENDPOINT_REQ 0x0a
- #define ITS_ENDPOINT_RES 0x0b
- #define ITS_GENERIC_DATA 0x09
- #define ITS_LOCAL_DISCOVER_REQ 0x05
- #define ITS_LOCAL_DISCOVER_RES 0x06
- #define ITS_NET_DISCOVER_REQ 0x07
- #define ITS_NET_DISCOVER_RES 0x08
- #define ITS_PENDING_DATA_REQ 0x04
- #define ITS_ROUTE_FAILURE 0x0d

Typedefs

- typedef uns8 its_device_handle

Functions

- its_device_handle its_add_local_device (uns16 device_id, uns16 pan_id, uns16 short_address)
- its_device_handle its_add_net_device (uns16 device_id, uns16 previous_hop)
- its_device_handle its_get_device_handle (uns16 device_id)
- uns16 its_get_device_id ()
- its_device_info * its_get_device_info (uns8 handle)
- uns16 its_get_network_id ()
- *Retrieve the current network ID.* uns8 its_get_next_sequence ()
- void its_init ()
- *Initialise ITS and lower layers.* void its_print_devices ()
- *Print devices currently known to this one.* void its_set_device_id (uns16 device_id)
- *Set the ITS device ID.* void its_set_network_id (uns16 network_id)
- *Set the ITS network ID.* void its_transmit_to_ea (uns8 *dest_ea, uns16 dest_its_device_id, uns8 packet_type, uns8 *data, uns8 data_length)
- void its_transmit_to_handle (its_device_handle handle, uns8 packet_type, uns8 *data, uns8 data_length)

- void its_transmit_to_sa (uns16 dest_pan_id, uns16 dest_sa, uns16 dest_device_id, uns8 packet_type, uns8 *data, uns8 data_length)

Define Documentation

#define ITS_ACK 0x03

#define ITS_APP_DATA 0x02

#define ITS_ASSOC_REQ 0x00

#define ITS_ASSOC_RES 0x01

#define ITS_DEVICE_NONE 0xff

#define ITS_ENDPOINT_DATA 0x0c

#define ITS_ENDPOINT_REQ 0x0a

#define ITS_ENDPOINT_RES 0x0b

#define ITS_GENERIC_DATA 0x09

#define ITS_LOCAL_DISCOVER_REQ 0x05

#define ITS_LOCAL_DISCOVER_RES 0x06

#define ITS_NET_DISCOVER_REQ 0x07

#define ITS_NET_DISCOVER_RES 0x08

#define ITS_PENDING_DATA_REQ 0x04

#define ITS_ROUTE_FAILURE 0x0d

Typedef Documentation

typedef uns8 its_device_handle

Definition of its_device_handle

Function Documentation

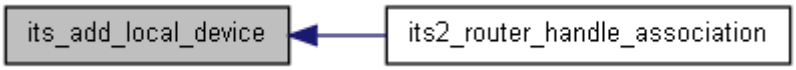
its_device_handle its_add_local_device (uns16 *device_id*, uns16 *pan_id*, uns16 *short_address*)

```
127
{
128
129 bit found_slot = 0;
130     debug_str("Adding device_id (");
131     debug_int_hex_16bit(device_id);
```

```
132     debug_str(" (pan_id ");
133     debug_int_hex_16bit(pan_id);
134     debug_str(" short addr ");
135     debug_int_hex_16bit(short_address);
136     debug_str(")\n");
137 
138     for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
139         if (its_devices[count].its_device_id == 0xffff) {
140             found_slot = 1;
141             break;
142         }
143     }
144     if (found_slot) {
145         its_devices[count].its_device_id = device_id;
146         its_devices[count].addr.local.pan_id = pan_id;
147         its_devices[count].addr.local.short_address = short_address;
148     } else {
149         count = ITS_DEVICE_NONE;
150     }
151 
152     its_print_devices();
153 
154     return count;
155 
156 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



its_device_handle its_add_net_device (uns16 *device_id*, uns16 *previous_hop*)

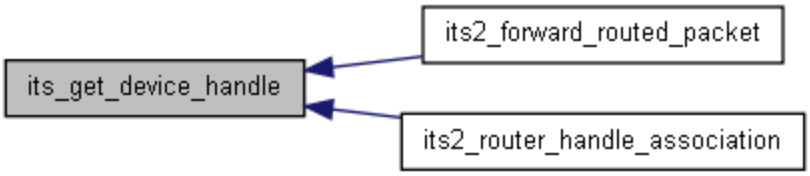
```
158                                                                         {
159 
160 bit found_slot = 0;
161     debug_str(" Add ");
162     debug_int_hex_16bit(device_id);
163     debug_str(" (via  ");
164     debug_int_hex_16bit(previous_hop);
165     debug_str(")\n");
166 
167     for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
168         if (its_devices[count].its_device_id == 0xffff) {
169             found_slot = 1;
170             break;
171         }
172     }
173     if (found_slot) {
174         its_devices[count].its_device_id = device_id;
175         its_devices[count].addr.remote.remote_indicator = 0xffff;
176         its_devices[count].addr.remote.prior_device_id = previous_hop;
177     } else {
178         count = ITS_DEVICE_NONE;
179     }
180     its_print_devices();
181     return count;
182 }
```

Here is the call graph for this function:

its_device_handle its_get_device_handle (uns16 *device_id*)

```
108                                                     {
109
110 bit found = 0;
111     if (device_id == 0xffff) {
112         return ITS_DEVICE_NONE;
113     }
114     for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
115         if (its_devices[count].its_device_id == device_id) {
116             found = 1;
117             break;
118         }
119     }
120     if (found) {
121         return count;
122     } else {
123         return ITS_DEVICE_NONE;
124     }
125 }
```

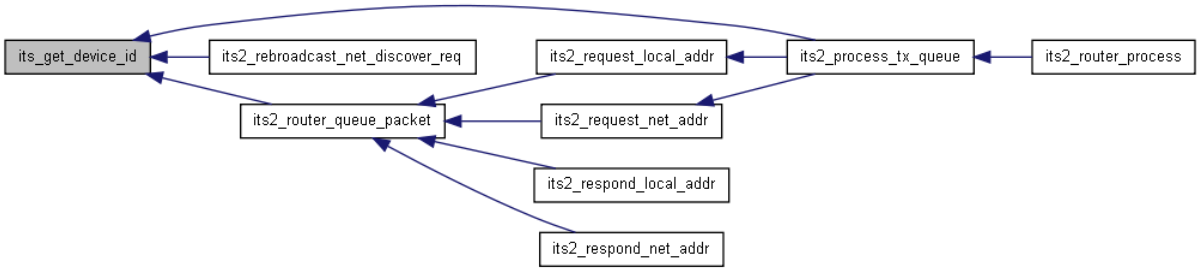
Here is the caller graph for this function:



uns16 its_get_device_id ()

```
89                              {
90     return its_device_id;
91 }
```

Here is the caller graph for this function:

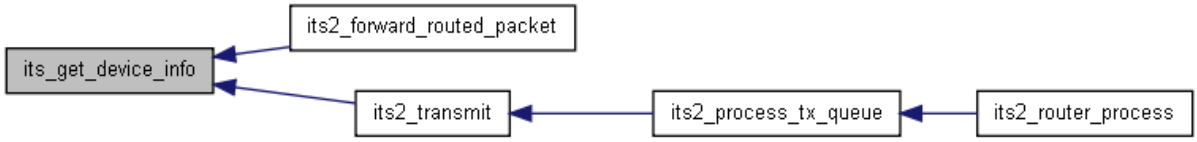


its_device_info* its_get_device_info (uns8 *handle*)

```
100                                                     {
101     if (handle < ITS_MAX_KNOWN_DEVICES) {
102         return &its_devices[handle];
```

```
103     } else {
104         return NULL;
105     }
106 }
```

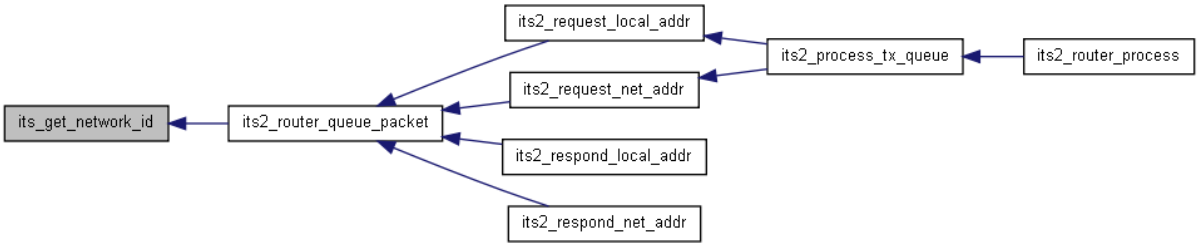
Here is the caller graph for this function:



uns16 its_get_network_id ()

```
81                              {
82     return its_network_id;
83 }
```

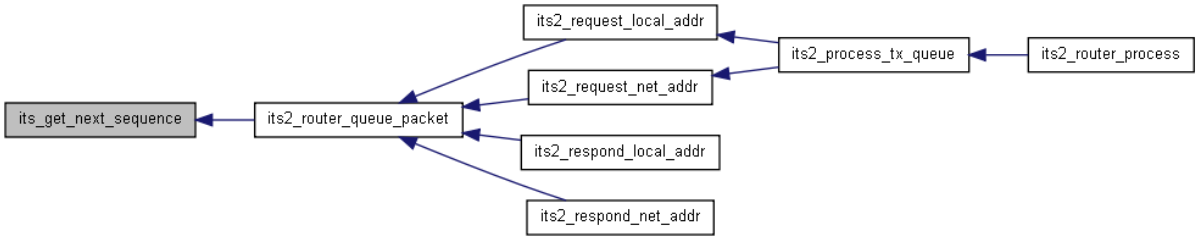
Here is the caller graph for this function:



uns8 its_get_next_sequence ()

```
72                               {
73     return its_sequence++;
74 }
```

Here is the caller graph for this function:

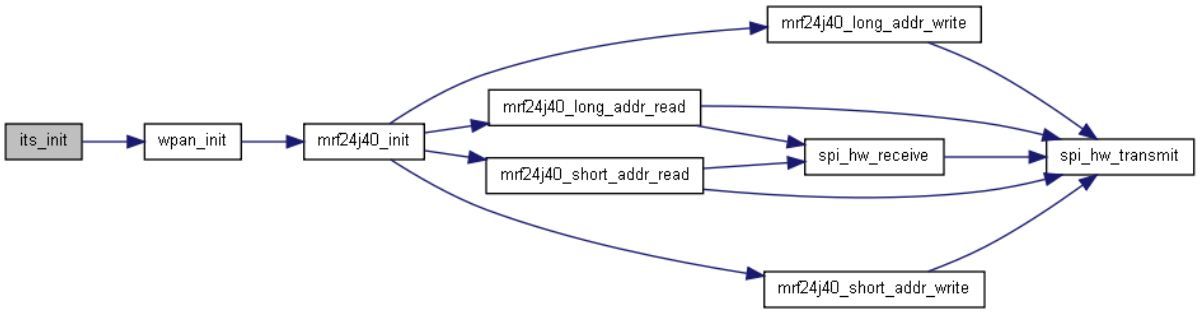


void its_init ()

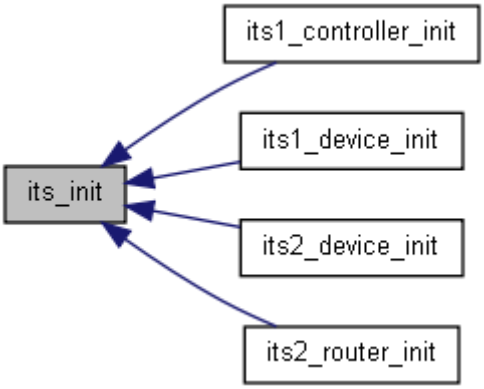
Call before using any ITS functionality

```
93                  {
94     for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
95         its_devices[count].its_device_id = 0xffff;   // 0xffff = blank entry
96     }
97     wpan_init();
98 }
```

Here is the call graph for this function:



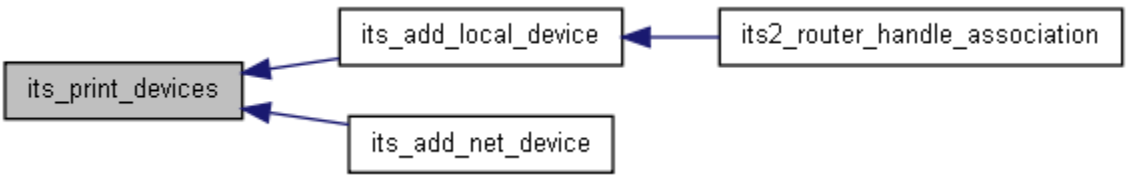
Here is the caller graph for this function:



void its_print_devices ()

```
51                          {
52     debug_nl();
53     for (uns8 count=0; count < ITS_MAX_KNOWN_DEVICES; count++) {
54         if (its_devices[count].its_device_id != 0xffff) {    // 0xffff = blank entry
55             debug_str(" dev= ");
56             debug_int_hex_16bit(its_devices[count].its_device_id);
57             if (its_devices[count].addr.remote.remote_indicator != 0xffff) {
58                 debug_str(" pan= ");
59                 debug_int_hex_16bit(its_devices[count].addr.local.pan_id);
60                 debug_str(" sa= ");
61                 debug_int_hex_16bit(its_devices[count].addr.local.short_address);
62             } else {
63                 debug_str(" via= ");
64                 debug_int_hex_16bit(its_devices[count].addr.remote.prior_device_id);
65             }
66             debug_nl();
67 
68         }
69     }
70 }
```

Here is the caller graph for this function:

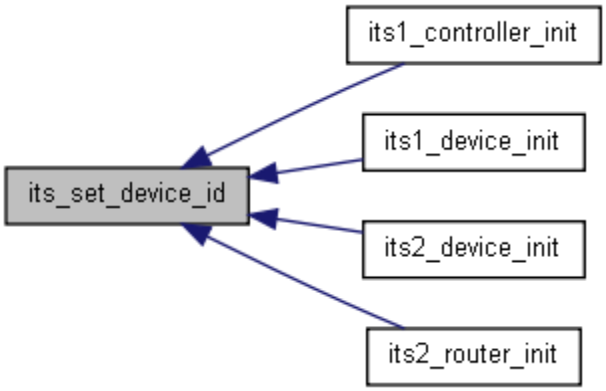


void its_set_device_id (uns16 *device_id*)

Set the ITS device ID before sending packets. See also: its_set_network_id(uns16 network_id);

```
85                                          {
86     its_device_id = device_id;
87 }
```

Here is the caller graph for this function:

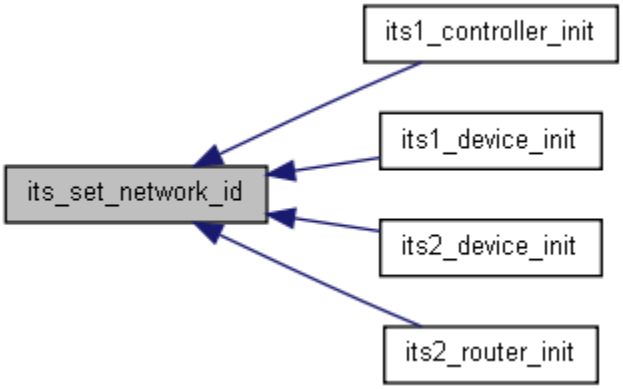


void its_set_network_id (uns16 *network_id*)

Set the ITS network ID before sending packets See also: its_set_device_id(uns16 device_id);

```
76                                          {
77     its_network_id = network_id;
78 }
```

Here is the caller graph for this function:

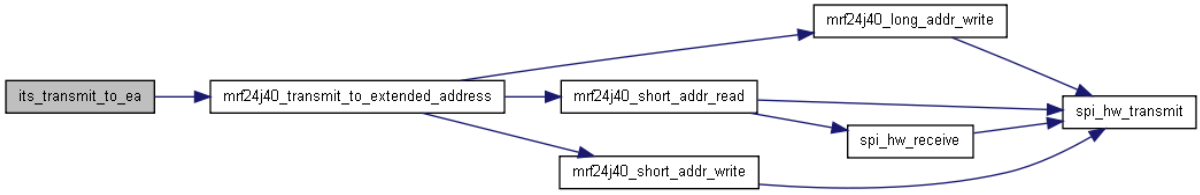


void its_transmit_to_ea (uns8 * *dest_ea*, uns16 *dest_its_device_id*, uns8 *packet_type*, uns8 * *data*, uns8 *data_length*)

```
232
{
233
234 // must do this more efficiently when we have it working!!!!
235
236 uns8 buffer[20];
237
```

```
238 uns8 count;
239
240     // should do some checks on handle to see if its valid, but hey...
241
242     buffer[0] = 'I';
243     buffer[1] = 'T';
244     buffer[2] = 11;     // counted by hand from below
245     buffer[3] = packet_type;
246     buffer[4] = its_sequence++;
247     // network
248     buffer[5] = its_network_id & 0xff;  // lsb
249     buffer[6] = its_network_id >> 8;    // msb
250     // source
251     buffer[7] = its_device_id & 0xff;   // lsb
252     buffer[8] = its_device_id >> 8; // msb
253     // dest
254
255     buffer[9] = dest_its_device_id & 0xff;  // lsb
256     buffer[10] = dest_its_device_id >> 8;   // msb
257     // routing
258     // in mode 2 we would go and look up our routing table here
259     buffer[11] = 1; // max hop count
260     buffer[12] = 0; // hop count
261     buffer[13] = 0; // num routes is zero
262                     // so no routes list
263     buffer[14] = data_length;   // length of data
264     for (count = 0; count < data_length; count++) {
265         buffer[15+count] = data[count];
266     }   // copy data in
267     mrf24j40_transmit_to_extended_address(FRAME_TYPE_DATA, /* pan */ 0xffff,
268                                           dest_ea,  &buffer,
269                                           data_length + 15, MRF_NO_ACK);
270 }
```

Here is the call graph for this function:

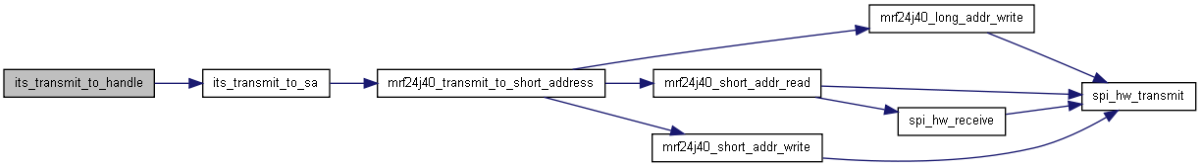


Here is the caller graph for this function:

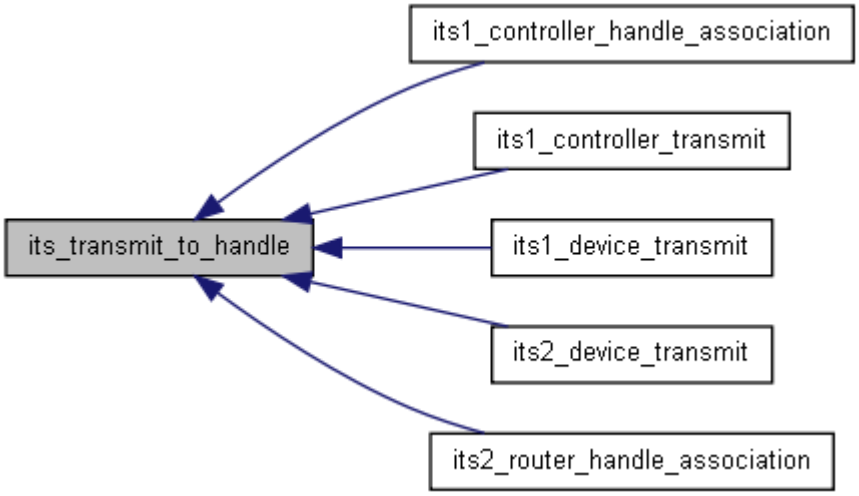
void its_transmit_to_handle (its_device_handle *handle*, uns8 *packet_type*, uns8 * *data*, uns8 *data_length*)

```
185
{
186
187     // !! should do some checks on handle to see if its valid, but hey...
188
189     its_transmit_to_sa(its_devices[handle].addr.local.pan_id,
190                        its_devices[handle].addr.local.short_address,
191                        its_devices[handle].its_device_id,
192                        packet_type, data, data_length);
193 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

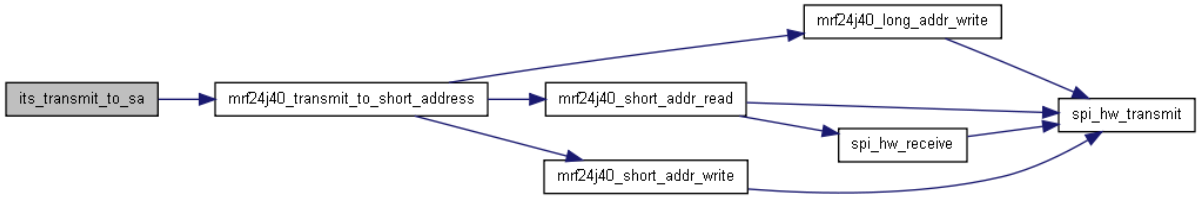


void its_transmit_to_sa (uns16 *dest_pan_id*, uns16 *dest_sa*, uns16 *dest_device_id*, uns8 *packet_type*, uns8 * *data*, uns8 *data_length*)

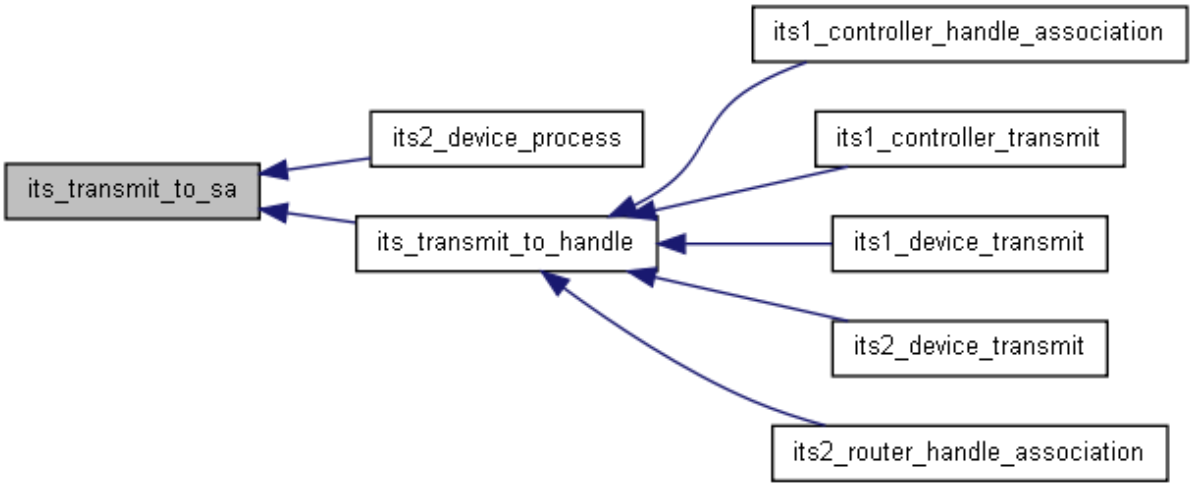
```
195
{
196
197 uns8 buffer[20];
198 uns16 temp;
199 uns8 count;
200
201     buffer[0] = 'I';
202     buffer[1] = 'T';
203     buffer[2] = 11;     // counted by hand from below
204     buffer[3] = packet_type;
205     buffer[4] = its_sequence++;
206     // network
207     buffer[5] = its_network_id & 0xff;  // lsb
208     buffer[6] = its_network_id >> 8;    // msb
209     // source
210     buffer[7] = its_device_id & 0xff;   // lsb
211     buffer[8] = its_device_id >> 8; // msb
212     // dest
213     temp = dest_device_id;
214     buffer[9] = temp & 0xff;    // lsb
215     buffer[10] = temp >> 8; // msb
216     // routing
217     // in mode 2 we would go and look up our routing table here
218     buffer[11] = 1; // max hop count
219     buffer[12] = 0; // hop count
220     buffer[13] = 0; // num routes is zero
221                     // so no routes list
222     buffer[14] = data_length;   // length of data
223     for (count = 0; count < data_length; count++) {
224         buffer[15+count] = data[count];
225     }   // copy data in
226     mrf24j40_transmit_to_short_address(FRAME_TYPE_DATA, dest_pan_id,
```

```
227                                             dest_sa, &buffer,
228                                             data_length + 15, MRF_ACK);
229 }
```

Here is the call graph for this function:



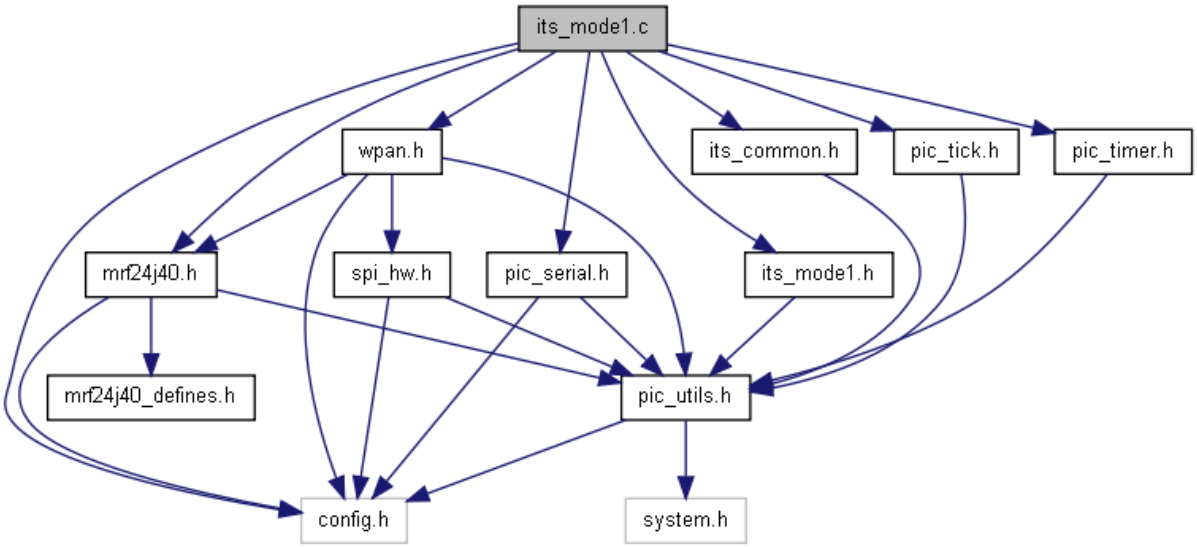
Here is the caller graph for this function:



---

its_mode1.c File Reference

Include dependency graph for its_mode1.c:



Functions

- its1_result its1_controller_handle_association (uns16 pan_id, uns16 its_device_id)
- uns8 its1_controller_init (uns16 my_device_id, uns16 network_id)
- void its1_controller_process ()
- void its1_controller_receive_callback (uns16 device_id, uns8 *data, uns8 data_length)
- its1_result its1_controller_transmit (uns16 device_id, uns8 *data, uns8 data_length)
- its1_result its1_device_init (uns16 my_device_id, uns16 network_id)
- void its1_device_process ()
- its1_result its1_device_transmit (uns8 *data, uns8 data_length)
- its1_result its1_find_controller ()
- void its1_setup_io ()
- void wpan_data_received_callback (wpan_address *addr, uns8 *data, uns8 data_size)

Variables

- uns8 channel
- uns8 controller_handle
- its1_state state = STATE_STARTUP
- uns16 tick_marker

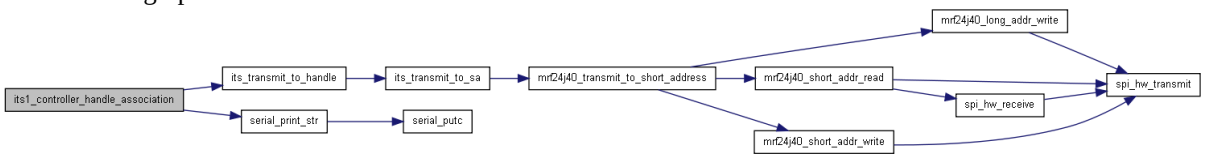
Function Documentation

its1_result its1_controller_handle_association (uns16 *pan_id*, uns16 *its_device_id*)

```
62                                                                                  {
63     // see if this device is in our list, if not, add it
64 its_device_handle device_handle;
65
66     device_handle = its_get_device(its_device_id);
67
68     if (device_handle == ITS_DEVICE_NONE) {
69         // in mode 1 we assume that
70         device_handle = its_add_device(/* device */ its_device_id, pan_id, /* short addr */
its_device_id);
71         if (device_handle == ITS_DEVICE_NONE) {
```

```
72              // panic!
73              serial_print_str("Too many devices! association failed");
74              return RESULT_FAILED;
75          } else {
76              // send association response
77              its_transmit_to_handle(device_handle, ITS_ASSOC_RES, /* nil data */ 0, /* zero
length */ 0);
78              return RESULT_SUCCESSFUL;
79              // although strictly speaking we should wait for the ack...
80          }
81
82      }
83  }
```

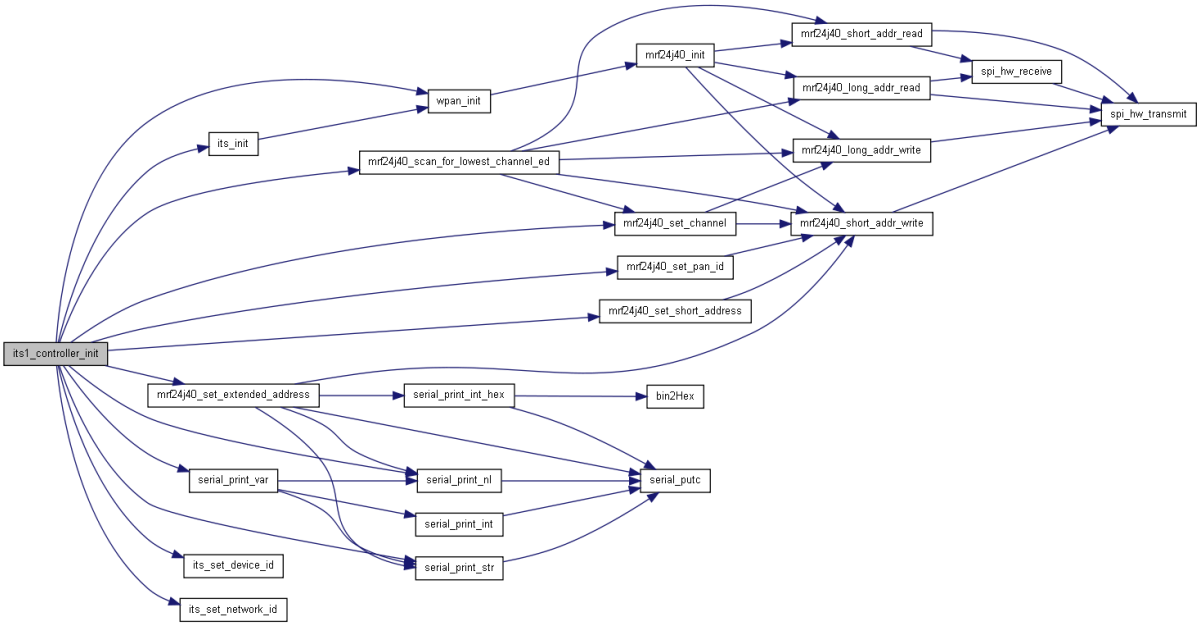
Here is the call graph for this function:



uns8 its1_controller_init (uns16 *my_device_id*, uns16 *network_id*)

```
235                                                                 {
236
237 uns8 lowest_channel_ed;
238 uns8 my_ed[8] = ITS_EA;
239
240     its_init();
241     wpan_init();
242
243     its_set_device_id(my_device_id);
244     its_set_network_id(network_id); // network id = controller id = pan id in mode 1
245
246     mrf24j40_set_pan_id(network_id);
247     mrf24j40_set_short_address(my_device_id);   // same as device id
248     mrf24j40_set_extended_address(&my_ed);
249
250     serial_print_str("Controller startup\n");
251
252     // Find a channel to play on
253     serial_print_str("Choosing channel\n");
254
255     lowest_channel_ed = mrf24j40_scan_for_lowest_channel_ed();
256
257     serial_print_var("Chosing channel", lowest_channel_ed);
258     serial_print_nl();
259
260     // Now set ourselves up on this channel
261     mrf24j40_set_channel(lowest_channel_ed);
262
263 }
```

Here is the call graph for this function:



void its1_controller_process ()

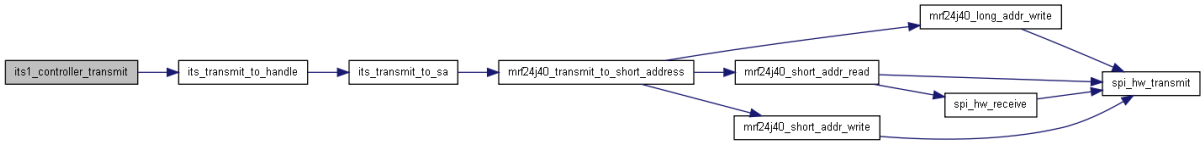
```
279                                 {
280 }
```

void its1_controller_receive_callback (uns16 *device_id*, uns8 * *data*, uns8 *data_length*)

its1_result its1_controller_transmit (uns16 *device_id*, uns8 * *data*, uns8 *data_length*)

```
265                                                                                                 {
266
267 its_device_handle device_handle;
268
269     device_handle = its_get_device(device_id);
270
271     its_transmit_to_handle(device_handle,  ITS_GENERIC_DATA, data, data_length);
272
273
274 }
```

Here is the call graph for this function:

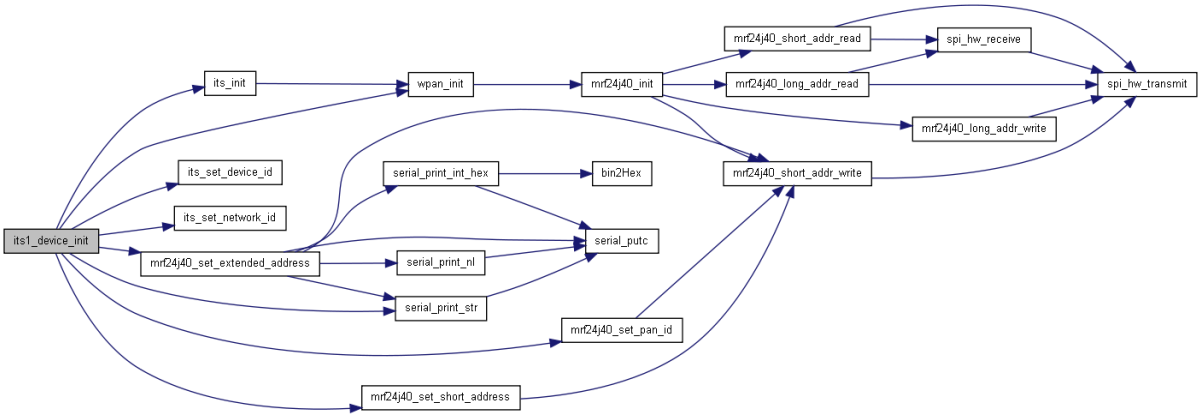


its1_result its1_device_init (uns16 *my_device_id*, uns16 *network_id*)

```
289                                                                 {
290
291 // ITS_EA is set in config.h
292
```

```
293 uns8 my_ea[8] = ITS_EA;
294
295     serial_print_str("Device startup\n");
296
297     its_init();
298     wpan_init();
299
300     its_set_device_id(my_device_id);
301     its_set_network_id(network_id); // network id = controller id = pan id in mode 1
302
303     mrf24j40_set_pan_id(network_id);
304     mrf24j40_set_short_address(my_device_id);   // same as device id
305     mrf24j40_set_extended_address(&my_ea);
306
307     state = STATE_UNASSOCIATED;
308 }
```

Here is the call graph for this function:

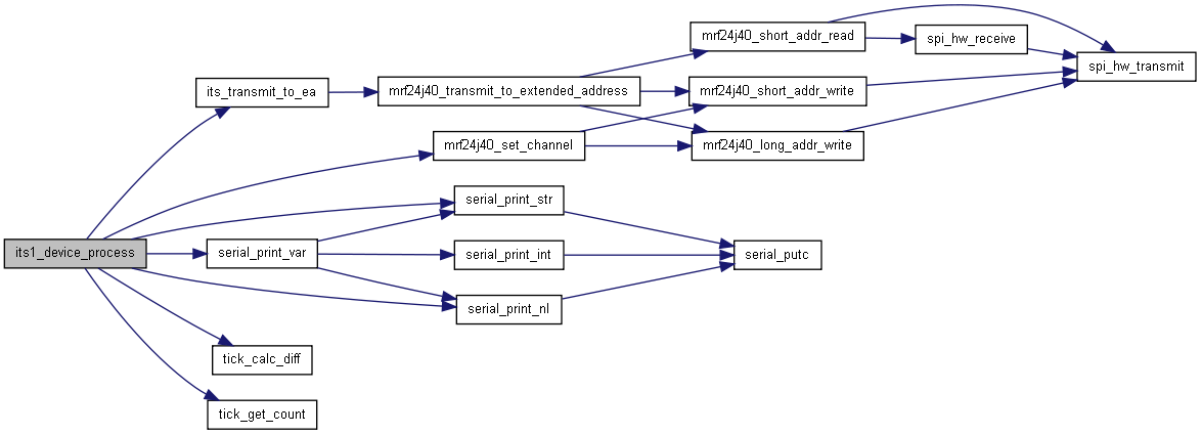


void its1_device_process ()

```
316                             {
317
318 uns16 test_tick;
319 uns8 controller_ea[8] = CONTROLLER_EA;
320
321     // are we searching?
322     // has time limit expired?
323     // go on to next channel
324     // send its association request to everyone
325     if (state == STATE_SEARCHING) {
326         test_tick = tick_get_count();
327         if (tick_calc_diff(tick_marker, test_tick) >= 250) {    // 250 = 1/4 second
328             tick_marker = test_tick;
329             if (channel == 0) {
330                 channel = MRF_FIRST_CHANNEL;
331             } else {
332                 if (channel == MRF_LAST_CHANNEL) {
333                     // Didn't find the controller
334                     state = STATE_UNASSOCIATED;
335                     serial_print_str("Didn't find controller\n");
336                 } else {
337                     channel++;
338                 }
339             }
340             // change channel
341             serial_print_var("Trying channel ", channel);
342             serial_print_nl();
343             mrf24j40_set_channel(channel);
344             // send broadcast association request
```

```
345             its_transmit_to_ea(&controller_ea, 0xffff, ITS_ASSOC_REQ, /* nil data */ 0, /*
zero length */ 0);
346         } // timer for another channel
347     } // we're searching
348
349 }
```

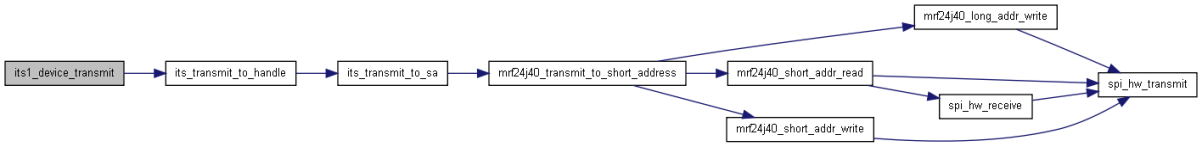
Here is the call graph for this function:



its1_result its1_device_transmit (uns8 * *data*, uns8 *data_length*)

```
352                                                                 {
353
354 // We want to transmit to our controller
355         its_transmit_to_handle(controller_handle,  ITS_GENERIC_DATA, data, data_length);
356 }
```

Here is the call graph for this function:



its1_result its1_find_controller ()

```
310                                 {
311     state = STATE_SEARCHING;
312     tick_marker = tick_get_count();
313     channel = 0;
314 }
```

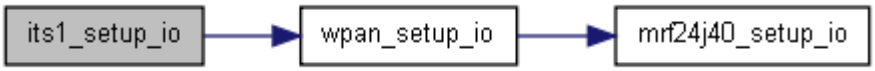
Here is the call graph for this function:

void its1_setup_io ()

```
57                      {
```

```
58      wpan_setup_io();
59  }
```

Here is the call graph for this function:

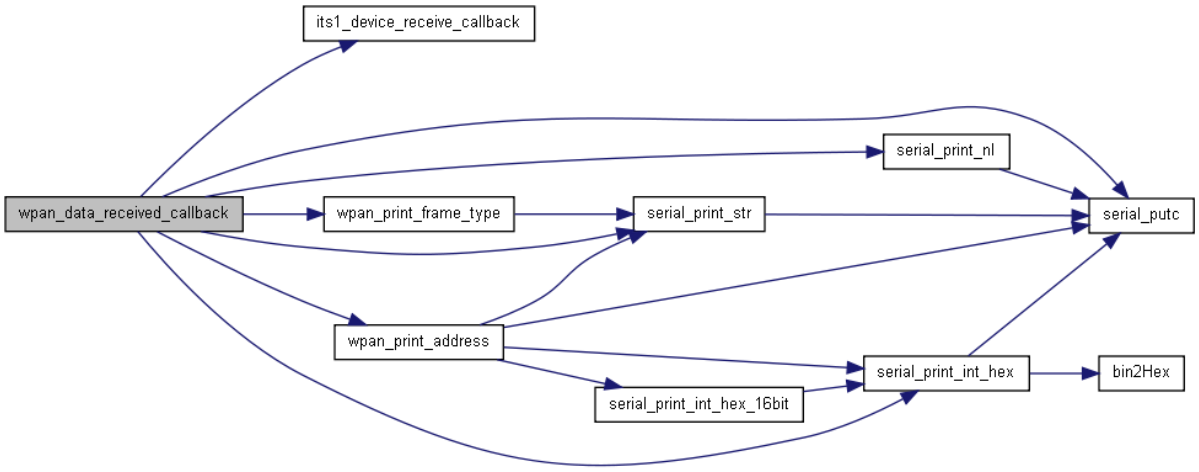


void wpan_data_received_callback (wpan_address * *addr*, uns8 * *data*, uns8 *data_size*)

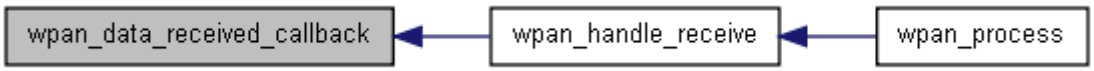
```
155                                                                                                 {
156
157
158
159 uns8 count;
160     wpan_print_address(addr);
161     wpan_print_frame_type(frame_type);
162     serial_print_str(" Data: ");
163     for (count = 0; count < data_size; count++) {
164         serial_print_int_hex(data[count]);
165         serial_putc(' ');
166         if ((data[count] > 31) && (data[count] < 127)) {
167             serial_putc(data[count]);
168         } else {
169             serial_putc('?');
170         }
171         serial_putc(' ');
172     }
173     serial_print_nl();
174
175
176
177
178
179     // check frame type
180     if ((data_size > 2) && (data[0] == 'I') && (data[1] == 'T')) {
181         serial_print_str("ITS Packet received\n");
182         uns8 length_header = data[2];
183         uns8 length_data   = data[3+length_header+1];
184         uns8 data_start    = 3+length_header+2;
185         // okay, but is it an its association request?
186         if (data[3] == ITS_ASSOC_RES) {
187             serial_print_str("Received association response\n");
188             if (state == STATE_SEARCHING) {
189                 serial_print_str("Changing state to associated\n");
190                 uns16 controller_device_id = data[8]; // device id of controller
191                 controller_device_id <<= 8;
192                 controller_device_id += data[7];
193
194                 uns8 device_handle = its_get_device(controller_device_id);
195
196                 if (device_handle == ITS_DEVICE_NONE) {
197
198                     device_handle = its_add_device(/* device */ controller_device_id,
addr->source_pan_id, controller_device_id);
199                     if (device_handle == ITS_DEVICE_NONE) {
200                         // panic!
201                         serial_print_str("Too many devices! association failed");
202                         state = STATE_UNASSOCIATED;
203                     } else {
204                         controller_handle = device_handle;
205                         state = STATE_ASSOCIATED;   // stop searching
206                     }
207                 } else { // it's already in the table
208                     state = STATE_ASSOCIATED;
209                 }
210
```

```
211             }
212         }   // association response
213         else if (data[3] == ITS_GENERIC_DATA) {
214             // is it for us?
215             // who cares right now!
216             // todo...
217             serial_print_str("Generic data pkt\n");
218             // just issue the callback
219             uns16 device_id = data[8];  // msb
220             device_id  <<= 8;
221             device_id += data[7];   // lsb
222             its1_device_receive_callback(&data[data_start], data_size - data_start);
223 
224         }   // GENERIC DATA
225     }
226 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



Variable Documentation

uns8 channel

uns8 controller_handle

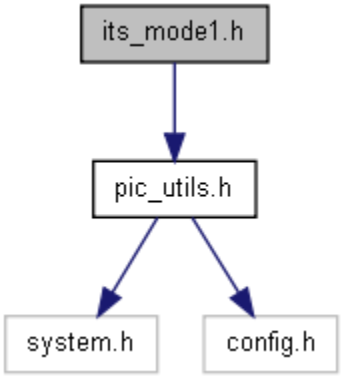
its1_state state = STATE_STARTUP

uns16 tick_marker

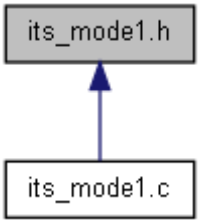
---

its_mode1.h File Reference

ITS networking mode 1.
Include dependency graph for its_mode1.h:



This graph shows which files directly or indirectly include this file:



Typedefs

- typedef enum _its1_result its1_result
- typedef enum _its1_state its1_state

Enumerations

- enum _its1_result { RESULT_SUCCESSFUL, RESULT_FAILED }
- enum _its1_state { STATE_STARTUP, STATE_RUNNING, STATE_SEARCHING, STATE_ASSOCIATED, STATE_UNASSOCIATED }

Functions

- its1_result its1_controller_handle_association (uns16 pan_id, uns16 its_device_id)
- uns8 its1_controller_init (uns16 my_device_id, uns16 network_id)
- void its1_controller_process ()
- void its1_controller_receive_callback (uns16 device_id, uns8 *data, uns8 data_length)
- its1_result its1_controller_transmit (uns16 device_id, uns8 *data, uns8 data_length)
- its1_result its1_device_init (uns16 my_device_id, uns16 network_id)
- void its1_device_process ()
- void its1_device_receive_callback (uns8 *data, uns8 data_length)
- its1_result its1_device_transmit (uns8 *data, uns8 data_length)
- its1_result its1_find_controller ()
- void its1_setup_io ()

Variables

- its1_state state

Detailed Description

Typedef Documentation

typedef enum _its1_result its1_result

typedef enum _its1_state its1_state

Enumeration Type Documentation

enum _its1_result

Enumerator:

RESULT_SUCCESSFUL
RESULT_FAILED

```
58 { RESULT_SUCCESSFUL, RESULT_FAILED };
```

enum _its1_state

Enumerator:

STATE_STARTUP
STATE_RUNNING
STATE_SEARCHING
STATE_ASSOCIATED
STATE_UNASSOCIATED

```
61                  { STATE_STARTUP, STATE_RUNNING, STATE_SEARCHING, STATE_ASSOCIATED,
62                    STATE_UNASSOCIATED };
```

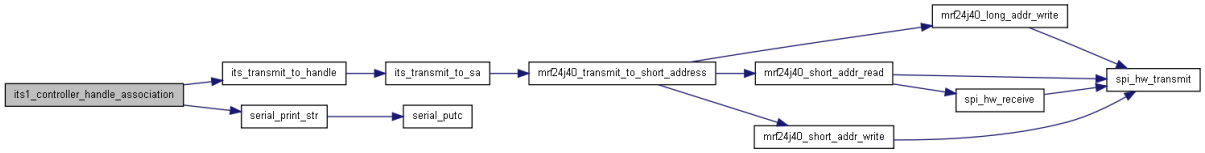
Function Documentation

its1_result its1_controller_handle_association (uns16 *pan_id*, uns16 *its_device_id*)

```
62                                                                                  {
63     // see if this device is in our list, if not, add it
64 its_device_handle device_handle;
65
66     device_handle = its_get_device(its_device_id);
67
68     if (device_handle == ITS_DEVICE_NONE) {
69         // in mode 1 we assume that
70         device_handle = its_add_device(/* device */ its_device_id, pan_id, /* short addr */
its_device_id);
71         if (device_handle == ITS_DEVICE_NONE) {
72             // panic!
73             serial_print_str("Too many devices! association failed");
74             return RESULT_FAILED;
```

```
75          } else {
76              // send association response
77              its transmit to handle(device handle, ITS ASSOC RES, /* nil data */ 0, /* zero
length */ 0);
78              return RESULT SUCCESSFUL;
79              // although strictly speaking we should wait for the ack...
80          }
81
82      }
83 }
```

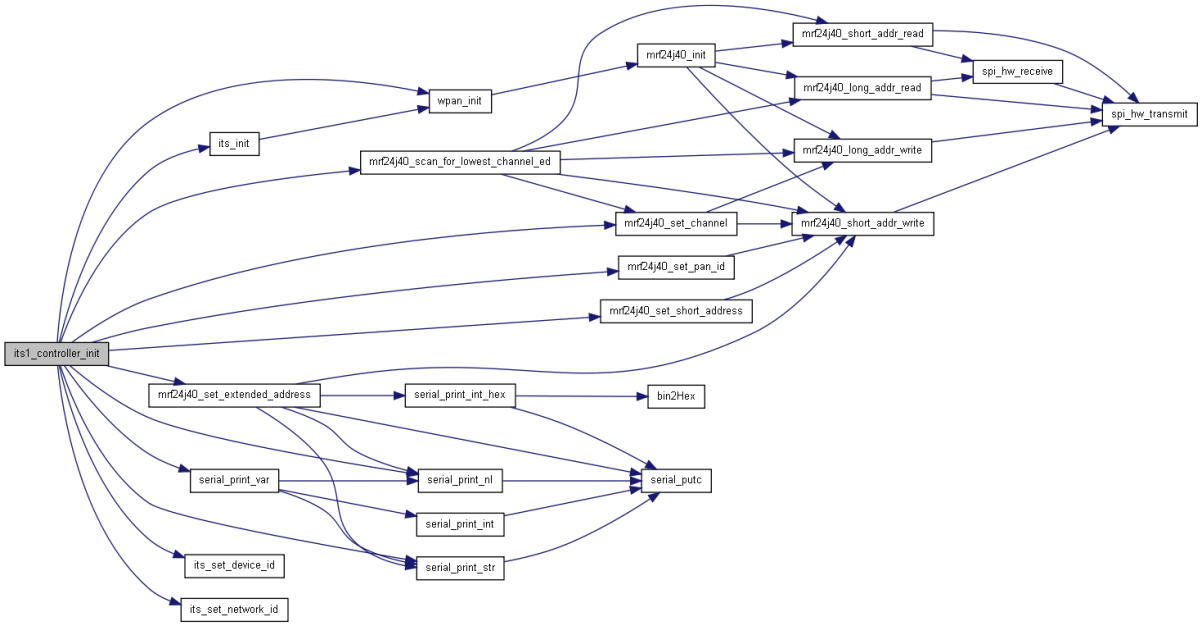
Here is the call graph for this function:



uns8 its1_controller_init (uns16 *my_device_id*, uns16 *network_id*)

```
235                                                                 {
236
237 uns8 lowest_channel_ed;
238 uns8 my_ed[8] = ITS_EA;
239
240     its_init();
241     wpan_init();
242
243     its_set_device_id(my_device_id);
244     its_set_network_id(network_id); // network id = controller id = pan id in mode 1
245
246     mrf24j40_set_pan_id(network_id);
247     mrf24j40_set_short_address(my_device_id);   // same as device id
248     mrf24j40_set_extended_address(&my_ed);
249
250     serial_print_str("Controller startup\n");
251
252     // Find a channel to play on
253     serial_print_str("Choosing channel\n");
254
255     lowest_channel_ed = mrf24j40_scan_for_lowest_channel_ed();
256
257     serial_print_var("Chosing channel", lowest_channel_ed);
258     serial_print_nl();
259
260     // Now set ourselves up on this channel
261     mrf24j40_set_channel(lowest_channel_ed);
262
263 }
```

Here is the call graph for this function:



void its1_controller_process ()

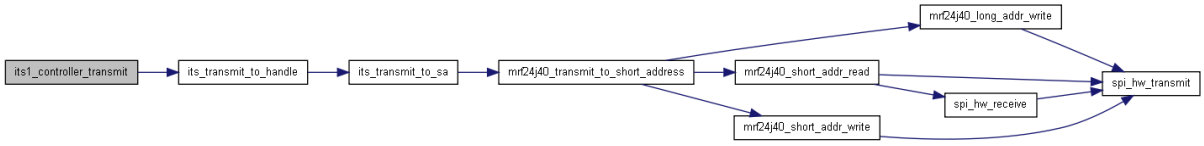
```
279                                 {
280 }
```

void its1_controller_receive_callback (uns16 *device_id*, uns8 * *data*, uns8 *data_length*)

its1_result its1_controller_transmit (uns16 *device_id*, uns8 * *data*, uns8 *data_length*)

```
265                                                                                                 {
266
267 its_device_handle device_handle;
268
269     device_handle = its_get_device(device_id);
270
271     its_transmit_to_handle(device_handle,  ITS_GENERIC_DATA, data, data_length);
272
273
274 }
```

Here is the call graph for this function:

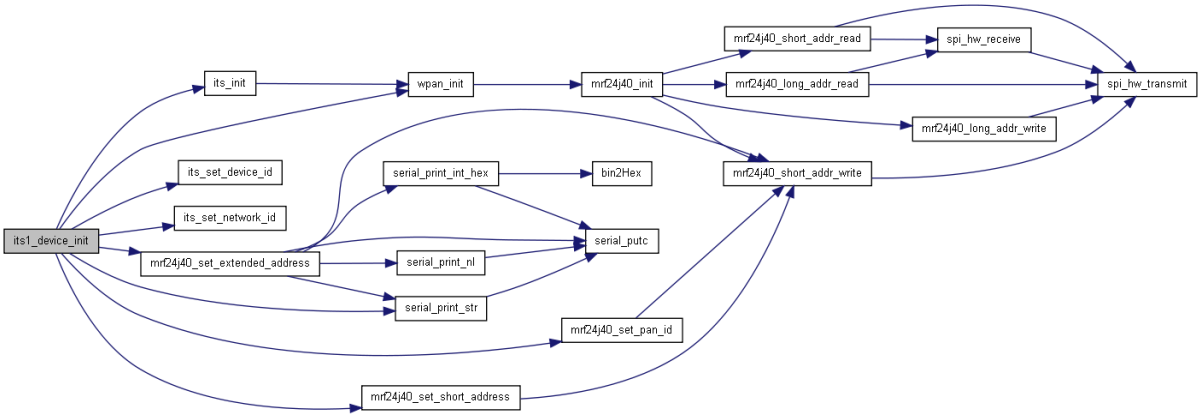


its1_result its1_device_init (uns16 *my_device_id*, uns16 *network_id*)

```
289                                                                 {
290
291 // ITS_EA is set in config.h
292
```

```
293 uns8 my_ea[8] = ITS_EA;
294
295     serial_print_str("Device startup\n");
296
297     its_init();
298     wpan_init();
299
300     its_set_device_id(my_device_id);
301     its_set_network_id(network_id); // network id = controller id = pan id in mode 1
302
303     mrf24j40_set_pan_id(network_id);
304     mrf24j40_set_short_address(my_device_id);   // same as device id
305     mrf24j40_set_extended_address(&my_ea);
306
307     state = STATE_UNASSOCIATED;
308 }
```

Here is the call graph for this function:

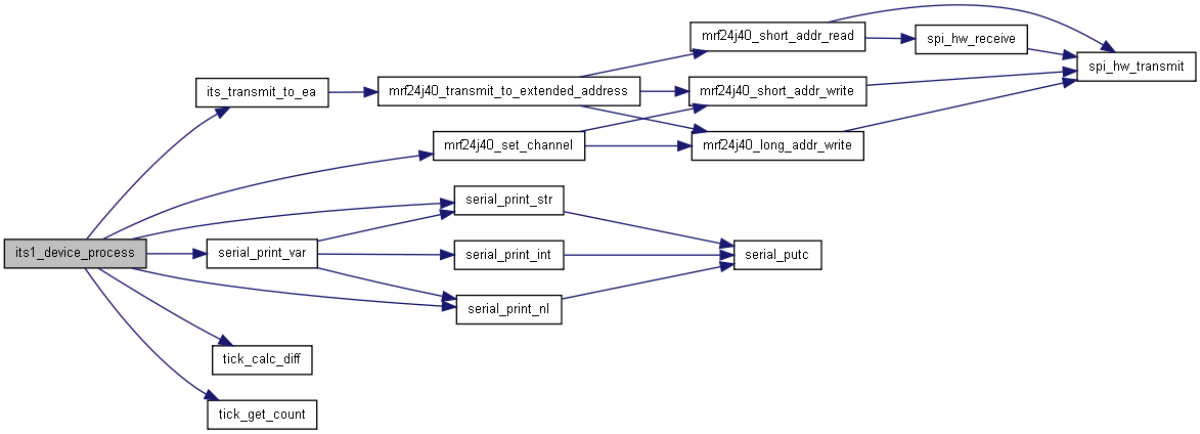


void its1_device_process ()

```
316                             {
317
318 uns16 test_tick;
319 uns8 controller_ea[8] = CONTROLLER_EA;
320
321     // are we searching?
322     // has time limit expired?
323     // go on to next channel
324     // send its association request to everyone
325     if (state == STATE_SEARCHING) {
326         test_tick = tick_get_count();
327         if (tick_calc_diff(tick_marker, test_tick) >= 250) {    // 250 = 1/4 second
328             tick_marker = test_tick;
329             if (channel == 0) {
330                 channel = MRF_FIRST_CHANNEL;
331             } else {
332                 if (channel == MRF_LAST_CHANNEL) {
333                     // Didn't find the controller
334                     state = STATE_UNASSOCIATED;
335                     serial_print_str("Didn't find controller\n");
336                 } else {
337                     channel++;
338                 }
339             }
340             // change channel
341             serial_print_var("Trying channel ", channel);
342             serial_print_nl();
343             mrf24j40_set_channel(channel);
344             // send broadcast association request
```

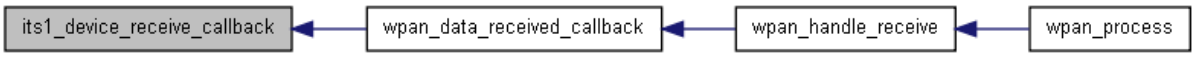
```
345             its_transmit_to_ea(&controller_ea, 0xffff, ITS_ASSOC_REQ, /* nil data */ 0, /*
zero length */ 0);
346         } // timer for another channel
347     } // we're searching
348
349 }
```

Here is the call graph for this function:



void its1_device_receive_callback (uns8 * *data*, uns8 *data_length*)

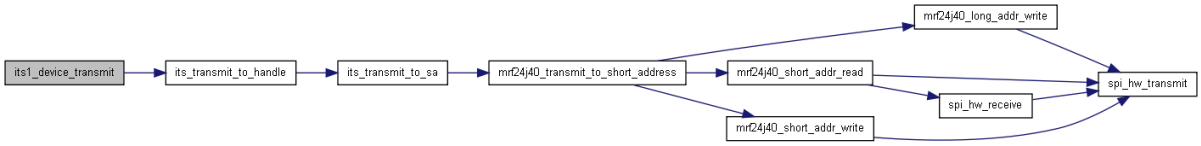
Here is the caller graph for this function:



its1_result its1_device_transmit (uns8 * *data*, uns8 *data_length*)

```
352                                                                 {
353
354 // We want to transmit to our controller
355         its_transmit_to_handle(controller_handle,  ITS_GENERIC_DATA, data, data_length);
356 }
```

Here is the call graph for this function:



its1_result its1_find_controller ()

```
310                                 {
311     state = STATE_SEARCHING;
312     tick_marker = tick_get_count();
313     channel = 0;
314 }
```

Here is the call graph for this function:

void its1_setup_io ()

```
57                      {
58      wpan_setup_io();
59  }
```

Here is the call graph for this function:

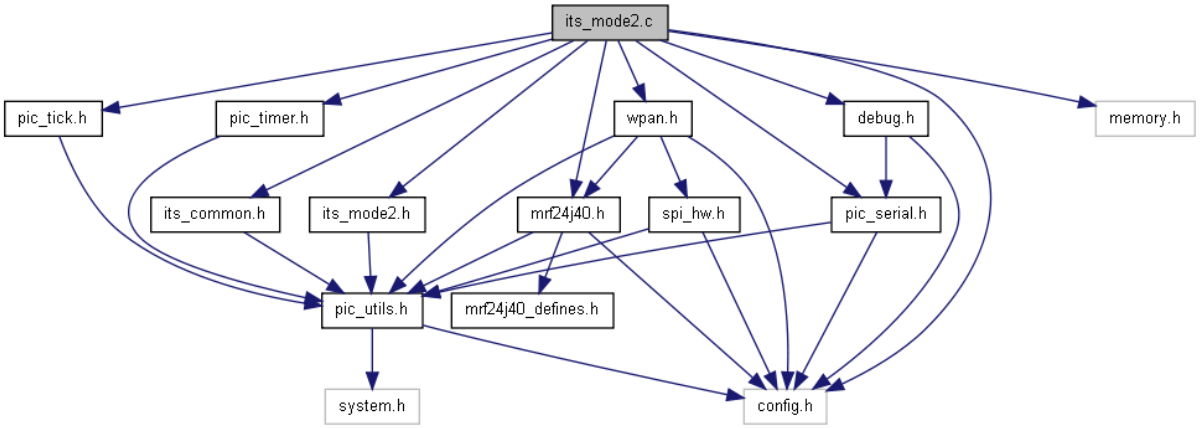
Variable Documentation

its1_state state

---

its_mode2.c File Reference

Include dependency graph for its_mode2.c:



Functions

- void its2_delete_item_from_queue (queued_item *item)
- its2_result its2_device_init (uns16 my_device_id, uns16 network_id)
- void its2_device_process ()
- its2_result its2_device_transmit (uns8 *data, uns8 data_length)
- its2_result its2_find_controller ()
- uns8 its2_find_free_queue_slot ()
- its2_result its2_forward_routed_packet (its2_packet *pkt, uns8 *data, uns8 data_length)
- void its2_print_packet (its2_packet *pkt)

- void its2_print_queue ()
- void its2_process_tx_queue ()
- void its2_rebroadcast_net_addr_req ()
- its2_result its2_rebroadcast_net_discover_req (its2_packet *pkt)
- void its2_request_local_addr (uns16 device_id)
- void its2_request_net_addr (uns16 device_id)
- void its2_respond_local_addr (uns16 device_id)
- void its2_respond_net_addr (uns16 device_id)
- its2_result its2_router_handle_association (uns16 pan_id, uns16 its_device_id)
- uns8 its2_router_init (uns16 my_device_id, uns16 network_id)
- void its2_router_process (queued_item *item)
- void its2_router_process ()
- its2_result its2_router_queue_packet (uns16 device_id, uns8 packet_type, uns8 *data, uns8 data_length, uns8 ack)
- void its2_setup_io ()
- void its2_transmit (queued_item *item)
- void turn_off_mrf_interrupts ()
- void turn_on_mrf_interrupts ()
- void wpan_data_received_callback (wpan_address *addr, uns8 *data, uns8 data_size)
- void wpan_data_transmitted_callback (uns8 status, uns8 retries, uns8 channel_busy)

Callback for data transmitted. Variables

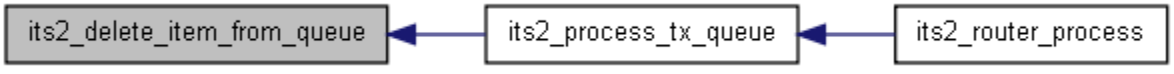
- uns8 channel
- uns8 controller_handle
- bit debug_module = 0
- uns8 its2_seen_index
- static seen_packet its2_seen_list [ITS2_SEEN_LIST_SIZE]
- bit its2_transmitting = 0
- static queued_item its2_tx_queue [ITS2_TX_QUEUE_SIZE]
- bit queue_processing = 0
- its2_state state = STATE_STARTUP
- uns16 state_timeout
- uns16 tick_marker

Function Documentation

void its2_delete_item_from_queue (queued_item * *item*)

```
1025                                                        {
1026     if (item->data_length > 0) {
1027         free((void *)item->data);
1028     }
1029     item->flag = ITS2_FLAG_DELETED;
1030 }
```

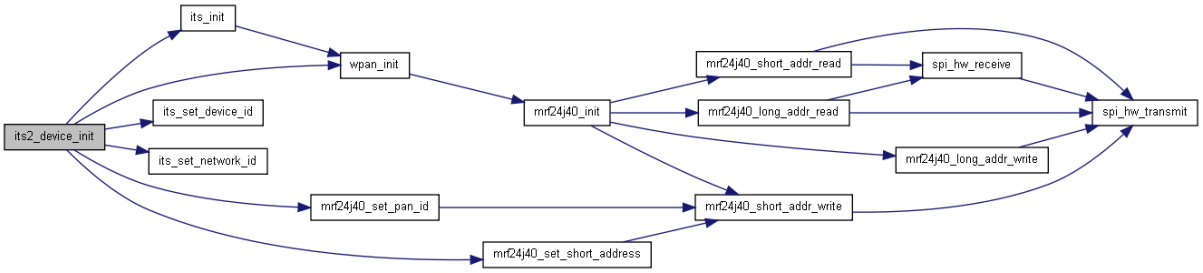
Here is the caller graph for this function:



its2_result its2_device_init (uns16 *my_device_id*, uns16 *network_id*)

```
1136                                                                    {
1137
1138 // ITS_EA is set in config.h
1139
1140 // uns8 my_ea[8] = ITS_EA;
1141
1142     debug_str("Device startup\n");
1143
1144     its_init();
1145     wpan_init();
1146
1147     its_set_device_id(my_device_id);
1148     its_set_network_id(network_id); // network id = controller id = pan id in mode 1
1149
1150     mrf24j40_set_pan_id(network_id);
1151     mrf24j40_set_short_address(my_device_id);   // same as device id
1152     //mrf24j40_set_extended_address(&my_ea);
1153
1154     state = STATE_UNASSOCIATED;
1155 }
```

Here is the call graph for this function:

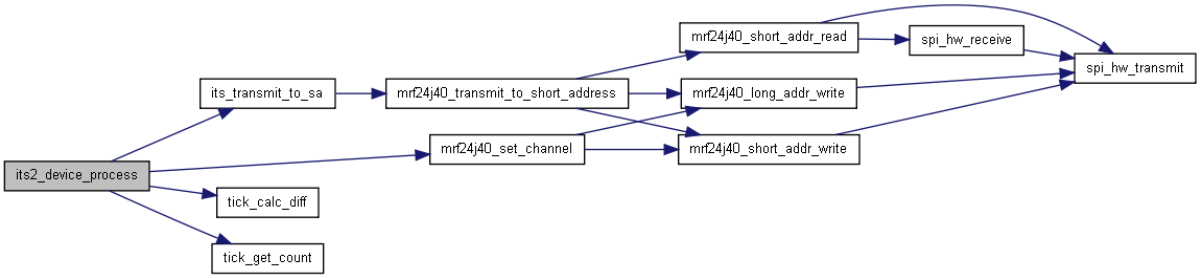


void its2_device_process ()

```
1163                            {
1164
1165 uns16 test_tick;
1166
1167     // are we searching?
1168     // has time limit expired?
1169     // go on to next channel
1170     // send its association request to everyone
1171     if (state == STATE_SEARCHING) {
1172         test_tick = tick_get_count();
1173         if (tick_calc_diff(tick_marker, test_tick) >= 250) {    // 250 = 1/4 second
1174             tick_marker = test_tick;
1175             if (channel == 0) {
1176                 channel = MRF_FIRST_CHANNEL;
1177             } else {
1178                 if (channel == MRF_LAST_CHANNEL) {
1179                     // Didn't find the controller
1180                     state = STATE_UNASSOCIATED;
1181                     debug_str("Didn't find controller\n");
1182                 } else {
1183                     channel++;
1184                 }
1185             }
1186             // change channel
1187             debug_var("Trying channel ", channel);
1188             debug_nl();
1189             mrf24j40_set_channel(channel);
1190             // send broadcast association request
```

```
1191             its_transmit_to_sa(/* pan id*/ 0xffff, /* sa */ 0xffff, /* dest device */ 0xffff,
1192                                ITS_ASSOC_REQ, /* nil data */ 0, /* zero length */ 0);
1193         } // timer for another channel
1194     } // we're searching
1195
1196 }
```

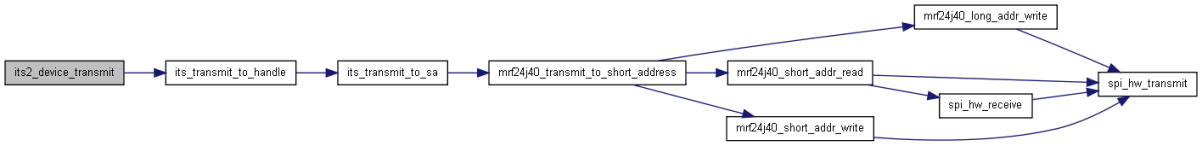
Here is the call graph for this function:



its2_result its2_device_transmit (uns8 * *data*, uns8 *data_length*)

```
1199                                                                    {
1200
1201 // We want to transmit to our controller
1202         its_transmit_to_handle(controller_handle,  ITS_GENERIC_DATA, data, data_length);
1203 }
```

Here is the call graph for this function:



its2_result its2_find_controller ()

```
1157                                    {
1158     state = STATE_SEARCHING;
1159     tick_marker = tick_get_count();
1160     channel = 0;
1161 }
```

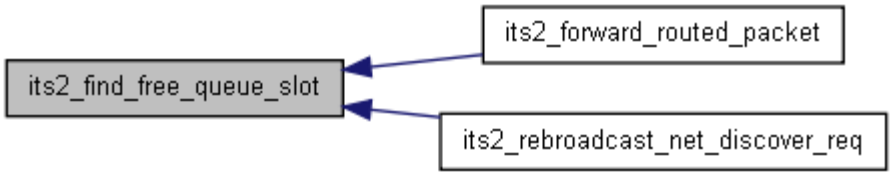
Here is the call graph for this function:

uns8 its2_find_free_queue_slot ()

```
99                                      {
100
101 bit found;
102 uns8 count;
103
104     found = 0;
105     for (count = 0; count < ITS2_TX_QUEUE_SIZE; count++) {
```

```
106         if (its2_tx_queue[count].flag  == ITS2_FLAG_DELETED) {
107             found = 1;
108             break;
109         }
110     }
111     if (found) {
112         return count;
113     } else {
114         return 0xff;
115     }
116 }
```

Here is the caller graph for this function:

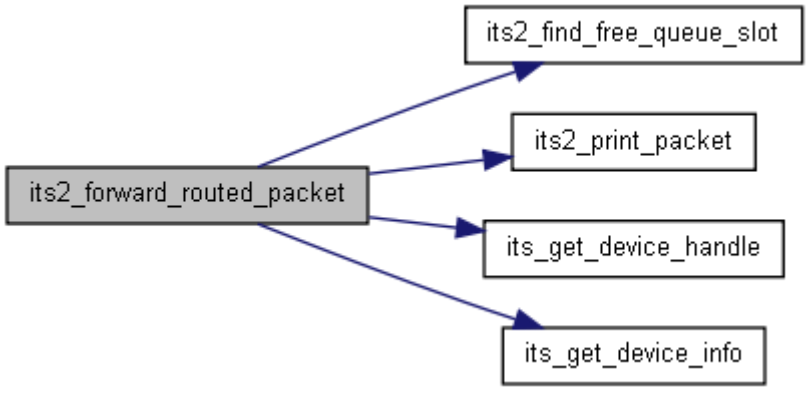


its2_result its2_forward_routed_packet (its2_packet * *pkt*, uns8 * *data*, uns8 *data_length*)

```
653                                                                                                 {
654
655 uns8 queue_slot;
656 queued_item *item;
657 void    *p;
658
659     queue_slot = its2_find_free_queue_slot();
660
661     if (queue_slot != ITS2_NO_AVAILABLE_SLOTS) {
662
663         debug_var("<Qed>", queue_slot);
664         debug_str(" data lengh=");
665         debug_int(data_length);
666         debug_spc();
667
668         item = &its2_tx_queue[queue_slot];
669         item->flag = ITS2_FLAG_NO_ACK;  // it's ours!
670         memcpy(/*dst*/  (void *)&item->packet, // copy it in
671                 /*src*/  (void *)pkt,
672                 /*len*/  11 + (pkt->num_routes * 2));
673
674         its2_print_packet(&item->packet);
675
676         item->packet.hop_count++;
677
678         // If hop_count now == num_routes then we are done routing
679         // and have to go final (local) destination.
680         // Otherwise, forward to next hop.
681
682         if (item->packet.hop_count == item->packet.num_routes) {
683             // Final desintation
684             debug_str(" going to final=");
685             item->dest_its_device_id = item->packet.its_dest_id;
686         } else {
687             // Forward to next hop
688             debug_str(" going to next hop=");
689             item->dest_its_device_id = item->packet.routers[item->packet.hop_count];
690         }
691
692         debug_int_hex_16bit(item->dest_its_device_id);
693
694         item->sent_count = 0;
695         item->dest_device_handle = its_get_device_handle(item->dest_its_device_id);
```

```
696
697         // Do we know about it already?
698         if (item->dest device handle == ITS DEVICE NONE) {
699             // Not in our routing table. Let's see if it's local
700             debug str(" unknown. Looking for local. ");
701             item->status = QS WAITING ON LOCAL ADDR;
702         } else {
703             // Check that it is really local
704             its device info *device_info = its get device info(item->dest device handle);
705
706             if (device info->addr.remote.remote indicator == 0xffff) {
707                 debug str(" Not local! Abandon ship ");
708                 item->flag = ITS2 FLAG DELETED; // delete it
709                 return NEXT HOP NOT LOCAL;  // let our caller know
710                 // !! we should send something back to originator
711                 // to say that we couldn't route it
712             } else {
713                 debug str(" found locally. ");
714                 item->status = QS READY TO SEND;
715             }
716         }
717         item->data length = 0;
718         item->data = NULL;
719
720         // copy data...
721
722         if (data_length > 0) {
723             p = alloc(data length);
724
725             memcpy(/*dst*/  p, // copy it in
726                 /*src*/  (void *)data,
727                 /*len*/  data length);
728             item->data =  (uns8*)p;
729         }
730
731         item->data length = data_length;
732
733         return ITEM QUEUED;
734     } else {
735         return QUEUE FULL;
736     }
737
738 }
```

Here is the call graph for this function:

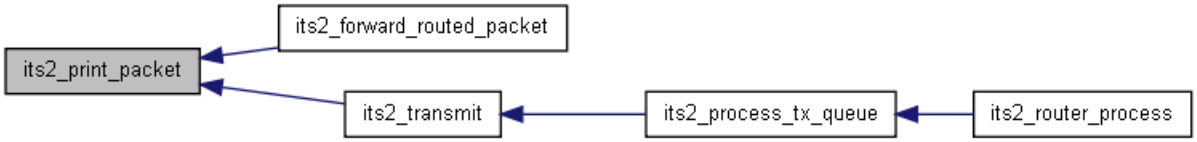


void its2_print_packet (its2_packet * *pkt*)

```
580                                         {
581
```

```
582     debug_str("(pkt type=");
583     debug_int(pkt->packet_type);
584     debug_str(" seq=");
585     debug_int(pkt->sequence);
586     debug_str(" net=");
587     debug_int_hex_16bit(pkt->its_network_id);
588     debug_str(" src=");
589     debug_int_hex_16bit(pkt->its_source_id);
590     debug_str(" dst=");
591     debug_int_hex_16bit(pkt->its_dest_id);
592     debug_str(" max_hop=");
593     debug_int(pkt->max_hop_count);
594     debug_str(" num_rts=");
595     debug_int(pkt->num_routes);
596     debug_str(" hop_cnt=");
597     debug_int(pkt->hop_count);
598 
599     uns8 count;
600     debug_str(" Rts=");
601     for (count = 0; count < pkt->num_routes; count++) {
602         if (count > ITS2_MAX_HOP_COUNT) {
603             debug_str(" Too many routes! ");
604             break;
605         } else {
606             debug_putc(' ');
607             debug_int_hex_16bit(pkt->routers[count]);
608         }
609     }
610     debug_str(") ");
611 }
```

Here is the caller graph for this function:



void its2_print_queue ()

```
1106                         {
1107 
1108 uns8 count;
1109 queued_item *item;
1110 
1111         debug_nl();
1112         for (count = 0; count < ITS2_TX_QUEUE_SIZE; count++) {
1113             item = &its2_tx_queue[count];
1114             if (item->flag != ITS2_FLAG_DELETED) {
1115                 debug_var("Q:",count);
1116                 debug_var(" FL:", item->flag);
1117                 debug_var(" SRC:", item->packet.its_source_id);
1118                 debug_var(" DST:", item->packet.its_dest_id);
1119                 debug_var(" STAT:", item->status);
1120                 debug_var(" SENT:", item->sent_count);
1121                 debug_var(" TICK:", item->tick_sent);
1122                 debug_nl();
1123             }
1124         }
1125         debug_var("Current tick", tick_get_count());
1126         debug_nl();
1127 }
```

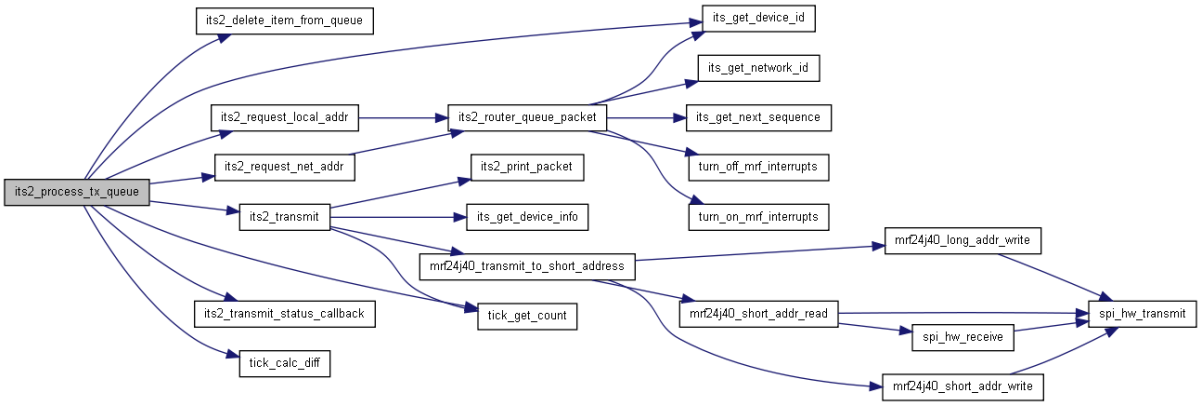
Here is the call graph for this function:

void its2_process_tx_queue ()

```
904                                 {
905
906 uns16       current_tick;
907 uns8        count;
908 queued item *item;
909 uns8        flag;
910 uns8        status;
911 uns16       my_id;
912     //debug_str("\n[PQ]\n");
913
914     current_tick = tick_get_count();
915     my_id = its_get_device_id();
916
917     //clear_bit(intcon3, INT1IE);       // disable int1
918
919
920     for (count = 0; count < ITS2_TX_QUEUE_SIZE; count++) { // search for a packet
921         item = &its2_tx_queue[count];   // grab item to save code
922         flag = item->flag;
923         status = item->status;
924         if (flag == ITS2_FLAG_DELETED) {
925             continue;
926         }
927         //debug_var("\nQ-", count);
928         if (status == QS_WAITING_ON_LOCAL_ADDR) {
929             if (item->sent_count == ITS2_SEND_MAX_TRIES) { // Sent too many times
930                 if (item->packet.its_source_id == my_id) {  // from me?
931                     debug_str(" Tried local, now time for net search");
932                     item->sent_count = 0;
933                     item->status = QS_WAITING_ON_NETWORK_ADDR;
934                 } else {    // Must have been a network route
935                     // but we failed to find the next hop locally
936                     // !! Should return something here to sender
937                     debug_str("Next hop not available locally. Giving up.");
938                     its2_transmit_status_callback(&item->packet,
ITS2_TX_STATUS_NEXT_HOP_UNKNOWN);
939                     its2_delete_item_from_queue(item);
940                 }
941             } else if ((item->sent_count == 0) || (tick_calc_diff(item->tick_sent,
current_tick)
942                     > ITS2_RESEND_TICK_DELAY)) {
943
944                 debug_str(" REQ local address for q ");
945                 debug_int(count);
946                 debug_str(" dev ");
947                 debug_int_hex_16bit(item->dest_its_device_id);
948
949                 item->tick_sent = current_tick;
950                 item->sent_count++;
951                 its2_request_local_addr(item->dest_its_device_id);
952                 item->tick_sent = current_tick; // set tick count to current
953             } else {
954                 debug_str(" Local waiting for ");
955                 debug_int(item->tick_sent);
956             }
957         } else
958         if (status == QS_WAITING_ON_NETWORK_ADDR) {
959             if (item->sent_count == ITS2_SEND_MAX_TRIES) { // Sent too many times
960                 // !! Need to issue callback here
961                 debug_str("No addr. killing");
962                 its2_transmit_status_callback(&item->packet, ITS2_TX_STATUS_NO_ROUTE);
963                 its2_delete_item_from_queue(item);
```

```
964             } else {
965                 //debug_str("Wait for ");
966                 //debug_int(item->tick_sent);
967                 //debug_str("+ 5000 current=");
968                 //debug_int(current_tick);
969                 if (tick_calc_diff(item->tick_sent, current_tick)
970                      > ITS2_RESEND_TICK_DELAY) {
971                     item->tick_sent = current_tick;
972                     item->sent_count++;
973                     its2_request_net_addr(item->dest_its_device_id);
974                 }
975             }
976         } else
977         if (status == QS_READY_TO_SEND) {
978             if (!its2_transmitting) {
979                 debug_str("<TX!>");
980                 its2_transmit(item);
981                 debug_str("<Done>");
982 
983             }
984         } else
985         if (status == QS_WAITING_ON_ACK) {
986             if (item->sent_count > ITS2_SEND_MAX_TRIES) { // Sent too many times
987                 // !! give up, go home... FAIL! No address
988                 // issue call back
989                 // kill from queue
990                 debug_str("Too many. killing.");
991                 its2_transmit_status_callback(&item->packet, ITS2_TX_STATUS_NO_ACK);
992                 its2_delete_item_from_queue(item);
993             }
994             if (tick_calc_diff(item->tick_sent, current_tick)
995                      > ITS2_RESEND_TICK_DELAY) {
996                 if (!its2_transmitting) {
997                     debug_str("<RETRYTX!>");
998                     its2_transmit(item);
999                     debug_str("<Done>");
1000                 }
1001             }
1002         }
1003         if (status == QS_ACK_RECEIVED) {
1004             // !! call back here
1005             its2_transmit_status_callback(&item->packet, ITS2_TX_STATUS_SUCCESS);
1006 
1007             its2_delete_item_from_queue(item);
1008         } else
1009         if (status == QS_SENT) {
1010             debug_str(" <SENT del> ");
1011             its2_transmit_status_callback(&item->packet, ITS2_TX_STATUS_SUCCESS);
1012             its2_delete_item_from_queue(item);
1013         } else
1014         if (status == QS_ROUTING_FAILED) {
1015             debug_str(" <RFAIL del> ");
1016             its2_transmit_status_callback(&item->packet, ITS2_TX_STATUS_NO_ROUTE);
1017             its2_delete_item_from_queue(item);
1018         }
1019     }
1020 
1021     //set_bit(intcon3, INT1IE);     // enable int1
1022 
1023 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void its2_rebroadcast_net_addr_req ()

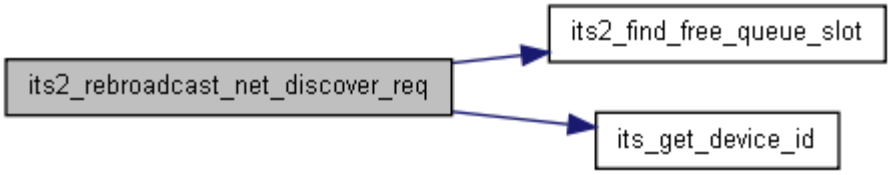
```
800                                         {
801 }
```

its2_result its2_rebroadcast_net_discover_req (its2_packet * *pkt*)

```
613                                                                         {
614
615 uns8 queue_slot;
616 queued_item *item;
617
618     if (pkt->hop_count == pkt->max_hop_count) {
619         debug_str("<Can't re-b: max hop count exceeded!>");
620         return ROUTING_TOO_MANY_HOPS;
621     }
622     queue_slot = its2_find_free_queue_slot();
623     if (queue_slot != ITS2_NO_AVAILABLE_SLOTS) {
624
625         debug_var("<Qed>", queue_slot);
626         item = &its2_tx_queue[queue_slot];
627
628         item->flag = ITS2_FLAG_NO_ACK;  // it's ours!
629         memcpy(/*dst*/  (void *)&item->packet, // copy it in
630                 /*src*/  (void *)pkt,
631                 /*len*/  11 + (pkt->num_routes * 2));
632         item->dest_its_device_id = 0xffff;
633         item->packet.routers[item->packet.hop_count] = its_get_device_id();
634         item->packet.hop_count++;
635         item->packet.num_routes++;
636         item->sent_count = 0;
637         item->dest_device_handle = ITS_DEVICE_NONE; // it's a broadcast
638         item->data_length = 0;
639         item->data = NULL;
640
641
642         item->status = QS_READY_TO_SEND;
643
644         its2_add_to_seen_list(item->packet.its_source_id, item->packet.sequence);
645
646         return ITEM_QUEUED;
```

```
647     } else {
648         return QUEUE_FULL;
649     }
650 
651 }
```

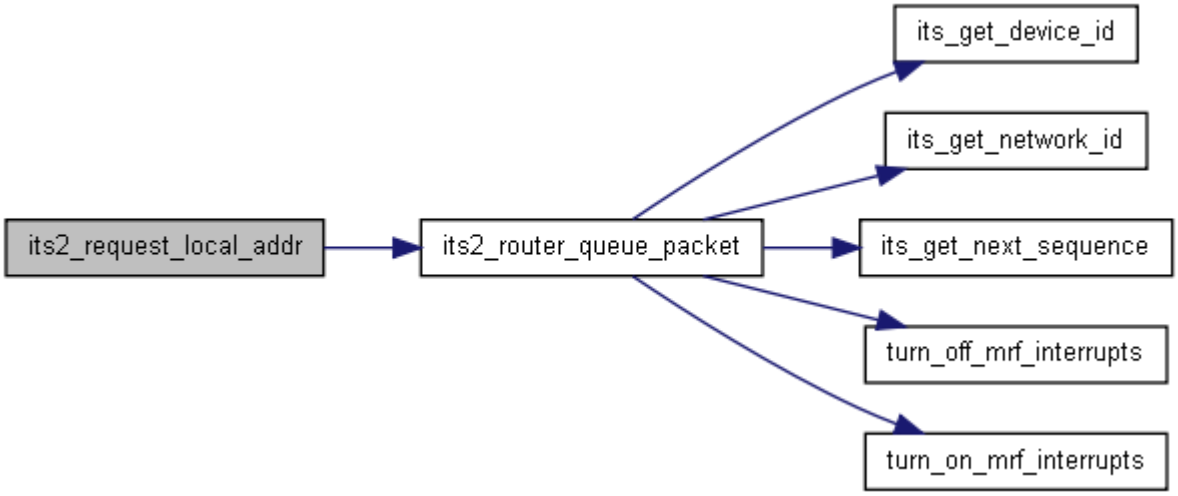
Here is the call graph for this function:



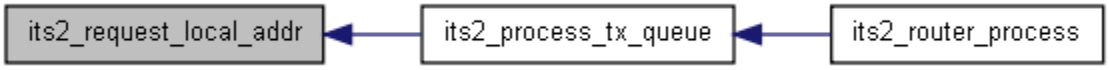
void its2_request_local_addr (uns16 *device_id*)

```
779                                             {
780     its2_router_queue_packet(device_id, ITS_LOCAL_DISCOVER_REQ, NULL, 0, ITS2_FLAG_NO_ACK);
781 }
```

Here is the call graph for this function:



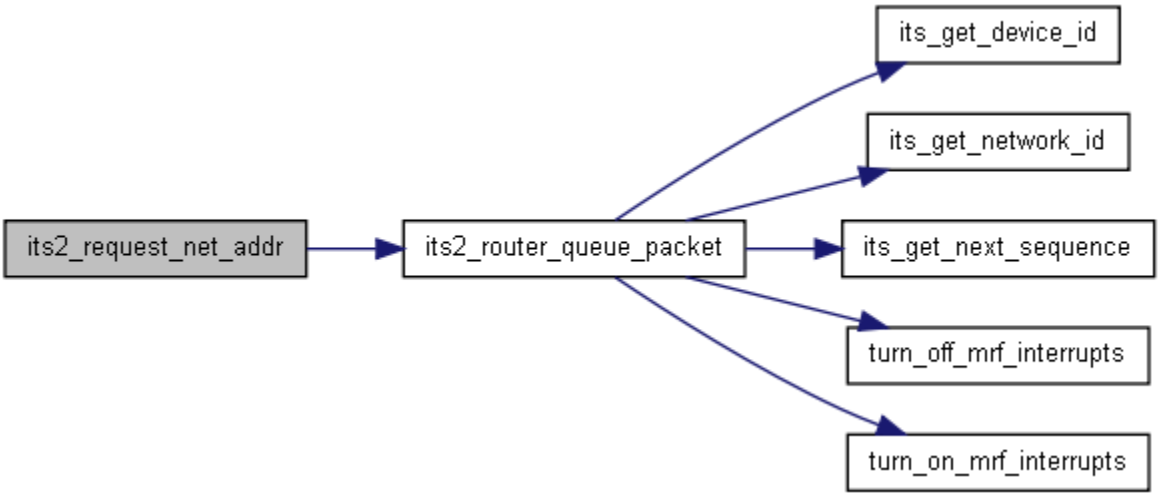
Here is the caller graph for this function:



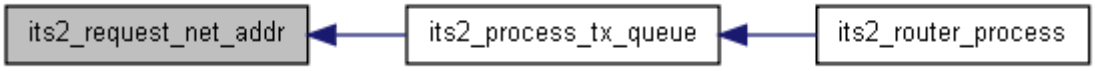
void its2_request_net_addr (uns16 *device_id*)

```
792                                         {
793     its2_router_queue_packet(device_id, ITS_NET_DISCOVER_REQ, NULL, 0, ITS2_FLAG_NO_ACK);
794 }
```

Here is the call graph for this function:



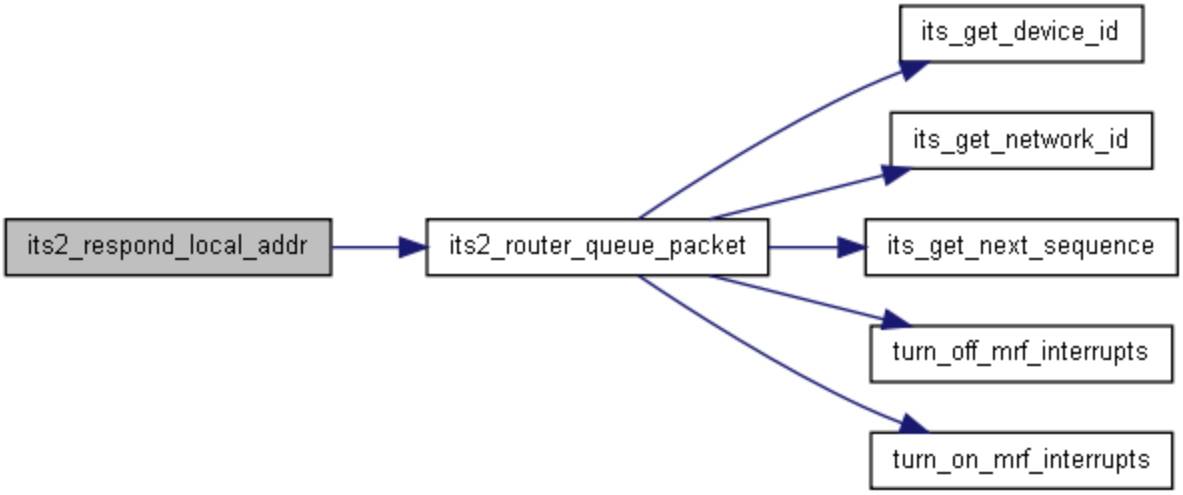
Here is the caller graph for this function:



void its2_respond_local_addr (uns16 *device_id*)

```
783                                                 {
784
785     if ((device_id != 0x0001) || !debug_module) {
786         its2_router_queue_packet(device_id, ITS_LOCAL_DISCOVER_RES, NULL, 0,
ITS2_FLAG_NO_ACK);
787     } else {
788         debug_str("IGNORE");
789     }
790 }
```

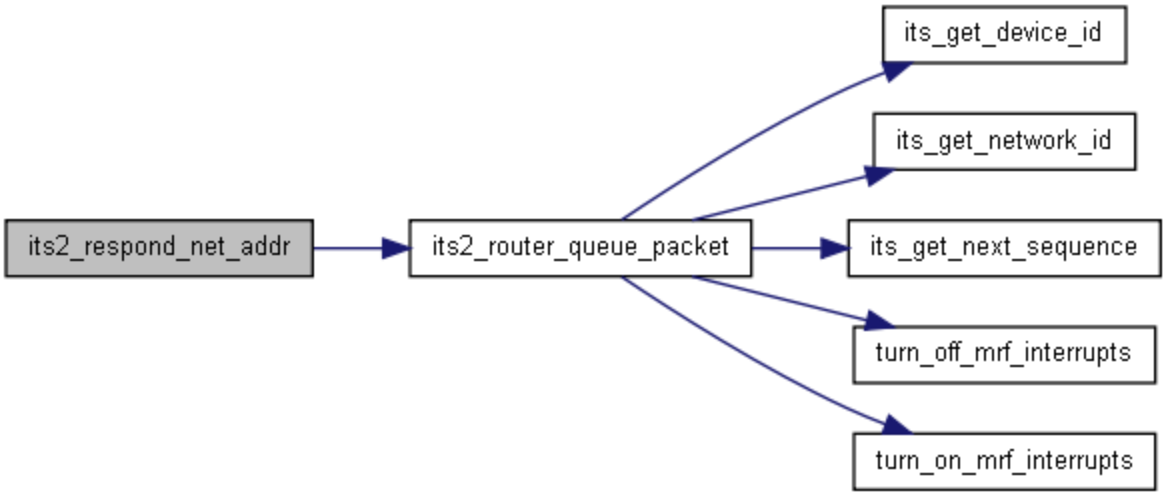
Here is the call graph for this function:



void its2_respond_net_addr (uns16 *device_id*)

```
796                                                 {
797     its2_router_queue_packet(device_id, ITS_NET_DISCOVER_RES, NULL, 0, ITS2_FLAG_NO_ACK);
798 }
```

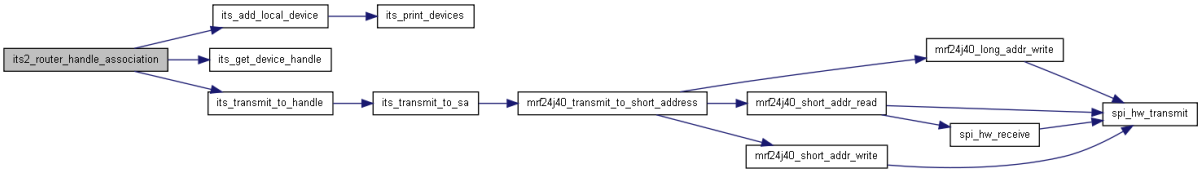
Here is the call graph for this function:



its2_result its2_router_handle_association (uns16 *pan_id*, uns16 *its_device_id*)

```
118                                                                                 {
119     // see if this device is in our list, if not, add it
120 its_device_handle device_handle;
121
122     device_handle = its_get_device_handle(its_device_id);
123
124     if (device_handle == ITS_DEVICE_NONE) {
125         // in mode 1 we assume that
126         device_handle = its_add_local_device(/* device */ its_device_id, pan_id, /* short addr
*/ its_device_id);
127         if (device_handle == ITS_DEVICE_NONE) {
128             // panic!
129             debug_str("Too many devices! association failed");
130             return RESULT_FAILED;
131         } else {
132             // send association response
133             its_transmit_to_handle(device_handle, ITS_ASSOC_RES, /* nil data */ 0, /* zero
length */ 0);
134             return RESULT_SUCCESSFUL;
135             // although strictly speaking we should wait for the ack...
136         }
137
138     }
139 }
```

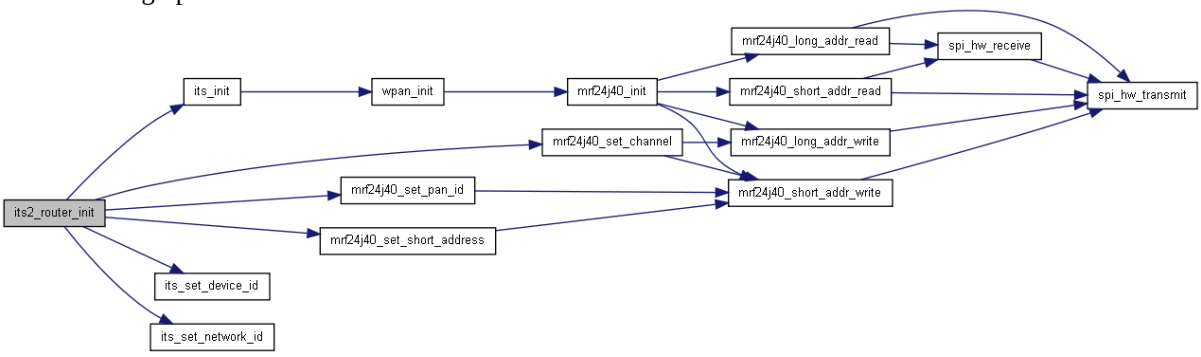
Here is the call graph for this function:



uns8 its2_router_init (uns16 *my_device_id*, uns16 *network_id*)

```
743                                                     {
744
745 uns8 lowest_channel_ed;
746 uns8 count;
747
748 //uns8 my_ed[8] = ITS_EA;
749
750     its2_seen_index = 0;
751
752     for (count=0; count < ITS2_SEEN_LIST_SIZE; count++) {
753         its2_seen_list[count].its_source_id = 0xffff;
754     }
755
756     for (count = 0; count < ITS2_TX_QUEUE_SIZE; count++) {
757         its2_tx_queue[count].flag  = ITS2_FLAG_DELETED;
758     }
759
760     its_init();
761
762     its_set_device_id(my_device_id);
763     its_set_network_id(network_id); // network id = controller id = pan id in mode 1
764
765     mrf24j40_set_pan_id(network_id);
766     mrf24j40_set_short_address(my_device_id);   // same as device id
767     //mrf24j40_set_extended_address(&my_ed);
768
769     debug_str("Router startup\n");
770
771     debug_var("Chosing channel", 15);
772     debug_nl();
773
774     // Now set ourselves up on this channel
775     mrf24j40_set_channel(15);
776
777 }
```

Here is the call graph for this function:



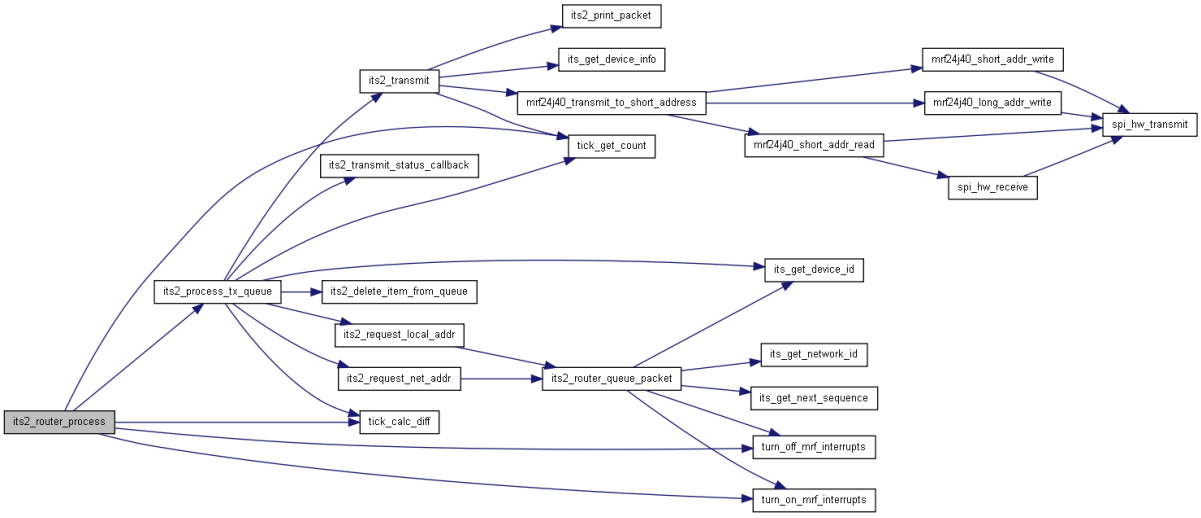
void its2_router_process (queued_item * *item*)

```
1102                                                    {
1103
1104 }
```

void its2_router_process ()

```
808                             {
809
810     turn_off_mrf_interrupts();
811
812     its2_process_tx_queue();
813
814 uns16 test_tick;
815     test_tick = tick_get_count();
816     if (tick_calc_diff(tick_marker, test_tick) >= state_timeout) {
817     // timeout
818     }
819     turn_on_mrf_interrupts();
820
821 }
```

Here is the call graph for this function:

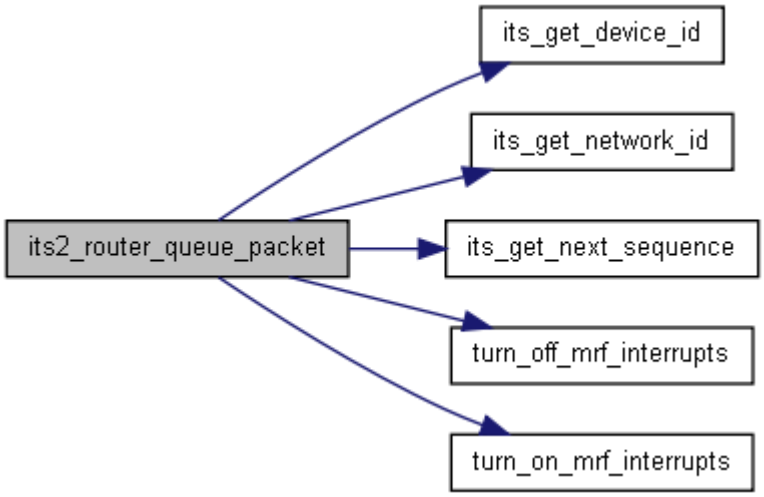


its2_result its2_router_queue_packet (uns16 *device_id*, uns8 *packet_type*, uns8 * *data*, uns8 *data_length*, uns8 *ack*)

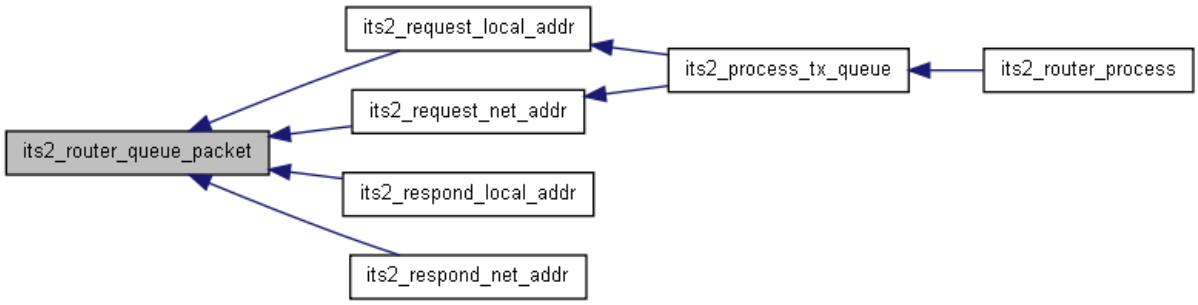
```
823
{
824
825 // return result - queue handle?
826
827 uns8 found;
828 void    *p;
829 queued_item *item;
830 uns8 count;
831
832 turn_off_mrf_interrupts();
833
834 debug_str("\n[its2 router queue packet]\n");
835
836     // find slot
837     found = 0;
838     for (count = 0; count < ITS2_TX_QUEUE_SIZE; count++) {
839         if (its2_tx_queue[count].flag  == ITS2_FLAG_DELETED) {
840             found = 1;
841             break;
842         }
843     }
```

```
844     if (found) {
845         debug_var("Qed: ", count);
846         item = &its2_tx_queue[count];
847         item->flag = ack;   // it's ours!
848         item->sent_count = 0;
849 
850         //item->packet.key_1 = 'I';
851         //item->packet.key_2 = 'T';
852         // item->length_header; // unknown as yet
853         item->packet.packet_type = packet_type;
854         item->packet.sequence = its_get_next_sequence();
855         item->packet.its_network_id = its_get_network_id();
856         item->packet.its_source_id = its_get_device_id();
857         item->packet.its_dest_id = device_id;
858         if ((packet_type == ITS_LOCAL_DISCOVER_REQ) ||
859             (packet_type == ITS_NET_DISCOVER_REQ)) {
860             item->dest_its_device_id = 0xffff;
861         } else {
862             item->dest_its_device_id = device_id;
863         }
864         item->packet.max_hop_count = ITS2_MAX_HOP_COUNT;
865         item->packet.hop_count = 0;
866         // item->packet.num_routes;
867 
868         // uns16 routers[5];
869             // create some memory to whack the data in
870         if (data_length > 0) {
871             p = alloc(data_length);
872 
873             memcpy(/*dst*/  p, // copy it in
874                 /*src*/  (void *)data,
875                 /*len*/  data_length);
876             item->data =  (uns8*)p;
877         }
878 
879         item->data_length = data_length;
880 
881         // Here we will update the route and also set
882         // dest_device_handle (which may not be the same as the
883         // eventual destination, since we may be going through
884         // other hops or doing a broadcast)
885         uns8 routing_result = its2_update_route(item);
886 
887         turn_on_mrf_interrupts();
888 
889         if (routing_result == ITS2_UPDATE_ROUTE_FAIL) {
890             debug_str("RoUtE fail");
891             return ROUTING_TOO_MANY_HOPS;
892         } else {
893             debug_str("qdone ");
894             return ITEM_QUEUED;
895         }
896     } else {
897         turn_on_mrf_interrupts();
898         debug_str(" Full! ");
899         return QUEUE_FULL;
900     }
901 }
```

Here is the call graph for this function:



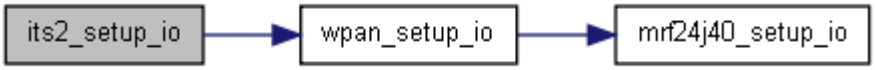
Here is the caller graph for this function:



void its2_setup_io ()

```
95                      {
96     wpan_setup_io();
97 }
```

Here is the call graph for this function:

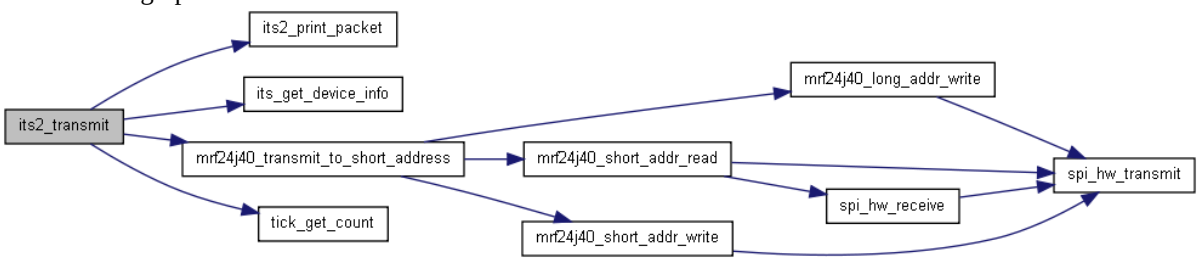


void its2_transmit (queued_item * *item*)

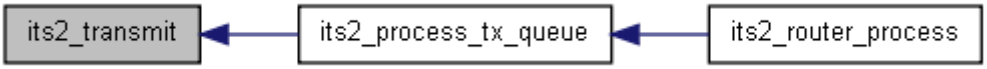
```
1035                                    {
1036
1037 uns8 buffer[50];
1038 uns8 count;
1039 uns8 index;
1040 its_device_info *device_info;
1041
1042     its2_transmitting = 1;
1043
1044     debug_str(" X: ");
1045     its2_print_packet(&item->packet);
1046     //p = (void *)alloc(item->data_size + ??);
1047     buffer[0] = 'I';
```

```
1048     buffer[1] = 'T';
1049     memcpy(/* dst */ (void *)&buffer[3], /* src */ (void *)&item->packet, 11 );
1050     // now need to copy routes
1051     index = 11 + 3;
1052     for (count = 0; count < item->packet.num_routes; count++) {
1053         buffer[index++] = item->packet.routers[count] & 0xff;
1054         buffer[index++] = item->packet.routers[count] >> 8;
1055     }
1056
1057     buffer[2] = index - 3;  // update header with length
1058
1059     // now copy data
1060     for (count = 0; count < item->data_length; count++) {
1061         buffer[index++] = item->data[count];
1062     }
1063     debug_str(" To=");
1064     debug_int_hex_16bit(item->dest_its_device_id);
1065     debug_str(" bytes=");
1066     debug_int(index);
1067
1068     item->sent_count++;
1069     item->tick_sent = tick_get_count();
1070     if (item->flag == ITS2_FLAG_ACK) {  // ack
1071         item->status = QS_WAITING_ON_ACK;
1072     } else {
1073         item->status = QS_SENT;
1074     }
1075     if (item->dest_device_handle != ITS_DEVICE_NONE) {  // we have handle
1076         debug_var(" Handle ", item->dest_device_handle);
1077         device_info = its_get_device_info(item->dest_device_handle);
1078         if (device_info != NULL) {
1079             debug_str(" Pan ");
1080             debug_int_hex_16bit(device_info->addr.local.pan_id);
1081             debug_str(" SA ");
1082             debug_int_hex_16bit(device_info->addr.local.short_address);
1083         } else {
1084             debug_str("!!Strange!! - NO DEVICE INFO!!\n");
1085         }
1086         mrf24j40_transmit_to_short_address(FRAME_TYPE_DATA,
device_info->addr.local.pan_id,
1087                                            device_info->addr.local.short_address,
1088                                            &buffer, index, MRF_ACK);
1089         debug_str(" Sent bytes ");
1090         debug_int(index);
1091         debug_putc(' ');
1092     }
1093     else {  // it's a broadcast
1094         debug_str(" It's a broadcast");
1095         mrf24j40_transmit_to_short_address(FRAME_TYPE_DATA, 0xffff,
1096                                            0xffff, &buffer, index, MRF_NO_ACK);
1097     }
1098
1099 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

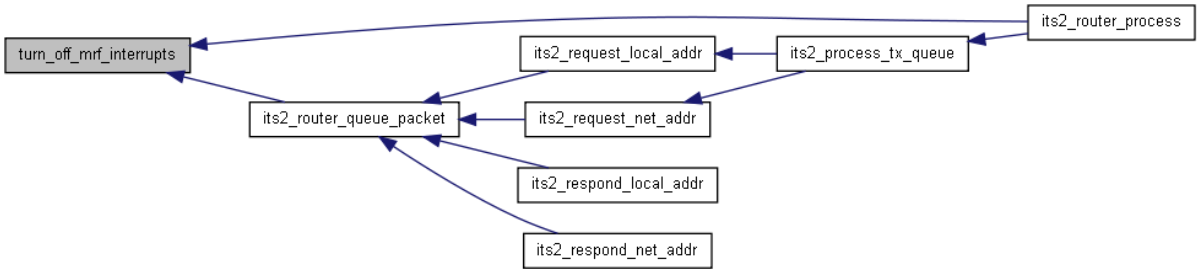


void turn_off_mrf_interrupts ()

!clear_bit(intcon, INT0IE);

```
82                                      {
84      nop();
85      nop();
86      nop();
87      nop();
88  }
```

Here is the caller graph for this function:

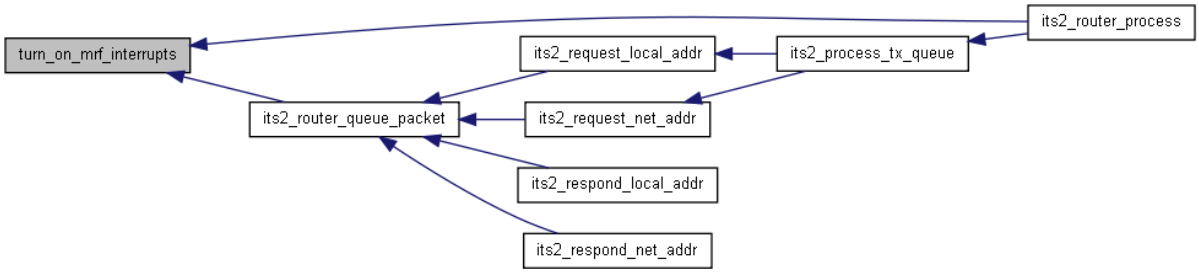


void turn_on_mrf_interrupts ()

!set_bit(intcon, INT0IE);

```
90                                      {
92  }
```

Here is the caller graph for this function:

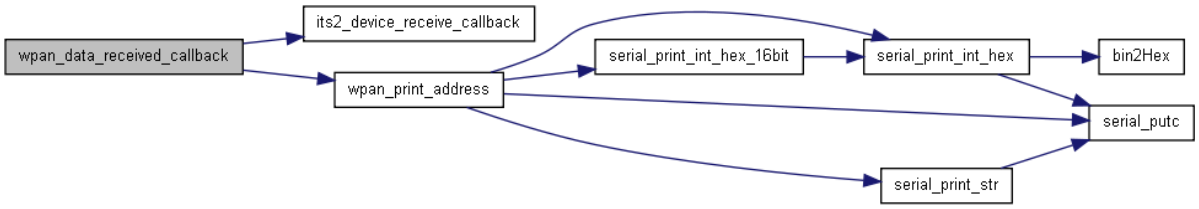


void wpan_data_received_callback (wpan_address * *addr*, uns8 * *data*, uns8 *data_size*)

```
508                                                                                 {
509
510 uns8 count;
511     wpan_print_address(addr);
512     //wpan_print_frame_type(frame_type);
513     debug_str(" Data: ");
```

```
514     for (count = 0; count < data_size; count++) {
515         debug_int_hex(data[count]);
516         debug_putc(' ');
517         if ((data[count] > 31) && (data[count] < 127)) {
518             debug_putc(data[count]);
519         } else {
520             debug_putc('?');
521         }
522         debug_putc(' ');
523     }
524     debug_nl();
525 
526     // check frame type
527     if ((data_size > 2) && (data[0] == 'I') && (data[1] == 'T')) {
528         debug_str("ITS Packet received\n");
529         uns8 length_header = data[2];
530         uns8 length_data   = data[3+length_header+1];
531         uns8 data_start    = 3+length_header+2;
532         // okay, but is it an its association request?
533         if (data[3] == ITS_ASSOC_RES) {
534             debug_str("Received association response\n");
535             if (state == STATE_SEARCHING) {
536                 debug_str("Changing state to associated\n");
537                 uns16 controller_device_id = data[8]; // device id of controller
538                 controller_device_id <<= 8;
539                 controller_device_id += data[7];
540 
541                 uns8 device_handle = its_get_device(controller_device_id);
542 
543                 if (device_handle == ITS_DEVICE_NONE) {
544 
545                     device_handle = its_add_device(/* device */ controller_device_id,
addr->source_pan_id, controller_device_id);
546                     if (device_handle == ITS_DEVICE_NONE) {
547                         // panic!
548                         debug_str("Too many devices! association failed");
549                         state = STATE_UNASSOCIATED;
550                     } else {
551                         controller_handle = device_handle;
552                         state = STATE_ASSOCIATED;   // stop searching
553                     }
554                 } else { // it's already in the table
555                     state = STATE_ASSOCIATED;
556                 }
557 
558             }
559         }   // association response
560         else if (data[3] == ITS_GENERIC_DATA) {
561             // is it for us?
562             // who cares right now!
563             // todo...
564             debug_str("Generic data pkt\n");
565             // just issue the callback
566             uns16 device_id = data[8];  // msb
567             device_id  <<= 8;
568             device_id += data[7];   // lsb
569             its2_device_receive_callback(&data[data_start], data_size - data_start);
570 
571         }   // GENERIC_DATA
572     }
573 }
```

Here is the call graph for this function:



void wpan_data_transmitted_callback (uns8 *status*, uns8 *retries*, uns8 *channel_busy*)

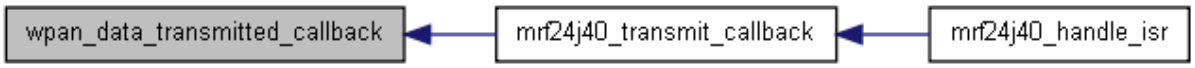
Once the lower layers have attempted to transmit the packet, the results will be presented to the callback.

Parameters:

status Set to 0 for success or 1 for failure
retries Set to the number of retries
channel_busy Set to 1 if failure was due to the channel being busy.

```
804                                                                    {
805     its2_transmitting = 0;
806 }
```

Here is the caller graph for this function:



Variable Documentation

uns8 channel

uns8 controller_handle

bit debug_module = 0

uns8 its2_seen_index

seen_packet its2_seen_list[ITS2_SEEN_LIST_SIZE] [static]

bit its2_transmitting = 0

queued_item its2_tx_queue[ITS2_TX_QUEUE_SIZE] [static]

bit queue_processing = 0

its2_state state = STATE_STARTUP

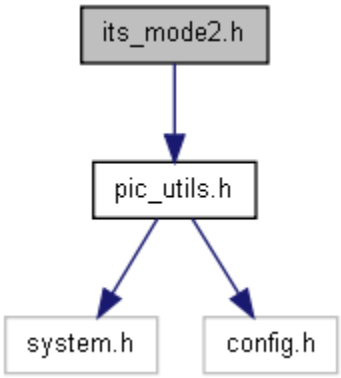
uns16 state_timeout

uns16 tick_marker

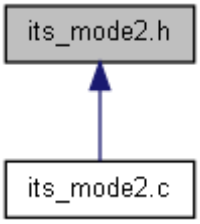
---

its_mode2.h File Reference

ITS Mesh networking mode 2.
Include dependency graph for its_mode2.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct its2_packet
- struct queued_item
- struct seen_packet

Defines

- #define ITS2_FLAG_ACK 0x01
- #define ITS2_FLAG_DELETED 0xff
- #define ITS2_FLAG_NO_ACK 0x02
- #define ITS2_NO_AVAILABLE_SLOTS 0xff
- #define ITS2_STATUS_QUEUED 0x00
- #define ITS2_STATUS_TX_QUEUE_FULL 0x01
- #define ITS2_TX_STATUS_NEXT_HOP_UNKNOWN 0x04
- #define ITS2_TX_STATUS_NO_ACK 0x02
- #define ITS2_TX_STATUS_NO_ROUTE 0x01
- #define ITS2_TX_STATUS_REMOTE_NO_ACK 0x03
- #define ITS2_TX_STATUS_SUCCESS 0x00
- #define ITS2_UPDATE_ROUTE_FAIL 0x01
- #define ITS2_UPDATE_ROUTE_SUCCESS 0x00
- #define POS_DEST_H 0x0a
- #define POS_DEST_L 0x09
- #define POS_HOP_COUNT 0x0d
- #define POS_KEY1 0x00
- *!! look at the adding in routes bit...* #define POS_KEY2 0x01

- #define POS_LENGTH_HEADER 0x02
- #define POS_MAX_HOP_COUNT 0x0b
- #define POS_NETWORK_H 0x06
- #define POS_NETWORK_L 0x05
- #define POS_NUM_ROUTES 0x0c
- #define POS_PKT_TYPE 0x03
- #define POS_ROUTE_START 0x0e
- #define POS_SEQUENCE 0x04
- #define POS_SOURCE_H 0x08
- #define POS_SOURCE_L 0x07
- #define QS_ACK_RECEIVED 0x04
- #define QS_READY_TO_SEND 0x02
- #define QS_ROUTING_FAILED 0x06
- #define QS_SENT 0x05
- #define QS_WAITING_ON_ACK 0x03
- #define QS_WAITING_ON_LOCAL_ADDR 0x00
- #define QS_WAITING_ON_NETWORK_ADDR 0x01
- #define REMOTE_DEVICE 0xffff

Typedefs

- typedef enum _its2_result its2_result
- typedef enum _its2_state its2_state

Enumerations

- enum _its2_result { RESULT_SUCCESSFUL, RESULT_FAILED, ITEM_QUEUED, QUEUE_FULL, ROUTING_TOO_MANY_HOPS, NEXT_HOP_NOT_LOCAL }
- enum _its2_state { STATE_STARTUP, STATE_RUNNING, STATE_SEARCHING, STATE_ASSOCIATED, STATE_UNASSOCIATED }

Functions

- void its2_delete_item_from_queue (queued_item *item)
- its2_result its2_device_init (uns16 my_device_id, uns16 network_id)
- void its2_device_process ()
- void its2_device_receive_callback (uns8 *data, uns8 data_length)
- its2_result its2_device_transmit (uns16 device_id, uns8 *data, uns8 data_length)
- its2_result its2_find_coordinator ()
- its2_result its2_forward_routed_packet (its2_packet *pkt, uns8 *data, uns8 data_length)
- void its2_print_packet (its2_packet *pkt)
- void its2_print_queue ()
- void its2_process_tx_queue ()
- its2_result its2_rebroadcast_net_discover_req (its2_packet *pkt)
- void its2_request_local_addr (uns16 device_id)
- void its2_request_net_addr (uns16 device_id)
- void its2_respond_local_addr (uns16 device_id)
- void its2_respond_net_addr (uns16 device_id)
- its2_result its2_router_handle_association (uns16 pan_id, uns16 its_device_id)
- uns8 its2_router_init (uns16 my_device_id, uns16 network_id)
- void its2_router_process ()
- its2_result its2_router_queue_packet (uns16 device_id, uns8 packet_type, uns8 *data, uns8 data_length, uns8 ack)
- void its2_router_receive_callback (uns16 device_id, uns8 *data, uns8 data_length)
- void its2_setup_io ()

- void its2_transmit (queued_item *item)
- void its2_transmit_status_callback (its2_packet *pkt, uns8 status)

Variables

- bit debug_module
- its2_state state

Detailed Description

Define Documentation

#define ITS2_FLAG_ACK 0x01

#define ITS2_FLAG_DELETED 0xff

#define ITS2_FLAG_NO_ACK 0x02

#define ITS2_NO_AVAILABLE_SLOTS 0xff

#define ITS2_STATUS_QUEUED 0x00

#define ITS2_STATUS_TX_QUEUE_FULL 0x01

#define ITS2_TX_STATUS_NEXT_HOP_UNKNOWN 0x04

#define ITS2_TX_STATUS_NO_ACK 0x02

#define ITS2_TX_STATUS_NO_ROUTE 0x01

#define ITS2_TX_STATUS_REMOTE_NO_ACK 0x03

#define ITS2_TX_STATUS_SUCCESS 0x00

#define ITS2_UPDATE_ROUTE_FAIL 0x01

#define ITS2_UPDATE_ROUTE_SUCCESS 0x00

#define POS_DEST_H 0x0a

#define POS_DEST_L 0x09

#define POS_HOP_COUNT 0x0d

#define POS_KEY1 0x00

#define POS_KEY2 0x01

#define POS_LENGTH_HEADER 0x02

#define POS_MAX_HOP_COUNT 0x0b

#define POS_NETWORK_H 0x06

#define POS_NETWORK_L 0x05

#define POS_NUM_ROUTES 0x0c

#define POS_PKT_TYPE 0x03

#define POS_ROUTE_START 0x0e

#define POS_SEQUENCE 0x04

#define POS_SOURCE_H 0x08

#define POS_SOURCE_L 0x07

#define QS_ACK_RECEIVED 0x04

#define QS_READY_TO_SEND 0x02

#define QS_ROUTING_FAILED 0x06

#define QS_SENT 0x05

#define QS_WAITING_ON_ACK 0x03

#define QS_WAITING_ON_LOCAL_ADDR 0x00

#define QS_WAITING_ON_NETWORK_ADDR 0x01

#define REMOTE_DEVICE 0xffff

Typedef Documentation

typedef enum _its2_result its2_result

typedef enum _its2_state its2_state

Enumeration Type Documentation

enum _its2_result

Enumerator:

RESULT_SUCCESSFUL
RESULT_FAILED
ITEM_QUEUED
QUEUE_FULL
ROUTING_TOO_MANY_HOPS
NEXT_HOP_NOT_LOCAL

```
150                   { RESULT_SUCCESSFUL, RESULT_FAILED, ITEM_QUEUED, QUEUE_FULL,
151                     ROUTING_TOO_MANY_HOPS, NEXT_HOP_NOT_LOCAL };
```

enum _its2_state

Enumerator:

STATE_STARTUP
STATE_RUNNING
STATE_SEARCHING
STATE_ASSOCIATED
STATE_UNASSOCIATED

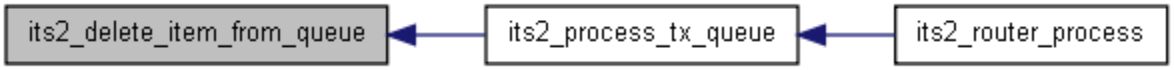
```
154                 {
155     STATE_STARTUP,
156     STATE_RUNNING,
157     STATE_SEARCHING,
158     STATE_ASSOCIATED,
159     STATE_UNASSOCIATED,
160 };
```

Function Documentation

void its2_delete_item_from_queue (queued_item * *item*)

```
1025                                                    {
1026     if (item->data_length > 0) {
1027         free((void *)item->data);
1028     }
1029     item->flag = ITS2_FLAG_DELETED;
1030 }
```

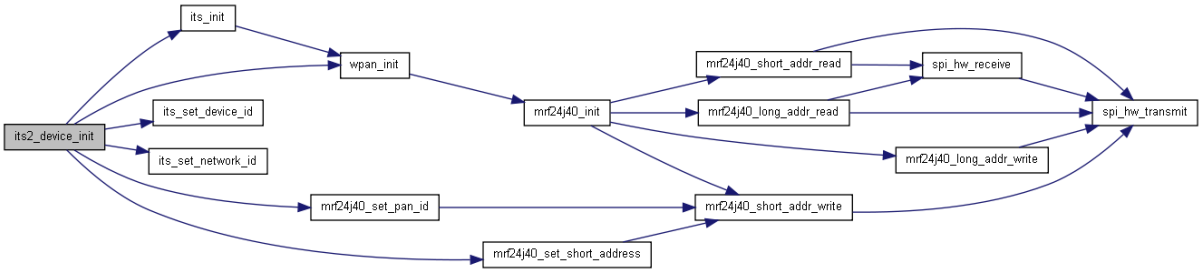
Here is the caller graph for this function:



its2_result its2_device_init (uns16 *my_device_id*, uns16 *network_id*)

```
1136                                                                    {
1137
1138 // ITS_EA is set in config.h
1139
1140 // uns8 my_ea[8] = ITS_EA;
1141
1142     debug_str("Device startup\n");
1143
1144     its_init();
1145     wpan_init();
1146
1147     its_set_device_id(my_device_id);
1148     its_set_network_id(network_id); // network_id = controller_id = pan_id in mode 1
1149
1150     mrf24j40_set_pan_id(network_id);
1151     mrf24j40_set_short_address(my_device_id);   // same as device id
1152     //mrf24j40_set_extended_address(&my_ea);
1153
1154     state = STATE_UNASSOCIATED;
1155 }
```

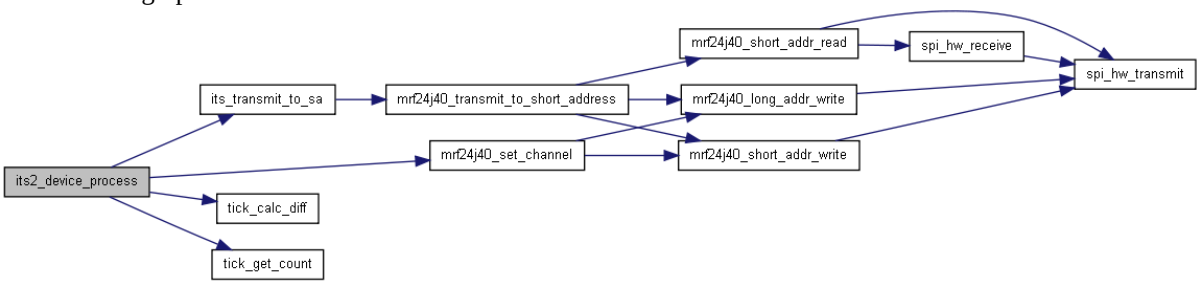
Here is the call graph for this function:



void its2_device_process ()

```
1163                            {
1164
1165 uns16 test_tick;
1166
1167     // are we searching?
1168     // has time limit expired?
1169     // go on to next channel
1170     // send its association request to everyone
1171     if (state == STATE_SEARCHING) {
1172         test_tick = tick_get_count();
1173         if (tick_calc_diff(tick_marker, test_tick) >= 250) {    // 250 = 1/4 second
1174             tick_marker = test_tick;
1175             if (channel == 0) {
1176                 channel = MRF_FIRST_CHANNEL;
1177             } else {
1178                 if (channel == MRF_LAST_CHANNEL) {
1179                     // Didn't find the controller
1180                     state = STATE_UNASSOCIATED;
1181                     debug_str("Didn't find controller\n");
1182                 } else {
1183                     channel++;
1184                 }
1185             }
1186             // change channel
1187             debug_var("Trying channel ", channel);
1188             debug_nl();
1189             mrf24j40_set_channel(channel);
1190             // send broadcast association request
1191             its_transmit_to_sa(/* pan id*/ 0xffff, /* sa */ 0xffff, /* dest device */ 0xffff,
1192                                ITS_ASSOC_REQ, /* nil data */ 0, /* zero length */ 0);
1193         } // timer for another channel
1194     } // we're searching
1195
1196 }
```

Here is the call graph for this function:



void its2_device_receive_callback (uns8 * *data*, uns8 *data_length*)

Here is the caller graph for this function:

its2_result its2_device_transmit (uns16 *device_id*, uns8 * *data*, uns8 *data_length*)

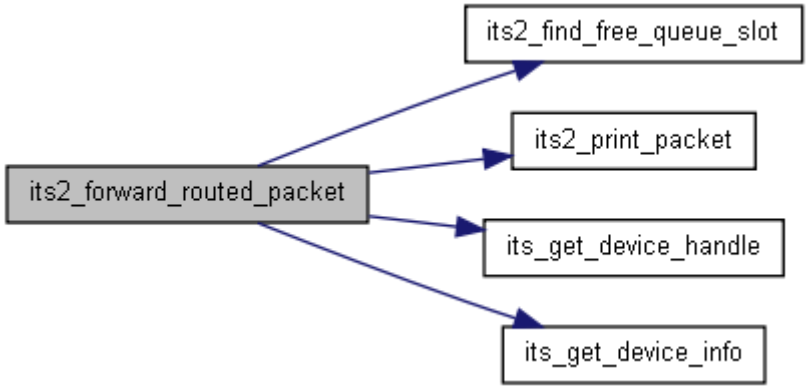
its2_result its2_find_coordinator ()

its2_result its2_forward_routed_packet (its2_packet * *pkt*, uns8 * *data*, uns8 *data_length*)

```
653                                                                                 {
654
655 uns8 queue_slot;
656 queued_item *item;
657 void    *p;
658
659     queue_slot = its2_find_free_queue_slot();
660
661     if (queue_slot != ITS2_NO_AVAILABLE_SLOTS) {
662
663         debug_var("<Qed>", queue_slot);
664         debug_str(" data lengh=");
665         debug_int(data_length);
666         debug_spc();
667
668         item = &its2_tx_queue[queue_slot];
669         item->flag = ITS2_FLAG_NO_ACK;  // it's ours!
670         memcpy(/*dst*/  (void *)&item->packet, // copy it in
671                 /*src*/  (void *)pkt,
672                 /*len*/  11 + (pkt->num_routes * 2));
673
674         its2_print_packet(&item->packet);
675
676         item->packet.hop_count++;
677
678         // If hop_count now == num_routes then we are done routing
679         // and have to go final (local) destination.
680         // Otherwise, forward to next hop.
681
682         if (item->packet.hop_count == item->packet.num_routes) {
683             // Final desintation
684             debug_str(" going to final=");
685             item->dest_its_device_id = item->packet.its_dest_id;
686         } else {
687             // Forward to next hop
688             debug_str(" going to next hop=");
689             item->dest_its_device_id = item->packet.routers[item->packet.hop_count];
690         }
691
692         debug_int_hex_16bit(item->dest_its_device_id);
693
694         item->sent_count = 0;
695         item->dest_device_handle = its_get_device_handle(item->dest_its_device_id);
696
697         // Do we know about it already?
698         if (item->dest_device_handle == ITS_DEVICE_NONE) {
699             // Not in our routing table. Let's see if it's local
700             debug_str(" unknown. Looking for local. ");
701             item->status = QS_WAITING_ON_LOCAL_ADDR;
702         } else {
703             // Check that it is really local
704             its_device_info *device_info = its_get_device_info(item->dest_device_handle);
705
706             if (device_info->addr.remote.remote_indicator == 0xffff) {
707                 debug_str(" Not local! Abandon ship ");
```

```
708                 item->flag = ITS2_FLAG_DELETED; // delete it
709                 return NEXT_HOP_NOT_LOCAL;  // let our caller know
710                 // !! we should send something back to originator
711                 // to say that we couldn't route it
712             } else {
713                 debug_str(" found locally. ");
714                 item->status = QS_READY_TO_SEND;
715             }
716         }
717         item->data_length = 0;
718         item->data = NULL;
719 
720         // copy data...
721 
722         if (data_length > 0) {
723             p = alloc(data_length);
724 
725             memcpy(/*dst*/  p, // copy it in
726                 /*src*/  (void *)data,
727                 /*len*/  data_length);
728             item->data =  (uns8*)p;
729         }
730 
731         item->data_length = data_length;
732 
733         return ITEM_QUEUED;
734     } else {
735         return QUEUE_FULL;
736     }
737 
738 }
```

Here is the call graph for this function:

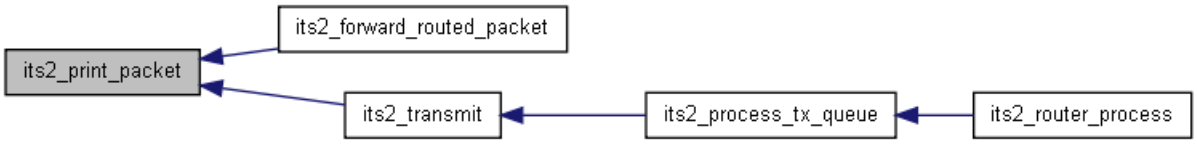


void its2_print_packet (its2_packet * *pkt*)

```
580                                         {
581 
582     debug_str("(pkt type=");
583     debug_int(pkt->packet_type);
584     debug_str(" seq=");
585     debug_int(pkt->sequence);
586     debug_str(" net=");
587     debug_int_hex_16bit(pkt->its_network_id);
588     debug_str(" src=");
589     debug_int_hex_16bit(pkt->its_source_id);
590     debug_str(" dst=");
591     debug_int_hex_16bit(pkt->its_dest_id);
592     debug_str(" max_hop=");
593     debug_int(pkt->max_hop_count);
```

```
594     debug_str(" num_rts=");
595     debug_int(pkt->num_routes);
596     debug_str(" hop cnt=");
597     debug_int(pkt->hop_count);
598 
599     uns8 count;
600     debug_str(" Rts=");
601     for (count = 0; count < pkt->num_routes; count++) {
602         if (count > ITS2_MAX_HOP_COUNT) {
603             debug_str(" Too many routes! ");
604             break;
605         } else {
606             debug_putc(' ');
607             debug_int_hex_16bit(pkt->routers[count]);
608         }
609     }
610     debug_str(") ");
611 }
```

Here is the caller graph for this function:



void its2_print_queue ()

```
1106                         {
1107 
1108 uns8 count;
1109 queued_item *item;
1110 
1111         debug_nl();
1112         for (count = 0; count < ITS2_TX_QUEUE_SIZE; count++) {
1113             item = &its2_tx_queue[count];
1114             if (item->flag != ITS2_FLAG_DELETED) {
1115                 debug_var("Q:",count);
1116                 debug_var(" FL:", item->flag);
1117                 debug_var(" SRC:", item->packet.its_source_id);
1118                 debug_var(" DST:", item->packet.its_dest_id);
1119                 debug_var(" STAT:", item->status);
1120                 debug_var(" SENT:", item->sent_count);
1121                 debug_var(" TICK:", item->tick_sent);
1122                 debug_nl();
1123             }
1124         }
1125         debug_var("Current tick", tick_get_count());
1126         debug_nl();
1127 }
```

Here is the call graph for this function:

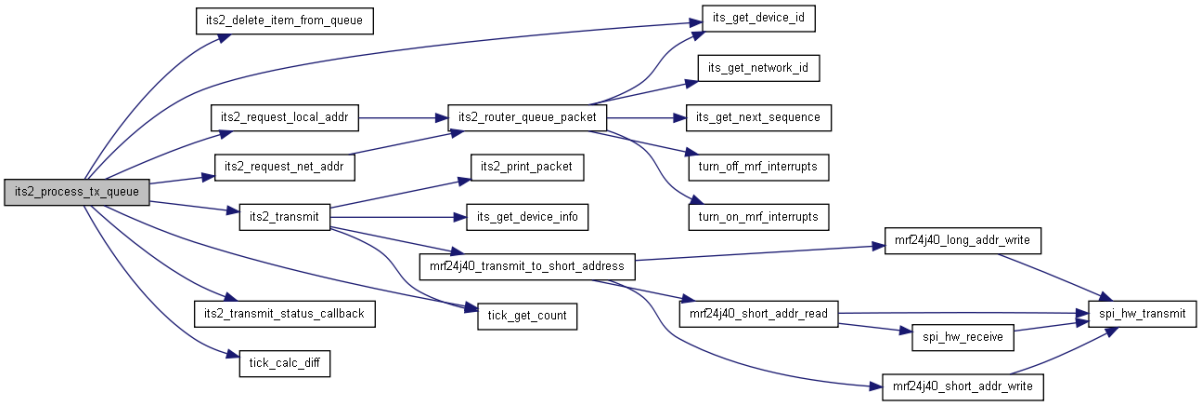
void its2_process_tx_queue ()

```
904                               {
905 
906 uns16       current_tick;
907 uns8        count;
```

```
908 queued item *item;
909 uns8        flag;
910 uns8        status;
911 uns16       my_id;
912     //debug_str("\n[PQ]\n");
913 
914     current_tick = tick get count();
915     my_id = its get device id();
916 
917     //clear_bit(intcon3, INT1IE);       // disable int1
918 
919 
920     for (count = 0; count < ITS2_TX_QUEUE_SIZE; count++) { // search for a packet
921         item = &its2 tx queue[count];   // grab item to save code
922         flag = item->flag;
923         status = item->status;
924         if (flag == ITS2 FLAG DELETED) {
925             continue;
926         }
927         //debug_var("\nQ-", count);
928         if (status == QS WAITING ON LOCAL ADDR) {
929             if (item->sent count == ITS2_SEND_MAX_TRIES) { // Sent too many times
930                 if (item->packet.its source id == my id) {  // from me?
931                     debug str(" Tried local, now time for net search");
932                     item->sent count = 0;
933                     item->status = QS WAITING ON NETWORK ADDR;
934                 } else {    // Must have been a network route
935                     // but we failed to find the next hop locally
936                     // !! Should return something here to sender
937                     debug str("Next hop not available locally. Giving up.");
938                     its2 transmit status callback(&item->packet,
ITS2 TX STATUS NEXT HOP UNKNOWN);
939                     its2 delete item from queue(item);
940                 }
941             } else if ((item->sent count == 0) || (tick calc diff(item->tick sent,
current_tick)
942                      > ITS2_RESEND_TICK_DELAY)) {
943 
944                 debug str(" REQ local address for q ");
945                 debug int(count);
946                 debug str(" dev ");
947                 debug int hex 16bit(item->dest its device id);
948 
949                 item->tick sent = current tick;
950                 item->sent count++;
951                 its2 request local addr(item->dest its device id);
952                 item->tick sent = current_tick; // set tick count to current
953             } else {
954                 debug str(" Local waiting for ");
955                 debug int(item->tick sent);
956             }
957         } else
958         if (status == QS WAITING ON NETWORK ADDR) {
959             if (item->sent count == ITS2_SEND_MAX_TRIES) { // Sent too many times
960                 // !! Need to issue callback here
961                 debug str("No addr. killing");
962                 its2 transmit status callback(&item->packet, ITS2 TX STATUS NO ROUTE);
963                 its2 delete item from queue(item);
964             } else {
965                 //debug_str("Wait for ");
966                 //debug int(item->tick sent);
967                 //debug_str("+ 5000 current=");
968                 //debug_int(current_tick);
969                 if (tick calc diff(item->tick sent, current_tick)
970                      > ITS2_RESEND_TICK_DELAY) {
971                     item->tick sent = current_tick;
972                     item->sent count++;
973                     its2 request net addr(item->dest its device id);
974                 }
975             }
976         } else
```

```
977         if (status == QS_READY_TO_SEND) {
978             if (!its2_transmitting) {
979                 debug_str("<TX!>");
980                 its2_transmit(item);
981                 debug_str("<Done>");
982 
983             }
984         } else
985         if (status == QS_WAITING_ON_ACK) {
986             if (item->sent_count > ITS2_SEND_MAX_TRIES) { // Sent too many times
987                 // !! give up, go home... FAIL! No address
988                 // issue call back
989                 // kill from queue
990                 debug_str("Too many. killing.");
991                 its2_transmit_status_callback(&item->packet, ITS2_TX_STATUS_NO_ACK);
992                 its2_delete_item_from_queue(item);
993             }
994             if (tick_calc_diff(item->tick_sent, current_tick)
995                     > ITS2_RESEND_TICK_DELAY) {
996                 if (!its2_transmitting) {
997                     debug_str("<RETRYTX!>");
998                     its2_transmit(item);
999                     debug_str("<Done>");
1000                 }
1001             }
1002         }
1003         if (status == QS_ACK_RECEIVED) {
1004             // !! call back here
1005             its2_transmit_status_callback(&item->packet, ITS2_TX_STATUS_SUCCESS);
1006 
1007             its2_delete_item_from_queue(item);
1008         } else
1009         if (status == QS_SENT) {
1010             debug_str(" <SENT del> ");
1011             its2_transmit_status_callback(&item->packet, ITS2_TX_STATUS_SUCCESS);
1012             its2_delete_item_from_queue(item);
1013         } else
1014         if (status == QS_ROUTING_FAILED) {
1015             debug_str(" <RFAIL del> ");
1016             its2_transmit_status_callback(&item->packet, ITS2_TX_STATUS_NO_ROUTE);
1017             its2_delete_item_from_queue(item);
1018         }
1019     }
1020 
1021     //set_bit(intcon3, INT1IE);     // enable int1
1022 
1023 }
```

Here is the call graph for this function:

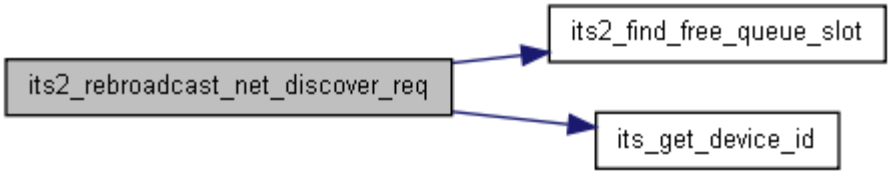


Here is the caller graph for this function:

its2_result its2_rebroadcast_net_discover_req (its2_packet * *pkt*)

```
613                                                                 {
614
615 uns8 queue_slot;
616 queued_item *item;
617
618     if (pkt->hop_count == pkt->max_hop_count) {
619         debug_str("<Can't re-b: max hop count exceeded!>");
620         return ROUTING_TOO_MANY_HOPS;
621     }
622     queue_slot = its2_find_free_queue_slot();
623     if (queue_slot != ITS2_NO_AVAILABLE_SLOTS) {
624
625         debug_var("<Qed>", queue_slot);
626         item = &its2_tx_queue[queue_slot];
627
628         item->flag = ITS2_FLAG_NO_ACK;  // it's ours!
629         memcpy(/*dst*/  (void *)&item->packet, // copy it in
630                 /*src*/  (void *)pkt,
631                 /*len*/  11 + (pkt->num_routes * 2));
632         item->dest_its_device_id = 0xffff;
633         item->packet.routers[item->packet.hop_count] = its_get_device_id();
634         item->packet.hop_count++;
635         item->packet.num_routes++;
636         item->sent_count = 0;
637         item->dest_device_handle = ITS_DEVICE_NONE; // it's a broadcast
638         item->data_length = 0;
639         item->data = NULL;
640
641
642         item->status = QS_READY_TO_SEND;
643
644         its2_add_to_seen_list(item->packet.its_source_id, item->packet.sequence);
645
646         return ITEM_QUEUED;
647     } else {
648         return QUEUE_FULL;
649     }
650
651 }
```

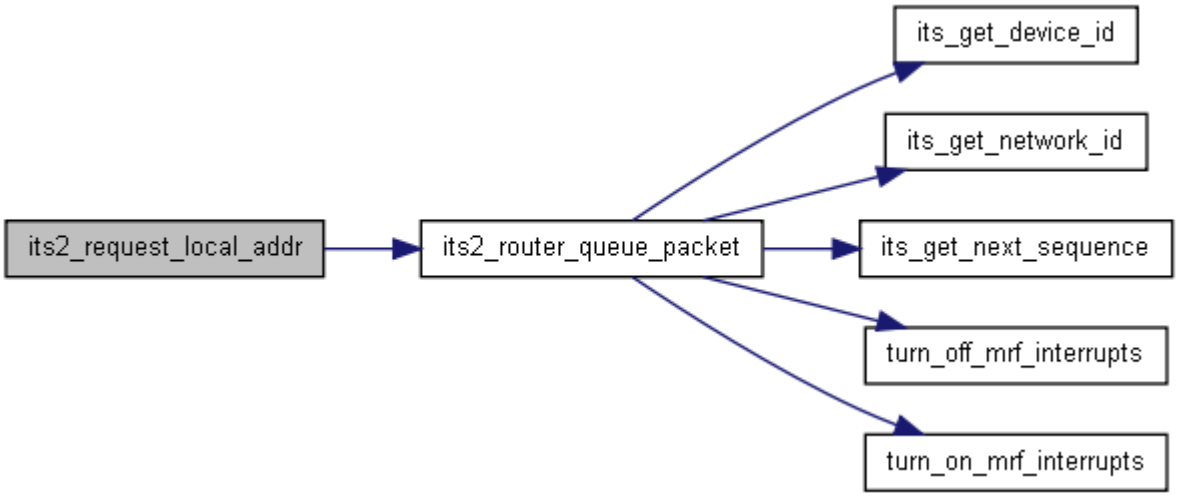
Here is the call graph for this function:



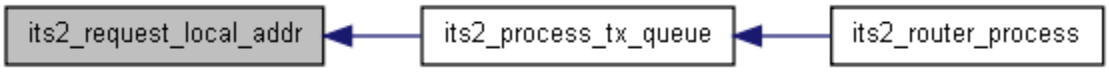
void its2_request_local_addr (uns16 *device_id*)

```
779                                                 {
780     its2_router_queue_packet(device_id, ITS_LOCAL_DISCOVER_REQ, NULL, 0, ITS2_FLAG_NO_ACK);
781 }
```

Here is the call graph for this function:



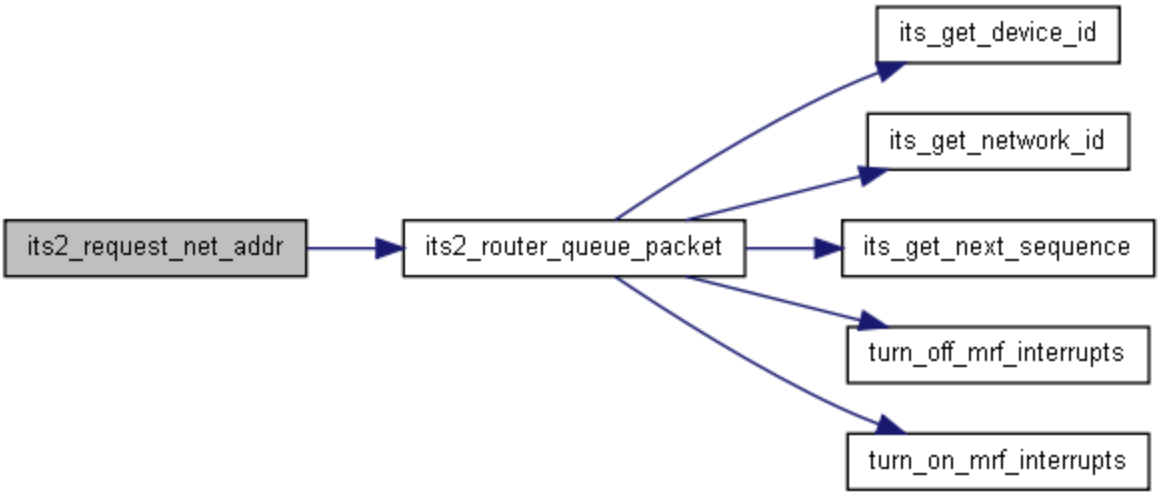
Here is the caller graph for this function:



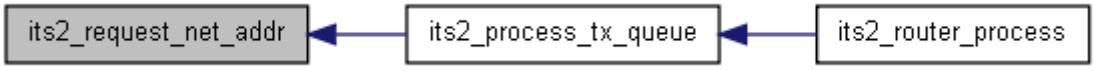
void its2_request_net_addr (uns16 *device_id*)

```
792                                             {
793     its2_router_queue_packet(device_id, ITS_NET_DISCOVER_REQ, NULL, 0, ITS2_FLAG_NO_ACK);
794 }
```

Here is the call graph for this function:



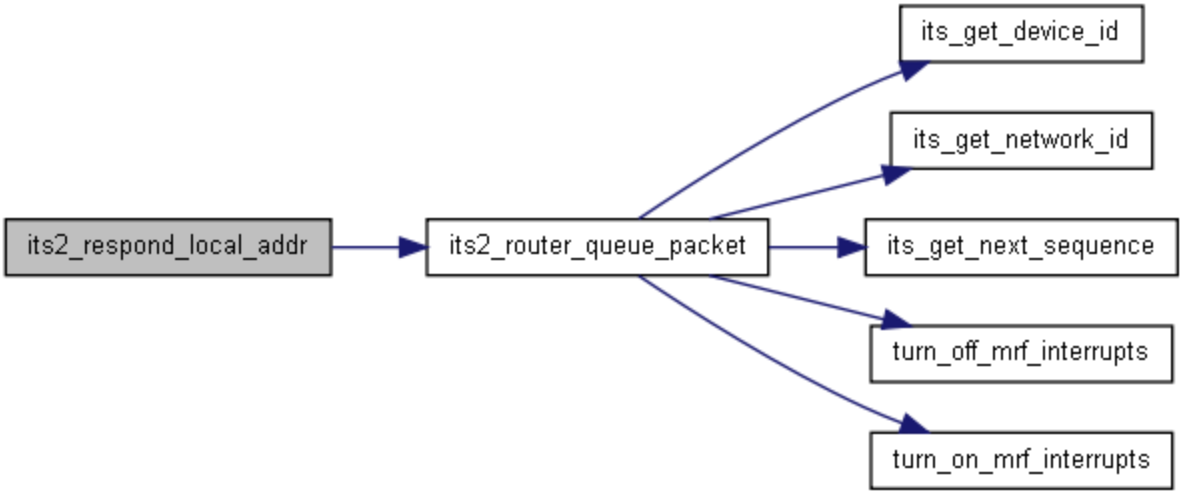
Here is the caller graph for this function:



void its2_respond_local_addr (uns16 *device_id*)

```
783                                                 {
784
785     if ((device_id != 0x0001) || !debug_module) {
786         its2_router_queue_packet(device_id, ITS_LOCAL_DISCOVER_RES, NULL, 0,
ITS2_FLAG_NO_ACK);
787     } else {
788         debug_str("IGNORE");
789     }
790 }
```

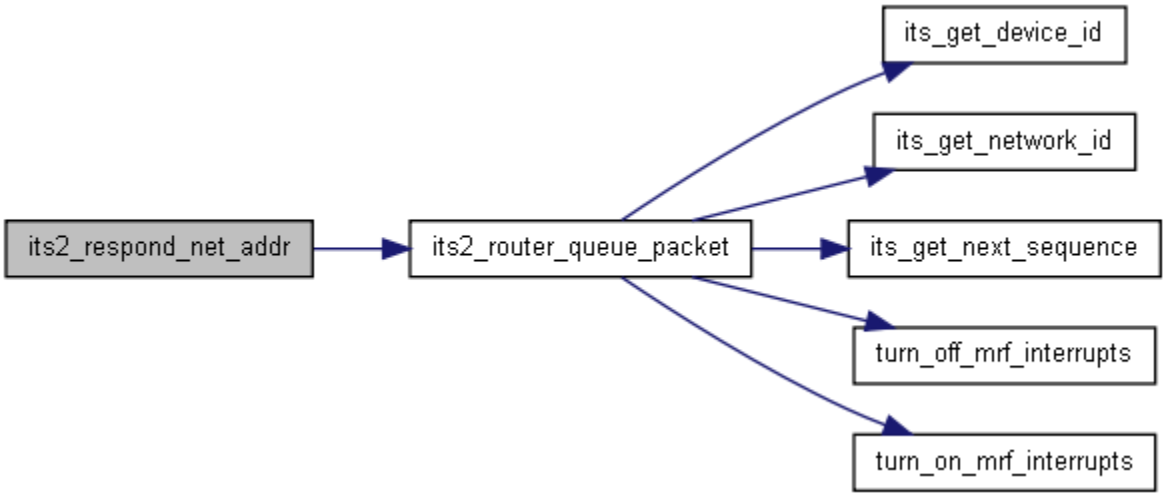
Here is the call graph for this function:



void its2_respond_net_addr (uns16 *device_id*)

```
796                                             {
797     its2_router_queue_packet(device_id, ITS_NET_DISCOVER_RES, NULL, 0, ITS2_FLAG_NO_ACK);
798 }
```

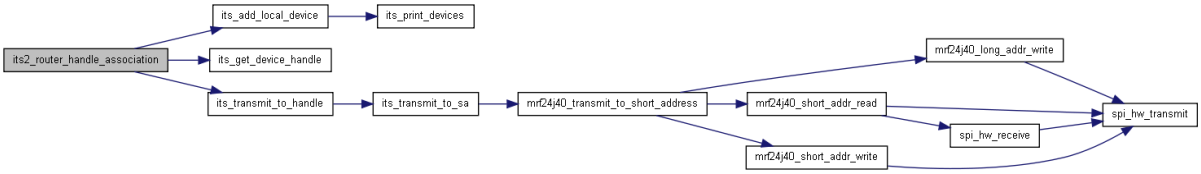
Here is the call graph for this function:



its2_result its2_router_handle_association (uns16 *pan_id*, uns16 *its_device_id*)

```
118                                                                                 {
119     // see if this device is in our list, if not, add it
120 its_device_handle device_handle;
121
122     device_handle = its_get_device_handle(its_device_id);
123
124     if (device_handle == ITS_DEVICE_NONE) {
125         // in mode 1 we assume that
126         device_handle = its_add_local_device(/* device */ its_device_id, pan_id, /* short addr
*/ its_device_id);
127         if (device_handle == ITS_DEVICE_NONE) {
128             // panic!
129             debug_str("Too many devices! association failed");
130             return RESULT_FAILED;
131         } else {
132             // send association response
133             its_transmit_to_handle(device_handle, ITS_ASSOC_RES, /* nil data */ 0, /* zero
length */ 0);
134             return RESULT_SUCCESSFUL;
135             // although strictly speaking we should wait for the ack...
136         }
137
138     }
139 }
```

Here is the call graph for this function:

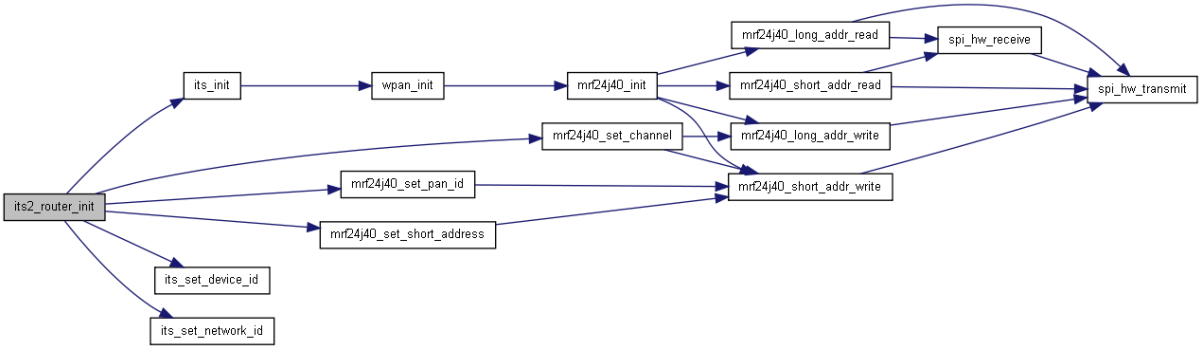


uns8 its2_router_init (uns16 *my_device_id*, uns16 *network_id*)

```
743                                                                 {
744
745 uns8 lowest_channel_ed;
746 uns8 count;
747
748 //uns8 my_ed[8] = ITS_EA;
749
750     its2_seen_index = 0;
751
752     for (count=0; count < ITS2_SEEN_LIST_SIZE; count++) {
753         its2_seen_list[count].its_source_id = 0xffff;
754     }
755
756     for (count = 0; count < ITS2_TX_QUEUE_SIZE; count++) {
757         its2_tx_queue[count].flag  = ITS2_FLAG_DELETED;
758     }
759
760     its_init();
761
762     its_set_device_id(my_device_id);
763     its_set_network_id(network_id); // network id = controller id = pan id in mode 1
764
765     mrf24j40_set_pan_id(network_id);
766     mrf24j40_set_short_address(my_device_id);   // same as device id
767     //mrf24j40_set_extended_address(&my_ed);
768
769     debug_str("Router startup\n");
770
```

```
771     debug_var("Chosing channel", 15);
772     debug_nl();
773 
774     // Now set ourselves up on this channel
775     mrf24j40_set_channel(15);
776 
777 }
```

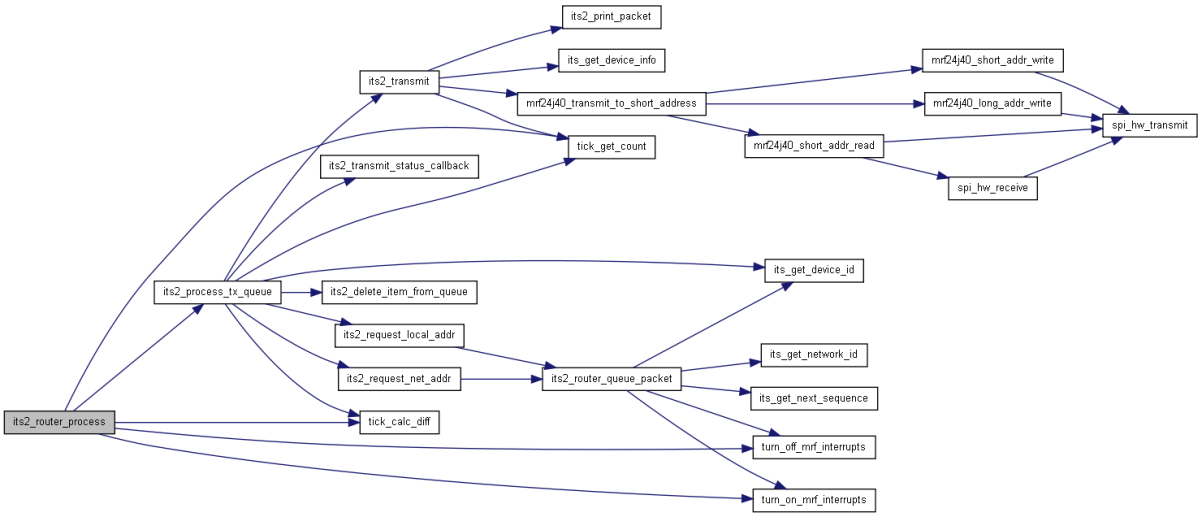
Here is the call graph for this function:



void its2_router_process ()

```
808                             {
809 
810     turn_off_mrf_interrupts();
811 
812     its2_process_tx_queue();
813 
814 uns16 test_tick;
815     test_tick = tick_get_count();
816     if (tick_calc_diff(tick_marker, test_tick) >= state_timeout) {
817     // timeout
818     }
819     turn_on_mrf_interrupts();
820 
821 }
```

Here is the call graph for this function:

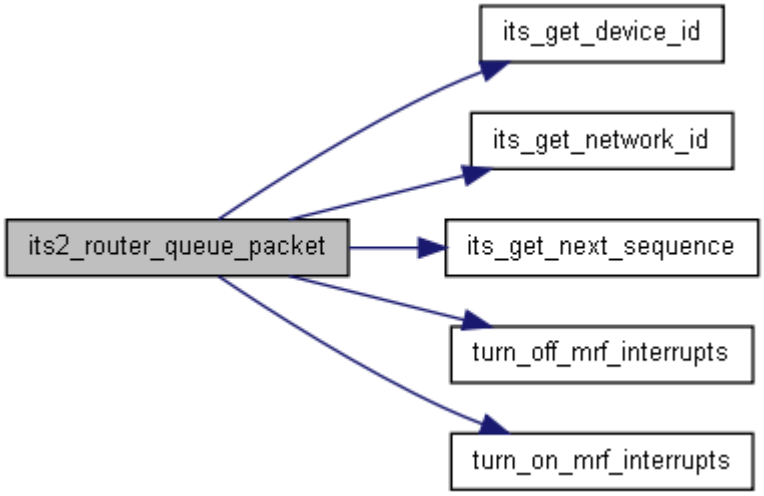


its2_result its2_router_queue_packet (uns16 *device_id*, uns8 *packet_type*, uns8 * *data*, uns8 *data_length*, uns8 *ack*)

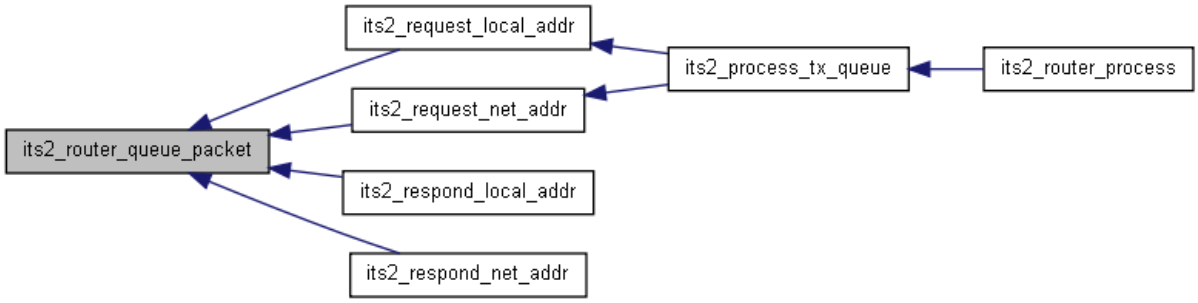
```
823
{
824
825 // return result - queue handle?
826
827 uns8 found;
828 void    *p;
829 queued_item *item;
830 uns8 count;
831
832 turn_off_mrf_interrupts();
833
834 debug_str("\n[its2_router_queue_packet]\n");
835
836     // find slot
837     found = 0;
838     for (count = 0; count < ITS2_TX_QUEUE_SIZE; count++) {
839         if (its2_tx_queue[count].flag  == ITS2_FLAG_DELETED) {
840             found = 1;
841             break;
842         }
843     }
844     if (found) {
845         debug_var("Qed: ", count);
846         item = &its2_tx_queue[count];
847         item->flag = ack;   // it's ours!
848         item->sent_count = 0;
849
850         //item->packet.key_1 = 'I';
851         //item->packet.key_2 = 'T';
852         // item->length header; // unknown as yet
853         item->packet.packet_type = packet_type;
854         item->packet.sequence = its_get_next_sequence();
855         item->packet.its_network_id = its_get_network_id();
856         item->packet.its_source_id = its_get_device_id();
857         item->packet.its_dest_id = device_id;
858         if ((packet_type == ITS_LOCAL_DISCOVER_REQ) ||
859             (packet_type == ITS_NET_DISCOVER_REQ)) {
860             item->dest_its_device_id = 0xffff;
861         } else {
862             item->dest_its_device_id = device_id;
863         }
864         item->packet.max_hop_count = ITS2_MAX_HOP_COUNT;
865         item->packet.hop_count = 0;
866         // item->packet.num_routes;
867
868         // uns16 routers[5];
869             // create some memory to whack the data in
870         if (data_length > 0) {
871             p = alloc(data_length);
872
873             memcpy(/*dst*/  p, // copy it in
874                 /*src*/  (void *)data,
875                 /*len*/  data_length);
876             item->data =  (uns8*)p;
877         }
878
879         item->data_length = data_length;
880
881         // Here we will update the route and also set
882         // dest_device_handle (which may not be the same as the
883         // eventual destination, since we may be going through
884         // other hops or doing a broadcast)
885         uns8 routing_result = its2_update_route(item);
886
887         turn_on_mrf_interrupts();
888
```

```
889         if (routing_result == ITS2_UPDATE_ROUTE_FAIL) {
890             debug_str("RoUtE fail");
891             return ROUTING_TOO_MANY_HOPS;
892         } else {
893             debug_str("qdone ");
894             return ITEM_QUEUED;
895         }
896     } else {
897         turn_on_mrf_interrupts();
898         debug_str(" Full! ");
899         return QUEUE_FULL;
900     }
901 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void its2_router_receive_callback (uns16 *device_id*, uns8 * *data*, uns8 *data_length*)

void its2_setup_io ()

```
95                      {
96     wpan_setup_io();
97 }
```

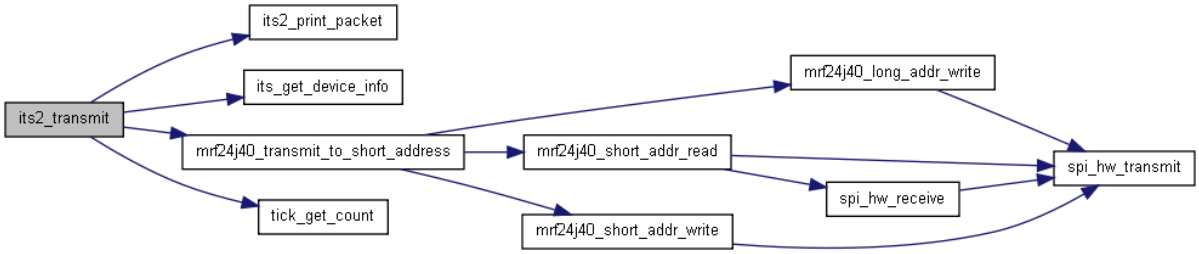
Here is the call graph for this function:

void its2_transmit (queued_item * *item*)

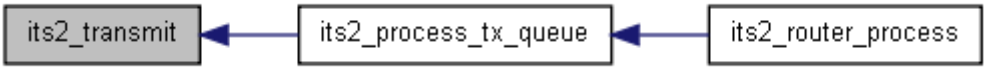
```
1035                                        {
1036
1037 uns8 buffer[50];
1038 uns8 count;
1039 uns8 index;
1040 its_device_info *device_info;
1041
1042     its2_transmitting = 1;
1043
1044     debug_str(" X: ");
1045     its2_print_packet(&item->packet);
1046     //p = (void *)alloc(item->data_size + ??);
1047     buffer[0] = 'I';
1048     buffer[1] = 'T';
1049     memcpy(/* dst */ (void *)&buffer[3], /* src */ (void *)&item->packet, 11 );
1050     // now need to copy routes
1051     index = 11 + 3;
1052     for (count = 0; count < item->packet.num_routes; count++) {
1053         buffer[index++] = item->packet.routers[count] & 0xff;
1054         buffer[index++] = item->packet.routers[count] >> 8;
1055     }
1056
1057     buffer[2] = index - 3;  // update header with length
1058
1059     // now copy data
1060     for (count = 0; count < item->data_length; count++) {
1061         buffer[index++] = item->data[count];
1062     }
1063     debug_str(" To=");
1064     debug_int_hex_16bit(item->dest_its_device_id);
1065     debug_str(" bytes=");
1066     debug_int(index);
1067
1068     item->sent_count++;
1069     item->tick_sent = tick_get_count();
1070     if (item->flag == ITS2_FLAG_ACK) {  // ack
1071         item->status = QS_WAITING_ON_ACK;
1072     } else {
1073         item->status = QS_SENT;
1074     }
1075     if (item->dest_device_handle != ITS_DEVICE_NONE) {  // we have handle
1076         debug_var(" Handle ", item->dest_device_handle);
1077         device_info = its_get_device_info(item->dest_device_handle);
1078         if (device_info != NULL) {
1079             debug_str(" Pan ");
1080             debug_int_hex_16bit(device_info->addr.local.pan_id);
1081             debug_str(" SA ");
1082             debug_int_hex_16bit(device_info->addr.local.short_address);
1083         } else {
1084             debug_str("!!Strange!! - NO DEVICE INFO!!\n");
1085         }
1086         mrf24j40_transmit_to_short_address(FRAME_TYPE_DATA,
device_info->addr.local.pan_id,
1087                                            device_info->addr.local.short_address,
1088                                            &buffer, index, MRF_ACK);
1089         debug_str(" Sent bytes ");
1090         debug_int(index);
1091         debug_putc(' ');
1092     }
1093     else {  // it's a broadcast
1094         debug_str(" It's a broadcast");
1095         mrf24j40_transmit_to_short_address(FRAME_TYPE_DATA, 0xffff,
1096                                            0xffff, &buffer, index, MRF_NO_ACK);
1097     }
1098
```

```
1099 }
```

Here is the call graph for this function:

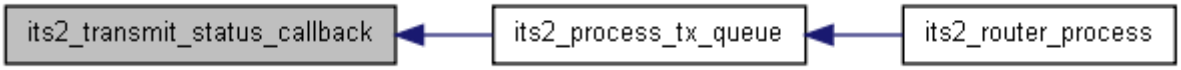


Here is the caller graph for this function:



void its2_transmit_status_callback (its2_packet * *pkt*, uns8 *status*)

Here is the caller graph for this function:



Variable Documentation

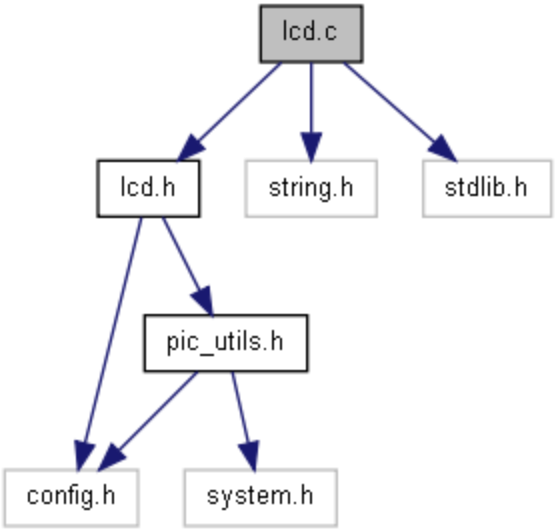
bit debug_module

its2_state state

---

lcd.c File Reference

Include dependency graph for lcd.c:



Functions

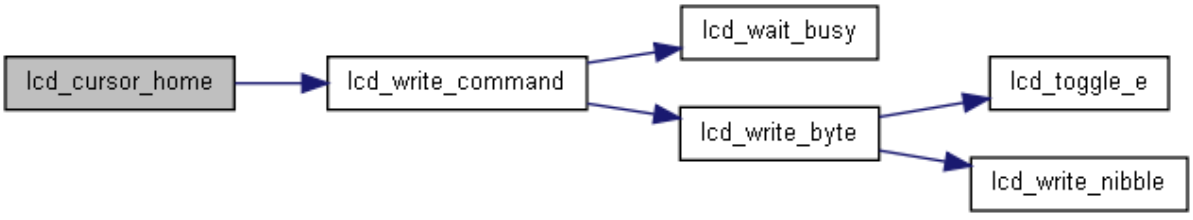
- void lcd_cursor_home ()
- void lcd_init ()
- *Initialise LCD ready for display.* void lcd_set_cgram_pos (uns8 x)
- void lcd_set_ddram_pos (uns8 x)
- void lcd_setup ()
- *Setup port and pins to talk to LCD.* void lcd_toggle_e ()
- void lcd_wait_busy ()
- *Wait while LCD is busy.* void lcd_write_byte (uns8 data)
- void lcd_write_command (uns8 data)
- *Sends a command to the LCD.* void lcd_write_data (uns8 data)
- *Send one byte of data to the LCD.* void lcd_write_data_int (uns16 i)
- *Print a 16 bit integer the the LCD.* void lcd_write_data_str (char *str)
- *Print a string to the LCD.* void lcd_write_nibble (uns8 data)

Function Documentation

void lcd_cursor_home ()

```
204                         {
205     lcd_write_command(LCD_CLEAR_DISP);
206 }
```

Here is the call graph for this function:

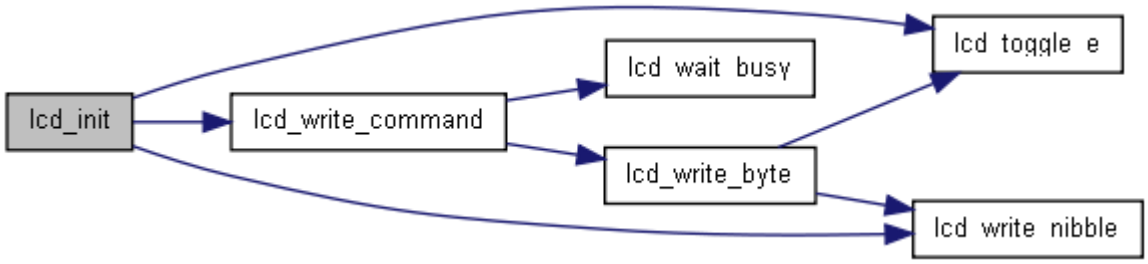


void lcd_init ()

Configures LCD for 4 bit operation and gets ready for displaying text

```
84                  {
85
86     delay_ms(16);
87     lcd_write_nibble(0x03);
88     lcd_toggle_e();
89     delay_ms(5);
90     lcd_write_nibble(0x03);
91     lcd_toggle_e();
92     delay_ms(1);
93     lcd_write_nibble(0x03);
94     lcd_toggle_e();
95     lcd_write_nibble(0x02); //0b 0010 1000
96     lcd_toggle_e();
97
98     // Now we are in 4bit mode
99
100     lcd_write_command(0b00101000); // 0x28 numlines=1 font=0
101
102     lcd_write_command(0b00001000); // disp off, curs off blink off
103
104     lcd_write_command(0b00000110); // cursor move, inc, no shift
105
106     lcd_write_command(0b00001100); // disp on, curs on blink on
107
108     lcd_write_command(LCD_CLEAR_DISP);
109     lcd_write_command(LCD_RETURN_HOME);
110
111 }
```

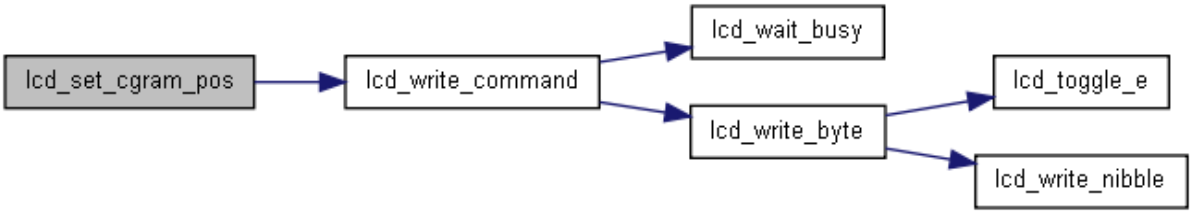
Here is the call graph for this function:



void lcd_set_cgram_pos (uns8 *x*)

```
198                                 {
199     x.7 = 0; // indicate we are setting cgram
200     x.6 = 1;
201     lcd_write_command(x);
202 }
```

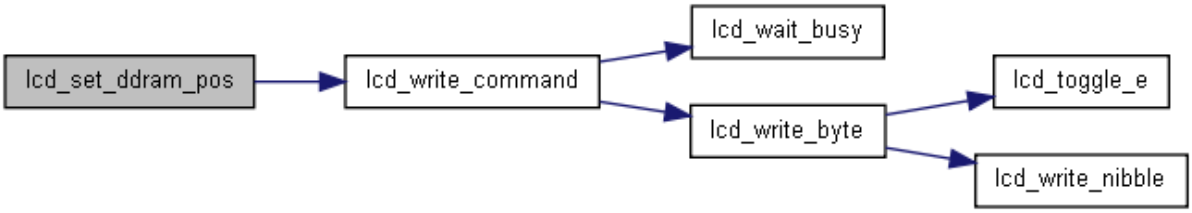
Here is the call graph for this function:



void lcd_set_ddram_pos (uns8 *x*)

```
193                                 {
194     x.7 = 1; // Set MSB bit, which indicates we are setting the RAM address
195     lcd_write_command(x);
196 }
```

Here is the call graph for this function:



void lcd_setup ()

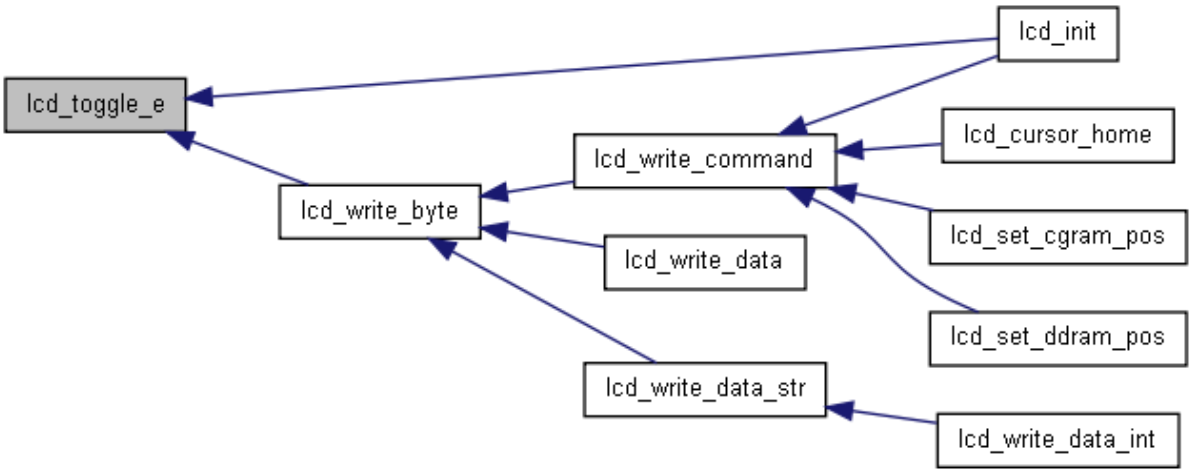
Call this routine first, to set up tris bits correctly to talk to the LCD

```
67                  {
68
69     // Set up tris bits
70     make_output(lcd_e_port, lcd_e_pin);
71     make_output(lcd_rs_port, lcd_rs_pin);
72     make_output(lcd_rw_port, lcd_rw_pin);
73     make_output(lcd_db7_port, lcd_db7_pin);
74     make_output(lcd_db6_port, lcd_db6_pin);
75     make_output(lcd_db5_port, lcd_db5_pin);
76     make_output(lcd_db4_port, lcd_db4_pin);
77
78     clear_pin(lcd_e_port, lcd_e_pin);
79     clear_pin(lcd_rs_port, lcd_rs_pin);
80     clear_pin(lcd_rw_port, lcd_rw_pin);
81 }
```

void lcd_toggle_e ()

```
40                      {
41     set_pin(lcd_e_port, lcd_e_pin);
42     delay_us(50);   // Despite the datasheet, 50us seems to be the right number...
43     clear_pin(lcd_e_port, lcd_e_pin);
44     delay_us(50);   // Despite the datasheet, 50us seems to be the right number...
45 }
```

Here is the caller graph for this function:

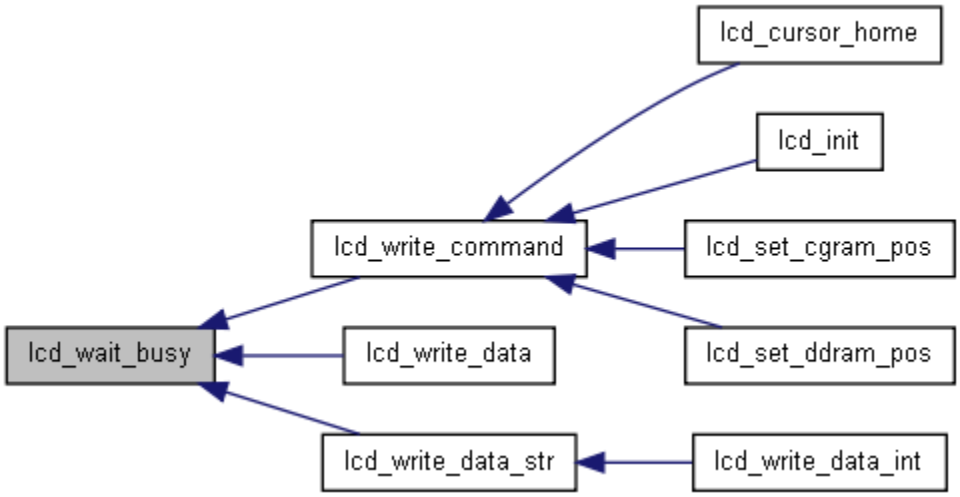


void lcd_wait_busy ()

Internal routine to wait while the LCD is busy and unable to accept more data

```
154                     {
155
156
157     set_bit(tris_array[lcd_db7_port - PORTA], lcd_db7_pin); // db7 input
158     set_bit(tris_array[lcd_db6_port - PORTA], lcd_db6_pin); // db6 input
159     set_bit(tris_array[lcd_db5_port - PORTA], lcd_db5_pin); // db5 input
160     set_bit(tris_array[lcd_db4_port - PORTA], lcd_db4_pin); // db4 input
161
162
163     clear_pin(lcd_rs_port, lcd_rs_pin);
164     set_pin(lcd_rw_port, lcd_rw_pin);
165
166     char counter = 0;
167
168     set_pin(lcd_e_port, lcd_e_pin);
169     // Wait for completion of the operation, with a timeout of ~.5 seconds
170     // LCD d7 is high if the operation is complete.
171     while ((test_pin(lcd_db7_port, lcd_db7_pin) == 1) && counter < 0xF0){
172         clear_pin(lcd_e_port, lcd_e_pin);
173         set_pin(lcd_e_port, lcd_e_pin);
174         delay_us(100);
175         clear_pin(lcd_e_port, lcd_e_pin);
176         //delay_us(100);
177         counter++;
178     }
179
180
181     // Check if the previous loop timed out
182     if (counter == 0xF0) {
183     }
184
185     clear_bit(tris_array[lcd_db7_port - PORTA], lcd_db7_pin); // db7 output
186     clear_bit(tris_array[lcd_db6_port - PORTA], lcd_db6_pin); // db6 output
187     clear_bit(tris_array[lcd_db5_port - PORTA], lcd_db5_pin); // db5 output
188     clear_bit(tris_array[lcd_db4_port - PORTA], lcd_db4_pin); // db4 output
189
190     return;
191 }
```

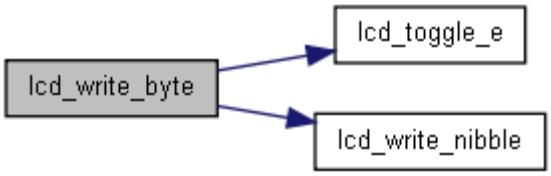
Here is the caller graph for this function:



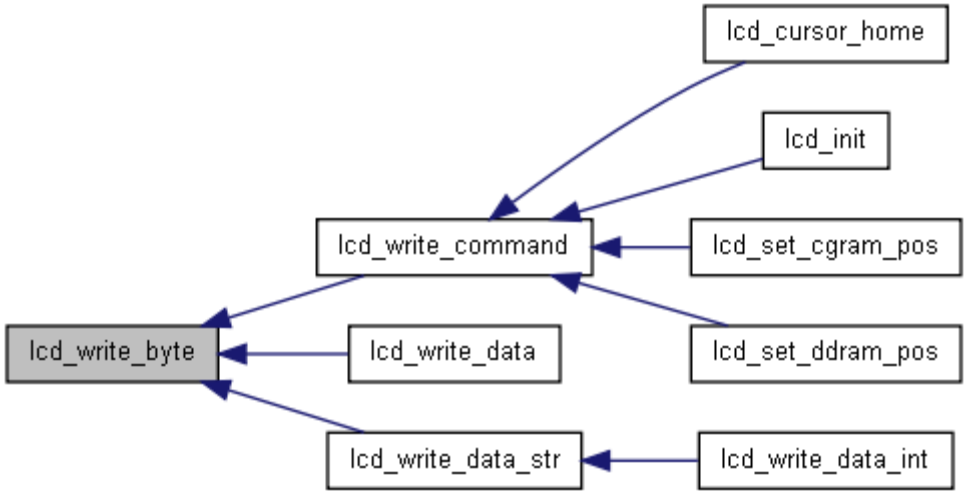
void lcd_write_byte (uns8 *data*)

```
58                                  {
59
60      lcd_write_nibble(data >> 4);    // Write upper nibble
61      lcd_toggle_e();
62      lcd_write_nibble(data); // Write lower nibble
63      lcd_toggle_e();
64  }
```

Here is the call graph for this function:



Here is the caller graph for this function:

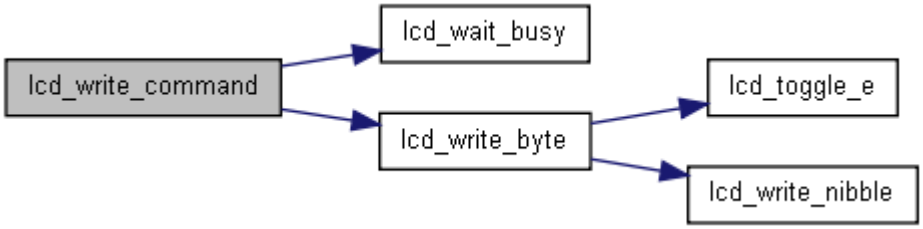


void lcd_write_command (uns8 *data*)

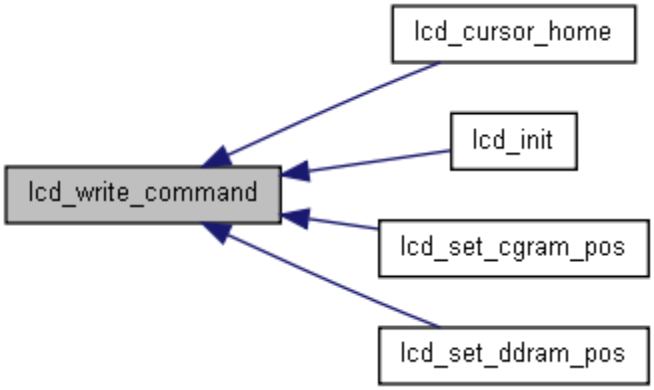
Use this to send commands to the LCD, eg, changing cursor position

```
113                                 {
114
115     lcd_wait_busy();
116
117     clear_pin(lcd_rs_port, lcd_rs_pin);
118     clear_pin(lcd_rw_port, lcd_rw_pin);
119
120     lcd_write_byte(data);
121 }
```

Here is the call graph for this function:



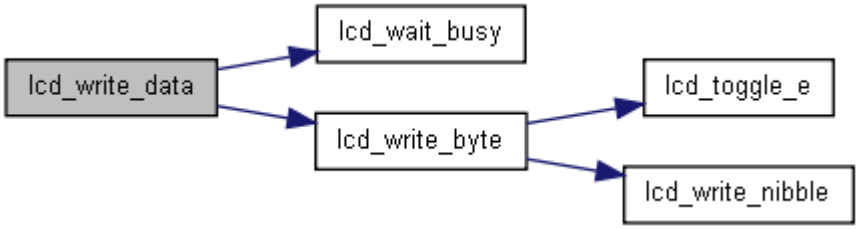
Here is the caller graph for this function:



void lcd_write_data (uns8 *data*)

```
123                                 {
124     lcd_wait_busy();
125
126     set_pin(lcd_rs_port, lcd_rs_pin);
127     clear_pin(lcd_rw_port, lcd_rw_pin);
128
129     lcd_write_byte(data);
130 }
```

Here is the call graph for this function:

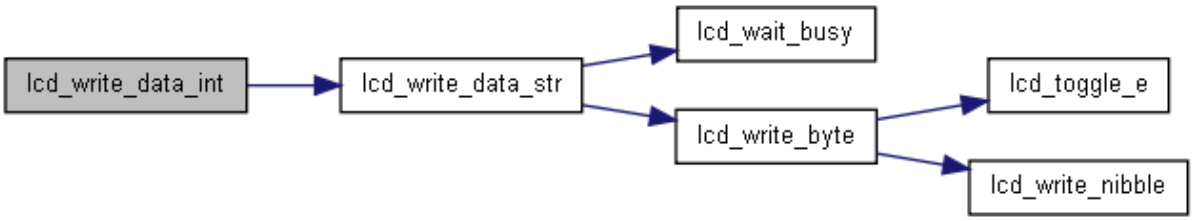


void lcd_write_data_int (uns16 *i*)

Displays an unsigned 16 bit integer on the LCD

```
145                                 {
146
147 char buffer[6];
148
149     itoa( i, buffer, 10 );
150     lcd_write_data_str(buffer);
151 }
```

Here is the call graph for this function:

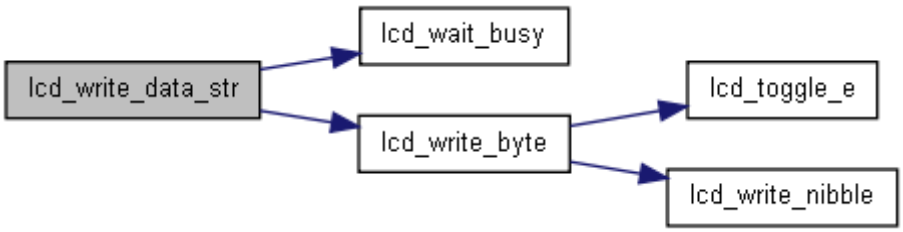


void lcd_write_data_str (char * *str*)

Display the string on the LCD from the current cursor position

```
132                                   {
133
134     lcd_wait_busy();
135
136     set_pin(lcd_rs_port, lcd_rs_pin);
137     clear_pin(lcd_rw_port, lcd_rw_pin);
138
139
140     while (*str) {
141         lcd_write_byte(*str++);
142     }
143 }
```

Here is the call graph for this function:

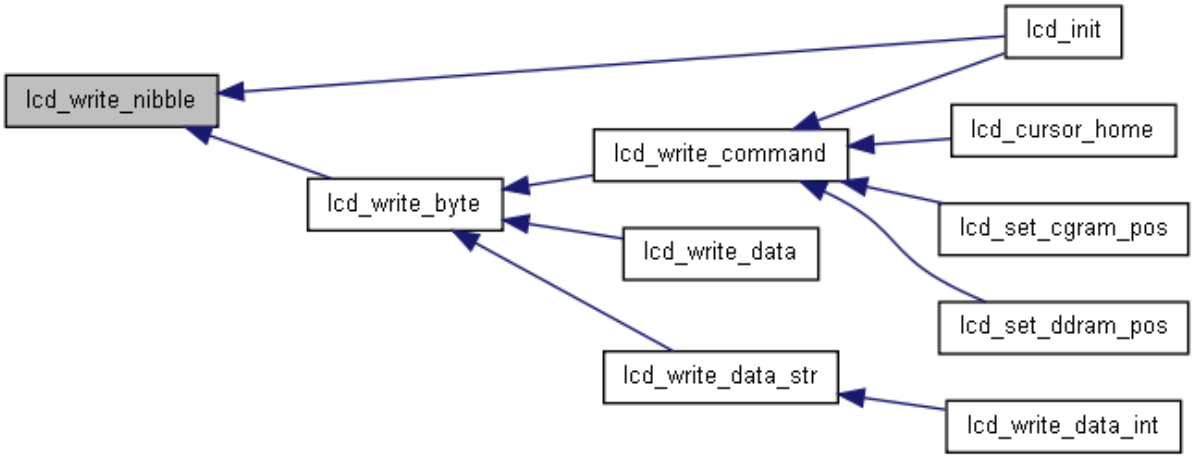


Here is the caller graph for this function:

void lcd_write_nibble (uns8 *data*)

```
47                                  {
48     // There would be a more efficient way to do this if all the pins
49     // were on the same port - but we can't guarantee that. At least
50     // this way all pins are protected from read before write problems.
51
52     change_pin(lcd_db4_port, lcd_db4_pin, data.0);
53     change_pin(lcd_db5_port, lcd_db5_pin, data.1);
54     change_pin(lcd_db6_port, lcd_db6_pin, data.2);
55     change_pin(lcd_db7_port, lcd_db7_pin, data.3);
56 }
```

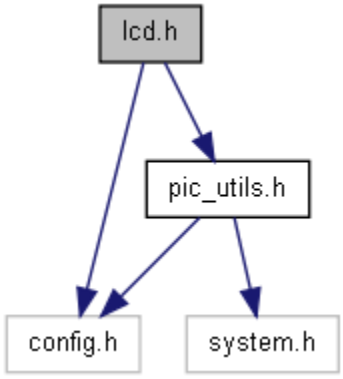
Here is the caller graph for this function:



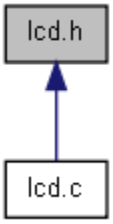
---

lcd.h File Reference

LCD routines.
Include dependency graph for lcd.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define __lcd_h include
- #define LCD_CLEAR_DISP 0b00000001
- #define lcd_clear_display() lcd_write_command(LCD_CLEAR_DISP);
- *Clear LCD display.* #define LCD_LINE1 0b10000000
- #define LCD_LINE2 0b11000000
- #define LCD_LINE3 0b10010100
- #define LCD_LINE4 0b11010100
- #define lcd_return_home() lcd_write_command(LCD_RETURN_HOME);
- *Return cursor home.* #define LCD_RETURN_HOME 0b00000010
- #define LCD_SET_DRAM_ADDR 0b10000000

Functions

- void lcd_init ()
- *Initialise LCD ready for display.* void lcd_setup ()
- *Setup port and pins to talk to LCD.* void lcd_wait_busy ()
- *Wait while LCD is busy.* void lcd_write_command (uns8 data)
- *Sends a command to the LCD.* void lcd_write_data (uns8 data)
- *Send one byte of data to the LCD.* void lcd_write_data_int (uns16 i)
- *Print a 16 bit integer the the LCD.* void lcd_write_data_str (char *str)

Print a string to the LCD.

Detailed Description

This module contains routines to communicate with an LCD via the 4 bit parallel mode. All pins are protected from read-before-write problems with picpack's port latch simulation. Reads ready flag to check LCD isn't busy before sending more data.

Define Documentation

#define __lcd_h include

#define LCD_CLEAR_DISP 0b00000001

Clear LCD display

#define lcd_clear_display() lcd_write_command(LCD_CLEAR_DISP);

#define LCD_LINE1 0b10000000

Move cursor to line 1

#define LCD_LINE2 0b11000000

Move cursor to line 2

#define LCD_LINE3 0b10010100

Move cursor to line 3

#define LCD_LINE4 0b11010100

Move cursor to line 4

#define lcd_return_home() lcd_write_command(LCD_RETURN_HOME);

#define LCD_RETURN_HOME 0b00000010

Move cursor to top left position

#define LCD_SET_DRAM_ADDR 0b10000000

Move to DRAM address (cursor position)

Function Documentation

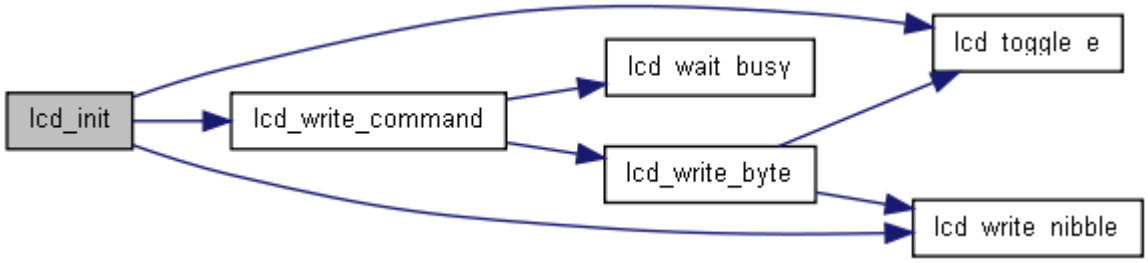
void lcd_init ()

Configures LCD for 4 bit operation and gets ready for displaying text

```
84                  {
85
86      delay_ms(16);
87      lcd_write_nibble(0x03);
88      lcd_toggle_e();
89      delay_ms(5);
90      lcd_write_nibble(0x03);
91      lcd_toggle_e();
92      delay_ms(1);
93      lcd_write_nibble(0x03);
94      lcd_toggle_e();
95      lcd_write_nibble(0x02); //0b 0010 1000
96      lcd_toggle_e();
97
98      // Now we are in 4bit mode
99
100      lcd_write_command(0b00101000); // 0x28 numlines=1 font=0
101
102      lcd_write_command(0b00001000); // disp off, curs off blink off
103
104      lcd_write_command(0b00000110); // cursor move, inc, no shift
105
```

```
106     lcd_write_command(0b00001100); // disp on, curs on blink on
107
108     lcd_write_command(LCD_CLEAR_DISP);
109     lcd_write_command(LCD_RETURN_HOME);
110
111 }
```

Here is the call graph for this function:



void lcd_setup ()

Call this routine first, to set up tris bits correctly to talk to the LCD

```
67                  {
68
69     // Set up tris bits
70     make_output(lcd_e_port, lcd_e_pin);
71     make_output(lcd_rs_port, lcd_rs_pin);
72     make_output(lcd_rw_port, lcd_rw_pin);
73     make_output(lcd_db7_port, lcd_db7_pin);
74     make_output(lcd_db6_port, lcd_db6_pin);
75     make_output(lcd_db5_port, lcd_db5_pin);
76     make_output(lcd_db4_port, lcd_db4_pin);
77
78     clear_pin(lcd_e_port, lcd_e_pin);
79     clear_pin(lcd_rs_port, lcd_rs_pin);
80     clear_pin(lcd_rw_port, lcd_rw_pin);
81 }
```

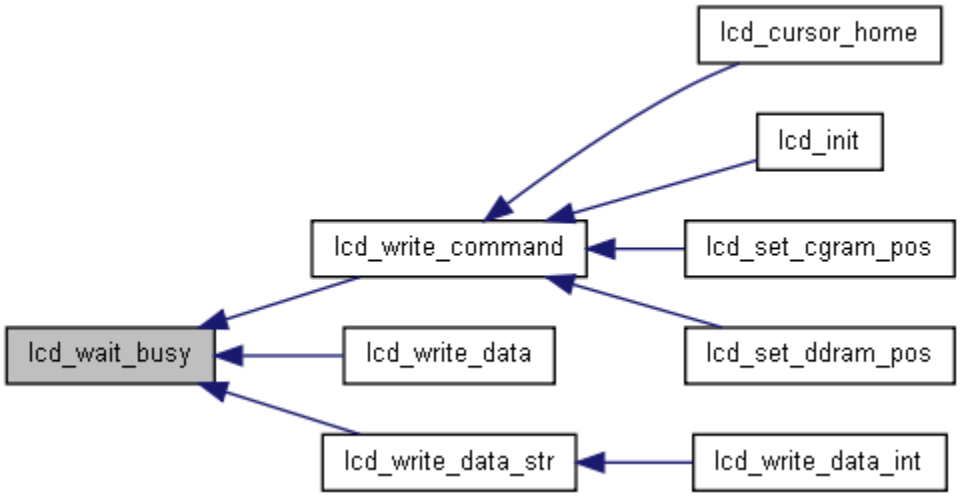
void lcd_wait_busy ()

Internal routine to wait while the LCD is busy and unable to accept more data

```
154                     {
155
156
157     set_bit(tris_array[lcd_db7_port - PORTA], lcd_db7_pin); // db7 input
158     set_bit(tris_array[lcd_db6_port - PORTA], lcd_db6_pin); // db6 input
159     set_bit(tris_array[lcd_db5_port - PORTA], lcd_db5_pin); // db5 input
160     set_bit(tris_array[lcd_db4_port - PORTA], lcd_db4_pin); // db4 input
161
162
163     clear_pin(lcd_rs_port, lcd_rs_pin);
164     set_pin(lcd_rw_port, lcd_rw_pin);
165
166     char counter = 0;
167
168     set_pin(lcd_e_port, lcd_e_pin);
169     // Wait for completion of the operation, with a timeout of ~.5 seconds
170     // LCD d7 is high if the operation is complete.
171     while ((test_pin(lcd_db7_port, lcd_db7_pin) == 1) && counter < 0xF0){
172         clear_pin(lcd_e_port, lcd_e_pin);
173         set_pin(lcd_e_port, lcd_e_pin);
174         delay_us(100);
175         clear_pin(lcd_e_port, lcd_e_pin);
176         //delay_us(100);
177         counter++;
```

```
178     }
179
180
181     // Check if the previous loop timed out
182     if (counter == 0xF0) {
183     }
184
185     clear_bit(tris_array[lcd_db7_port - PORTA], lcd_db7_pin); // db7 output
186     clear_bit(tris_array[lcd_db6_port - PORTA], lcd_db6_pin); // db6 output
187     clear_bit(tris_array[lcd_db5_port - PORTA], lcd_db5_pin); // db5 output
188     clear_bit(tris_array[lcd_db4_port - PORTA], lcd_db4_pin); // db4 output
189
190     return;
191 }
```

Here is the caller graph for this function:

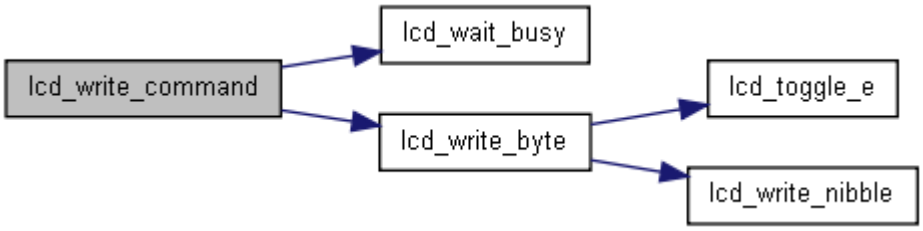


void lcd_write_command (uns8 *data*)

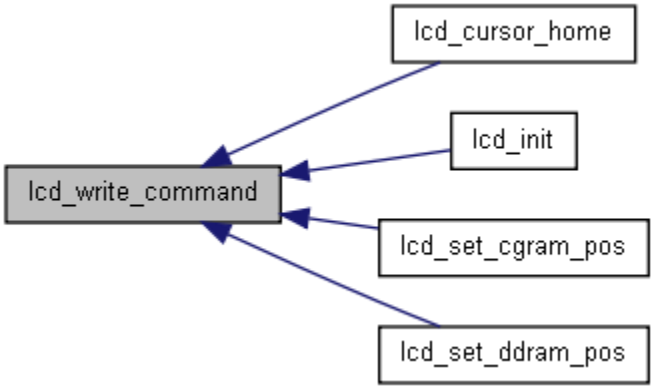
Use this to send commands to the LCD, eg, changing cursor position

```
113                                     {
114
115     lcd_wait_busy();
116
117     clear_pin(lcd_rs_port, lcd_rs_pin);
118     clear_pin(lcd_rw_port, lcd_rw_pin);
119
120     lcd_write_byte(data);
121 }
```

Here is the call graph for this function:



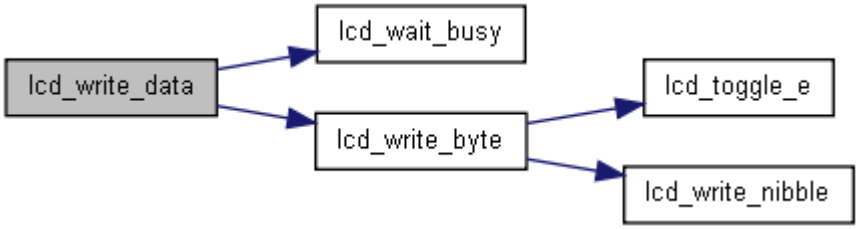
Here is the caller graph for this function:



void lcd_write_data (uns8 *data*)

```
123                                 {
124     lcd_wait_busy();
125
126     set_pin(lcd_rs_port, lcd_rs_pin);
127     clear_pin(lcd_rw_port, lcd_rw_pin);
128
129     lcd_write_byte(data);
130 }
```

Here is the call graph for this function:

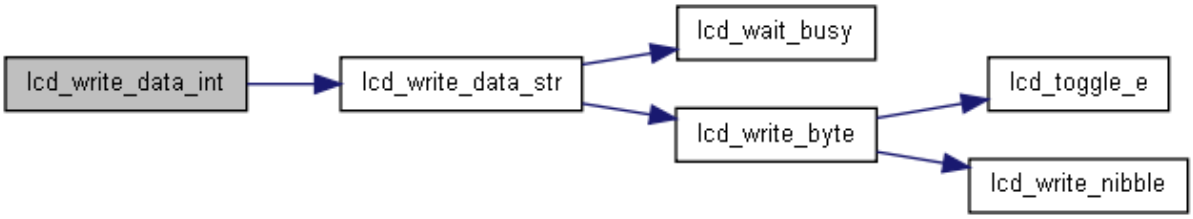


void lcd_write_data_int (uns16 *i*)

Displays an unsigned 16 bit integer on the LCD

```
145                                  {
146
147 char buffer[6];
148
149     itoa( i, buffer, 10 );
150     lcd_write_data_str(buffer);
151 }
```

Here is the call graph for this function:

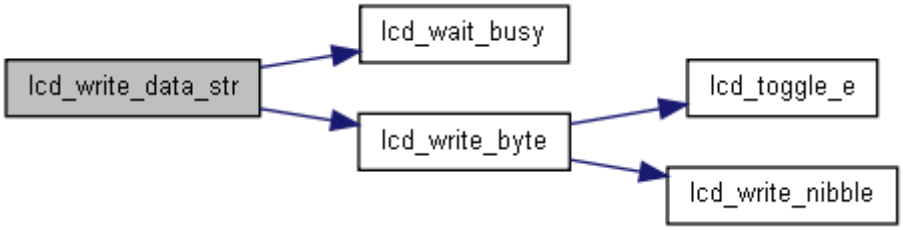


void lcd_write_data_str (char * *str*)

Display the string on the LCD from the current cursor position

```
132                                     {
133
134     lcd_wait_busy();
135
136     set_pin(lcd_rs_port, lcd_rs_pin);
137     clear_pin(lcd_rw_port, lcd_rw_pin);
138
139
140     while (*str) {
141         lcd_write_byte(*str++);
142     }
143 }
```

Here is the call graph for this function:

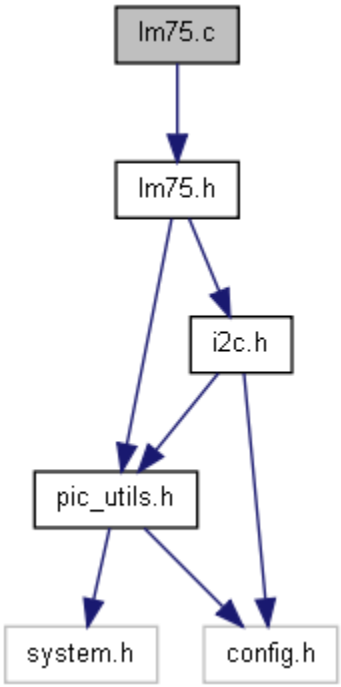


Here is the caller graph for this function:

---

lm75.c File Reference

Include dependency graph for lm75.c:



Functions

- uns8 lm75_get_config (uns8 addr)
- *Get LM75 config register.* uns16 lm75_get_temp (uns8 addr)
- *Request temperature from LM75.* void lm75_set_config (uns8 addr, uns8 config)
- *Set LM75 config register.* void lm75_setup ()

Setup lm75 ports and pins.

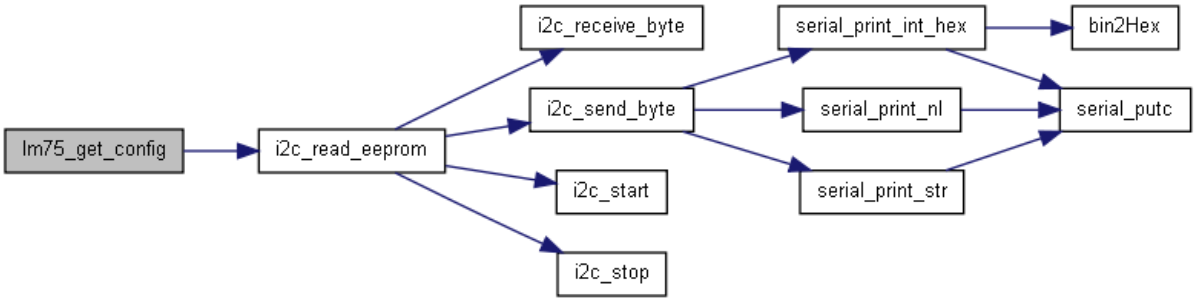
Function Documentation

uns8 lm75_get_config (uns8 *addr*)

Gets the LM75 config register (memory location 0x01)

```
49 {
50     return i2c_read_eeprom(0x90 + addr, 0x01);
51 }
```

Here is the call graph for this function:

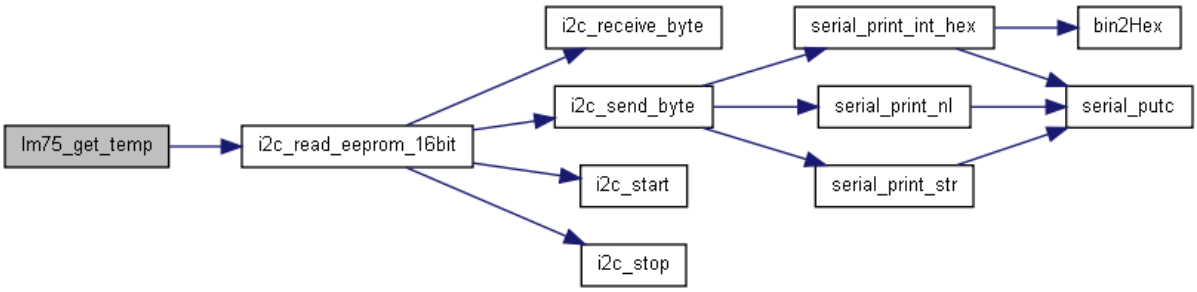


uns16 lm75_get_temp (uns8 *addr*)

Returns 16bit raw temperature register from LM75

```
55 {
56
57     return i2c_read_eeprom_16bit(0x90 + addr, 0x00);
58
59 }
```

Here is the call graph for this function:

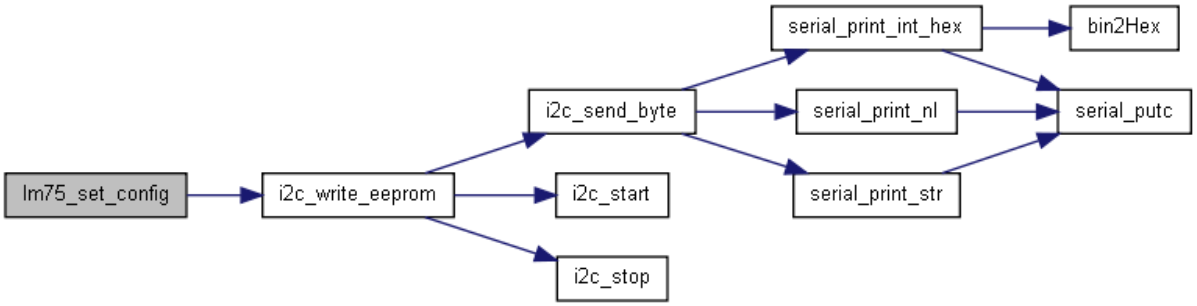


void lm75_set_config (uns8 *addr*, uns8 *config*)

Sets the LM75 config register (memory location 0x01)

```
44 {
45     i2c_write_eeprom(0x90 + addr, 0x01, config);
46 }
```

Here is the call graph for this function:



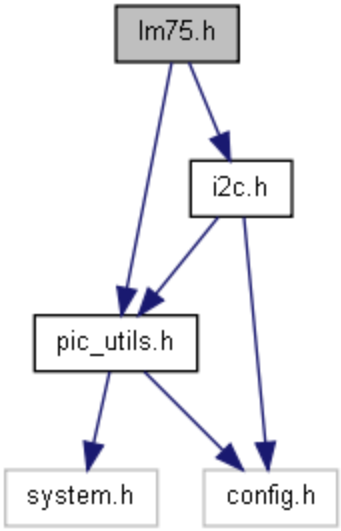
void lm75_setup (void)

```
39                     {
40     i2c_setup();
41 }
```

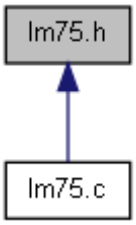
---

lm75.h File Reference

LM75 temperature sensor routines.
Include dependency graph for lm75.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define LM75_NORMAL 0
- #define LM75_SHUTDOWN 1

Functions

- uns8 lm75_get_config (uns8 addr)
- *Get LM75 config register.* uns16 lm75_get_temp (uns8 addr)
- *Request temperature from LM75.* void lm75_set_config (uns8 addr, uns8 config)
- *Set LM75 config register.* void lm75_setup (void)

Setup lm75 ports and pins.

Detailed Description

A library to communicate with the LM75 sensor

Define Documentation

#define LM75_NORMAL 0

config define for normal mode

#define LM75_SHUTDOWN 1

Config define for low power mode

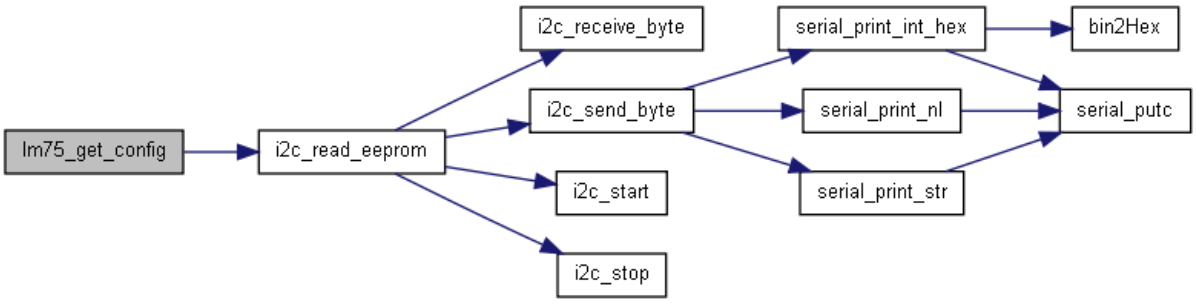
Function Documentation

uns8 lm75_get_config (uns8 *addr*)

Gets the LM75 config register (memory location 0x01)

```
49 {
50     return i2c_read_eeprom(0x90 + addr, 0x01);
51 }
```

Here is the call graph for this function:

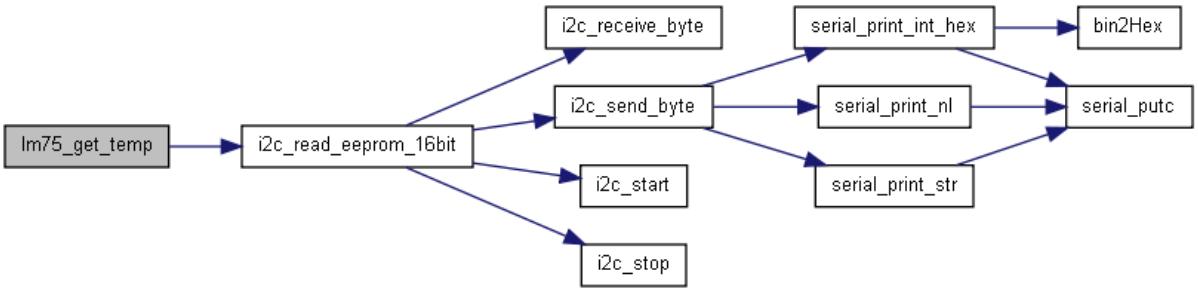


uns16 lm75_get_temp (uns8 *addr*)

Returns 16bit raw temperature register from LM75

```
55 {
56
57     return i2c_read_eeprom_16bit(0x90 + addr, 0x00);
58
59 }
```

Here is the call graph for this function:

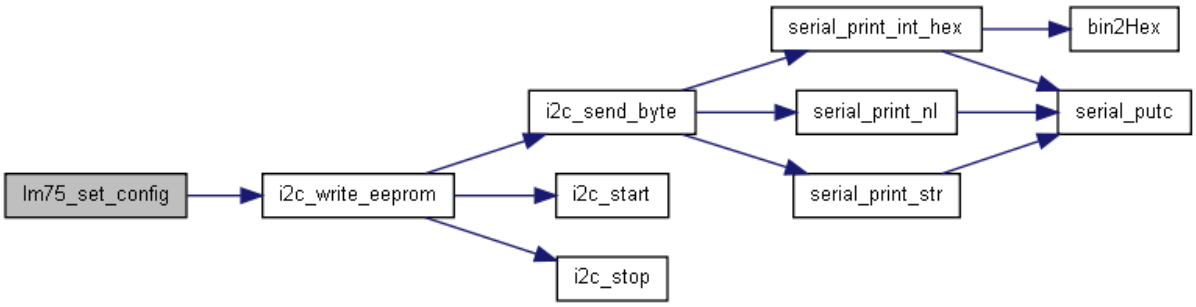


void lm75_set_config (uns8 *addr*, uns8 *config*)

Sets the LM75 config register (memory location 0x01)

```
44 {
45     i2c_write_eeprom(0x90 + addr, 0x01, config);
46 }
```

Here is the call graph for this function:



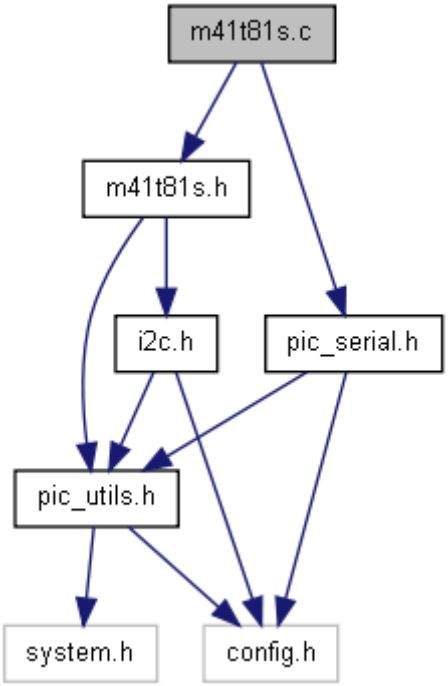
void lm75_setup (void)

```
39                      {
40      i2c_setup();
41  }
```

---

m41t81s.c File Reference

Include dependency graph for m41t81s.c:



Functions

- uns8 bcd_to_dec (uns8 bcd)
- uns8 dec_to_bcd (uns8 dec)
- uns8 rtc_get_date ()

- *Get the date register from the ds1307.* uns8 rtc_get_dow ()
- *Get the day register from the m41t81s.* uns8 rtc_get_hours ()
- *Get the decoded hours register from the ds1307.* uns8 rtc_get_minutes ()
- *Get the decoded minutes register from the ds1307.* uns8 rtc_get_month ()
- *Get the month register from the ds1307.* uns8 rtc_get_register (uns8 reg)
- uns8 rtc_get_seconds ()
- *Get the decoded seconds register from the ds1307.* uns8 rtc_get_year ()
- *Get the year register from the ds1307.* void rtc_set_date (uns8 date)
- *Set the date register from the ds1307.* void rtc_set_day (uns8 day)
- *Set the day of the week register from the ds1307.* void rtc_set_hours (uns8 hours)
- *Set the hours register in the ds1307.* void rtc_set_minutes (uns8 minutes)
- *Set the minutes register from the m41t81s.* void rtc_set_month (uns8 month)
- *Set the month register in the ds1307.* void rtc_set_register (uns8 reg, uns8 data)
- void rtc_set_seconds (uns8 seconds)
- *Set the seconds register in the ds1307.* void rtc_set_sqw_freq (uns8 freq)
- *Set the frequence of the square wave output pin.* void rtc_set_year (uns8 year)
- *Set the year register from the m41t81s.* void rtc_setup_io ()
- *Setup ports and pins for use in the ds1307.* void rtc_start_clock ()
- *Starts the clock in the ds1307.* void rtc_start_sqw_output ()
- *Start pulsing on square wave output pin.* void rtc_stop_clock ()
- *Stop the clock in the ds1307.* void rtc_stop_sqw_output ()

Stop pulsing on square wave output pin.

Function Documentation

uns8 bcd_to_dec (uns8 *bcd*)

```
40                              {
41     return (bcd & 0b00001111) + ((bcd >> 4) * 10);
42 }
```

uns8 dec_to_bcd (uns8 *dec*)

```
44                              {
45     return ((dec / 10) << 4) + (dec % 10);
46 }
```

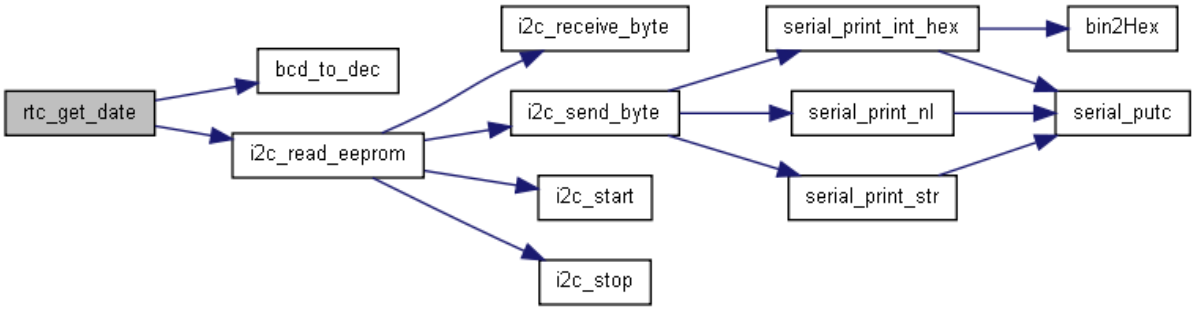
uns8 rtc_get_date ()

Get the date register from the m41t81s.

Returns the date in month from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 1 through 28/29/30/31 depending on month

```
66                      {
67     return bcd_to_dec(i2c_read_eeprom(m41t81s_device_addr, m41t81s_date_reg));
68 }
```

Here is the call graph for this function:

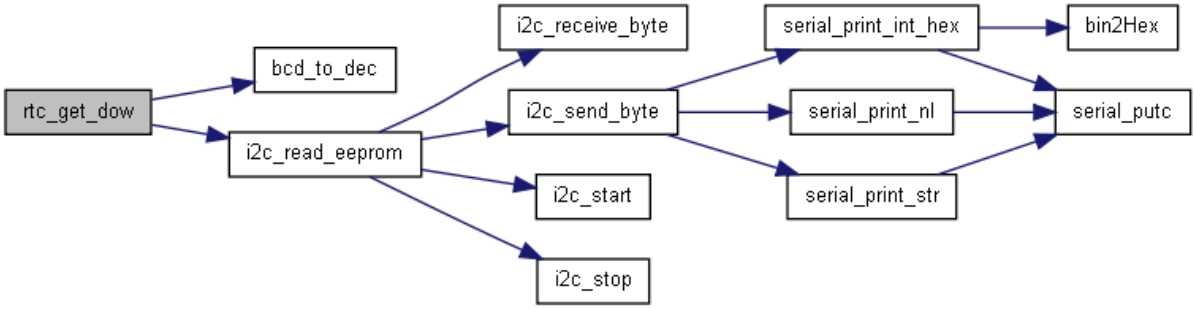


uns8 rtc_get_dow ()

Returns the day of the week from the m41t81s. The result is coverted to decimal from BCD and is ready to use. Range - 1 through 7

```
62                      {
63     return bcd_to_dec(i2c_read_eeprom(m41t81s_device_addr, m41t81s_dow_reg));
64 }
```

Here is the call graph for this function:



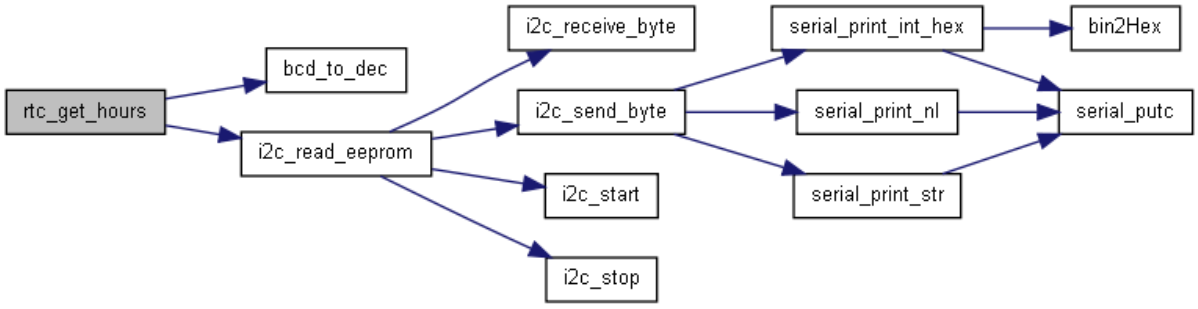
uns8 rtc_get_hours ()

Get the decoded hours register from the m41t81s.

Returns hour from the ds1307. The result is coverted to decimal from BCD and is ready to use. These routines assume the ds1307 is running in 24 hour mode. Range - 0 through 23

```
51                      {
52
53     // m41t81s is always in 24 hour mode
54
55     return bcd_to_dec(0b00111111 & i2c_read_eeprom(m41t81s_device_addr, m41t81s_hours_reg));
56 }
```

Here is the call graph for this function:



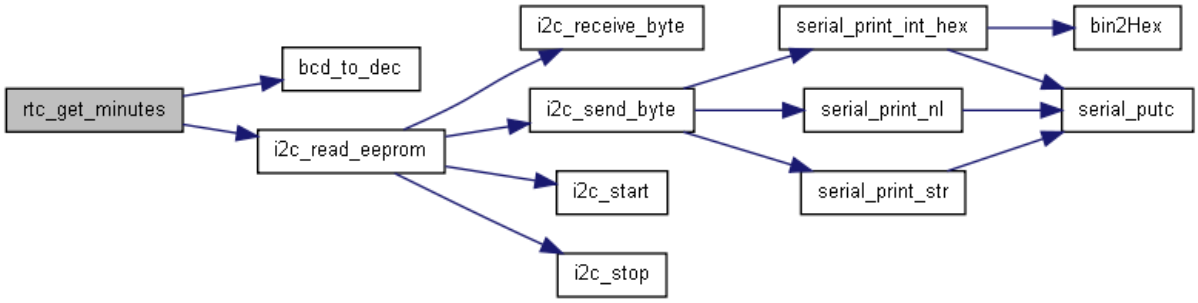
uns8 rtc_get_minutes ()

Get the decoded minutes register from the m41t81s.

Returns the number of minutes past the hour from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

```
48                          {
49     return bcd_to_dec(i2c_read_eeprom(m41t81s_device_addr, m41t81s_minutes_reg));
50 }
```

Here is the call graph for this function:



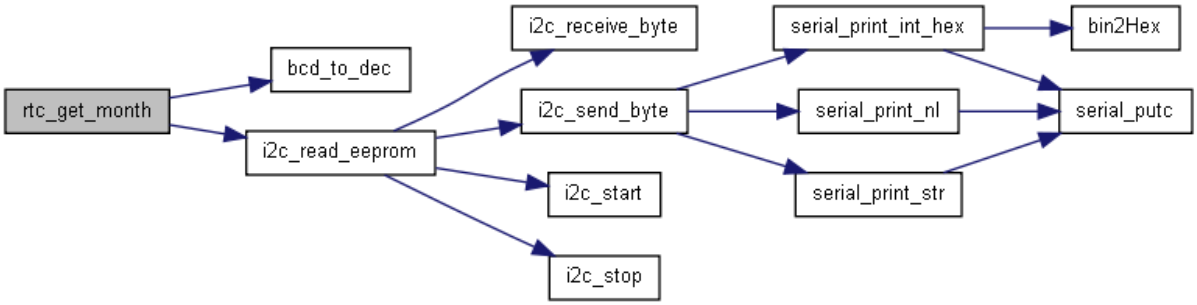
uns8 rtc_get_month ()

Get the month register from the m41t81s.

Returns the month of the year from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 1 through 12

```
70                          {
71     return bcd_to_dec(i2c_read_eeprom(m41t81s_device_addr, m41t81s_month_reg));
72 }
```

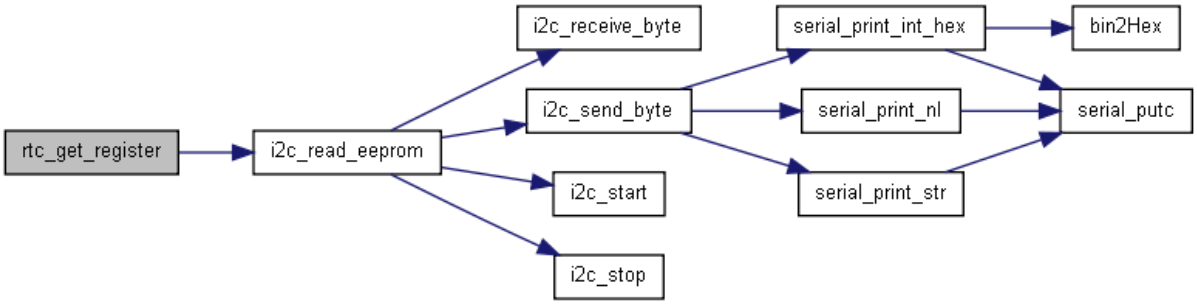
Here is the call graph for this function:



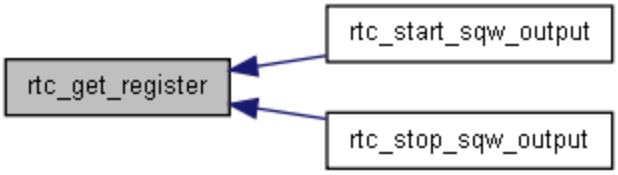
uns8 rtc_get_register (uns8 *reg*)

```
78                                      {
79     return i2c_read_eeprom(m41t81s_device_addr, reg);
80 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



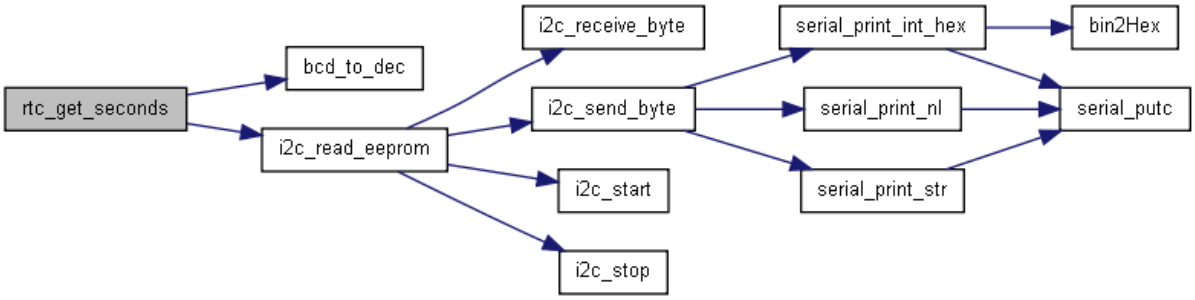
uns8 rtc_get_seconds ()

Get the decoded seconds register from the m41t81s.

Returns seconds from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

```
58                          {
59     return bcd_to_dec(0b01111111 & i2c_read_eeprom(m41t81s_device_addr,
m41t81s_seconds_reg));
60 }
```

Here is the call graph for this function:



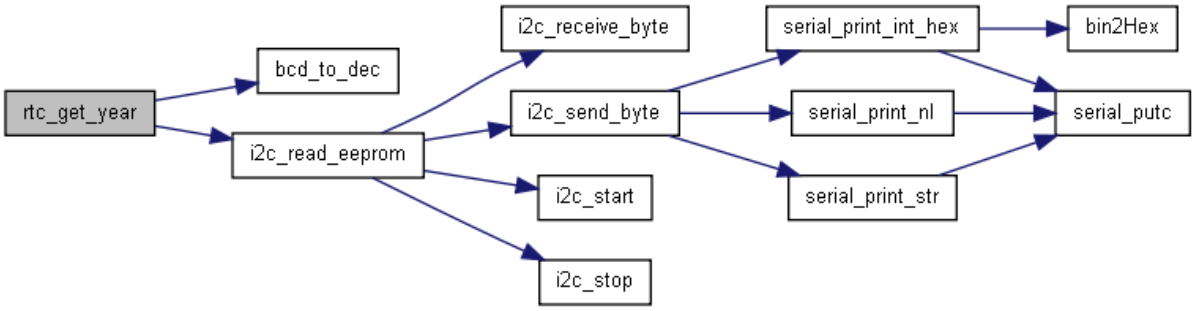
uns8 rtc_get_year ()

Get the year register from the m41t81s.

Returns the year from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 0 through 99

```
74                          {
75     return bcd_to_dec(i2c_read_eeprom(m41t81s_device_addr, m41t81s_year_reg));
76 }
```

Here is the call graph for this function:



void rtc_set_date (uns8 *date*)

Set the date register from the m41t81s.

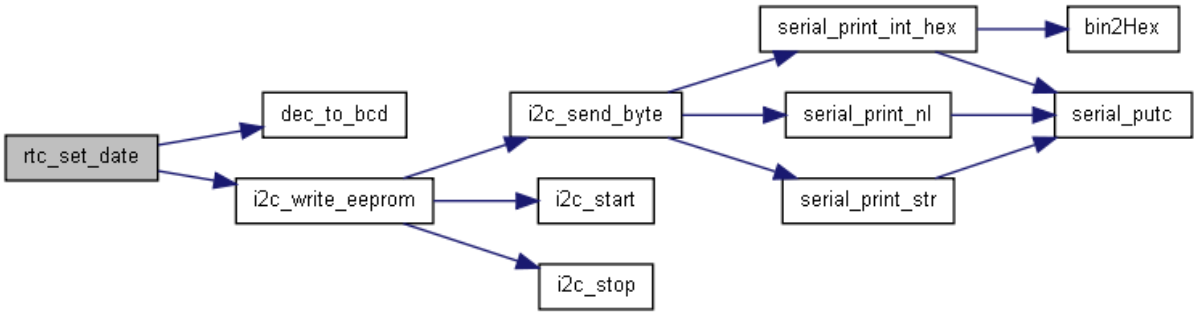
Changes the date in the ds1307.

Parameters:

seconds Value to set date to

```
107                                {
108     i2c_write_eeprom(m41t81s_device_addr, m41t81s_date_reg, dec_to_bcd(date));
109 }
```

Here is the call graph for this function:



void rtc_set_day (uns8 *day*)

Set the day of the week register from the m41t81s.

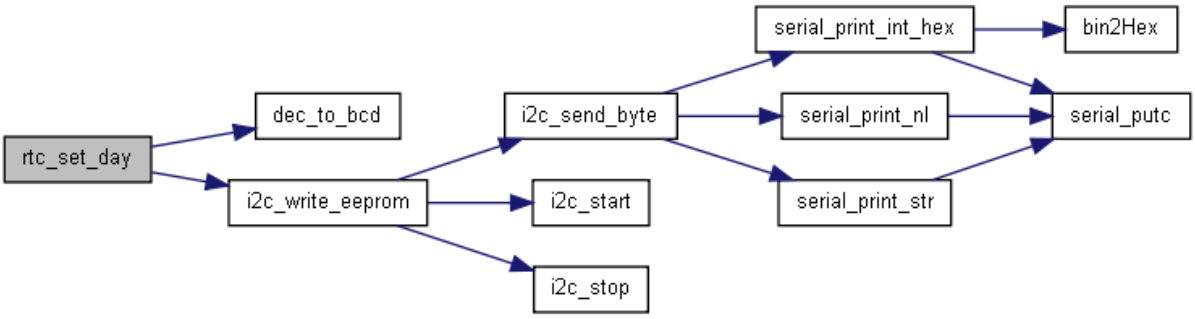
Changes the day of the week in the ds1307.

Parameters:

seconds Value to set day to

```
104                               {
105     i2c_write_eeprom(m41t81s_device_addr, m41t81s_dow_reg, dec_to_bcd(day));
106 }
```

Here is the call graph for this function:



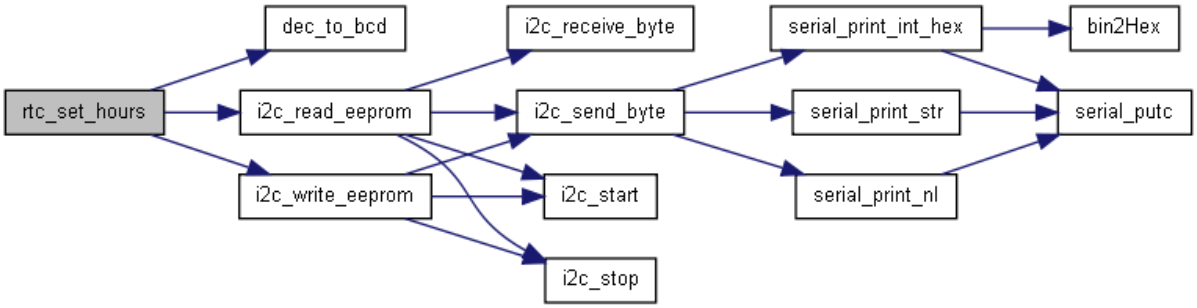
void rtc_set_hours (uns8 *hours*)

Set the hours register in the m41t81s.

Changes the hours in the ds1307. Forces the ds1307 into 24 hour mode.

```
116                                 {
117     // preserve the century / century enable bits
118     i2c_write_eeprom(m41t81s_device_addr, m41t81s_hours_reg, (0b11000000 &
i2c_read_eeprom(m41t81s_device_addr, m41t81s_hours_reg)) | dec_to_bcd(hours));
119 }
```

Here is the call graph for this function:



void rtc_set_minutes (uns8 *minutes*)

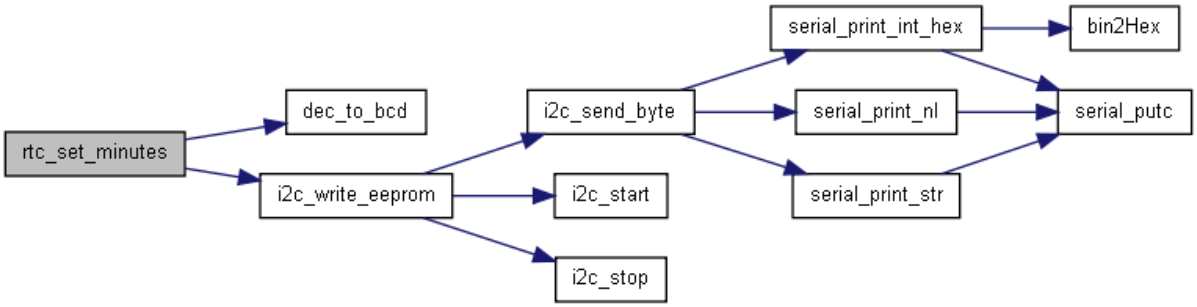
Changes the minutes in the m41t81s.

Parameters:

seconds Value to set minutes to

```
101                                     {
102     i2c_write_eeprom(m41t81s_device_addr, m41t81s_minutes_reg, dec_to_bcd(minutes));
103 }
```

Here is the call graph for this function:



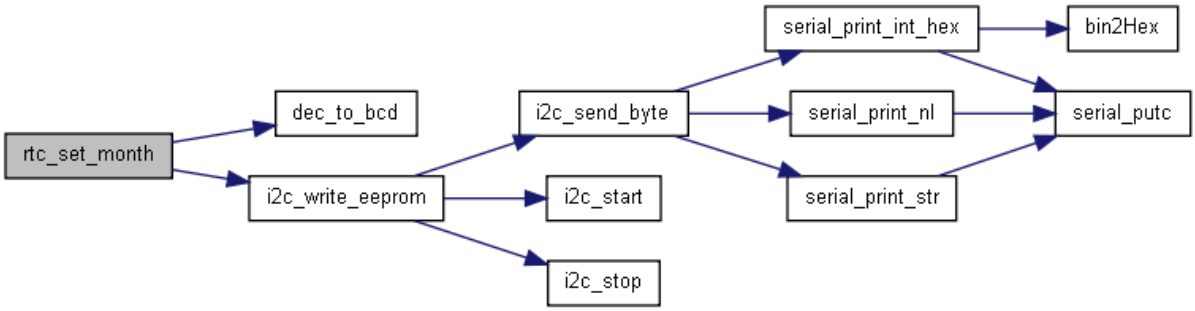
void rtc_set_month (uns8 *month*)

Set the month register in the m41t81s.

Changes the month in the ds1307.

```
121                                 {
122     i2c_write_eeprom(m41t81s_device_addr, m41t81s_month_reg , dec_to_bcd(month));
123 }
```

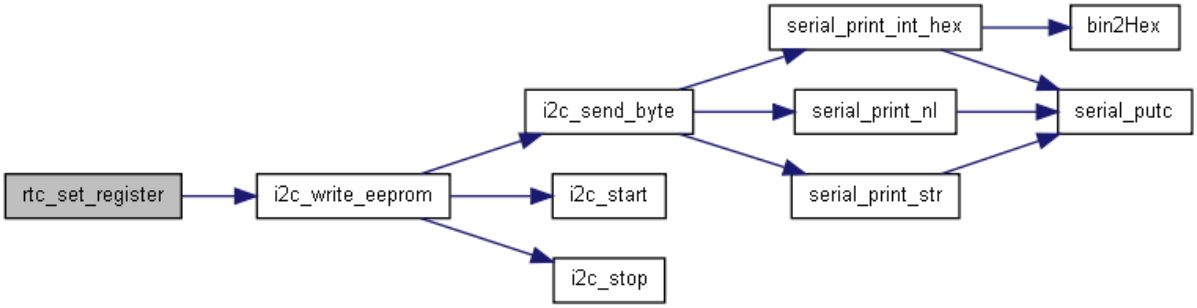
Here is the call graph for this function:



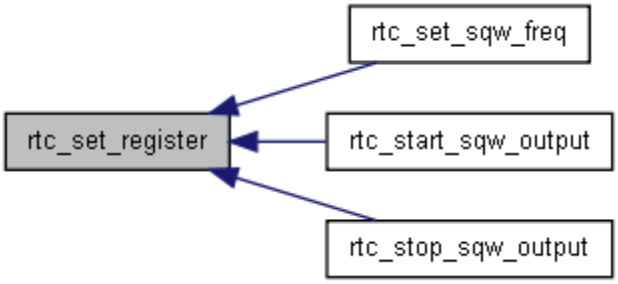
void rtc_set_register (uns8 *reg*, uns8 *data*)

```
82                                          {
83     i2c_write_eeprom(m41t81s_device_addr, reg, data);
84 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void rtc_set_seconds (uns8 *seconds*)

Set the seconds register in the m41t81s.

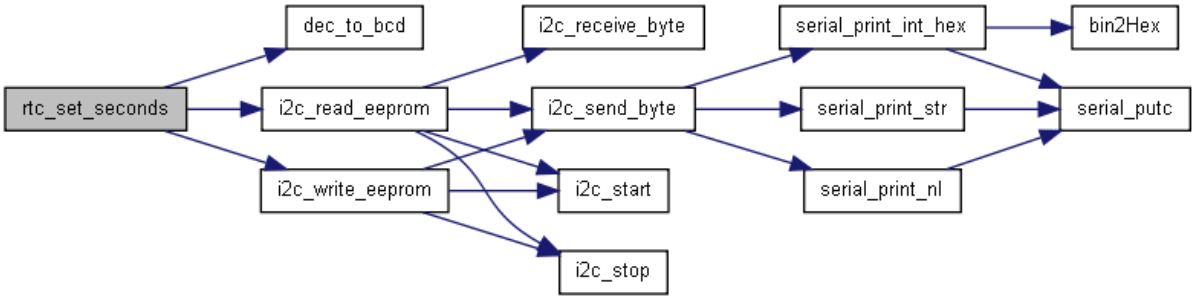
Changes the seconds in the ds1307.

Parameters:

seconds Value to set seconds to

```
111                                     {
112     // preserve the stop bit
113     i2c_write_eeprom(m41t81s_device_addr, m41t81s_seconds_reg, (0b10000000 &
i2c_read_eeprom(m41t81s_device_addr, m41t81s_seconds_reg)) | dec_to_bcd(seconds));
114 }
```

Here is the call graph for this function:



void rtc_set_sqw_freq (uns8 *freq*)

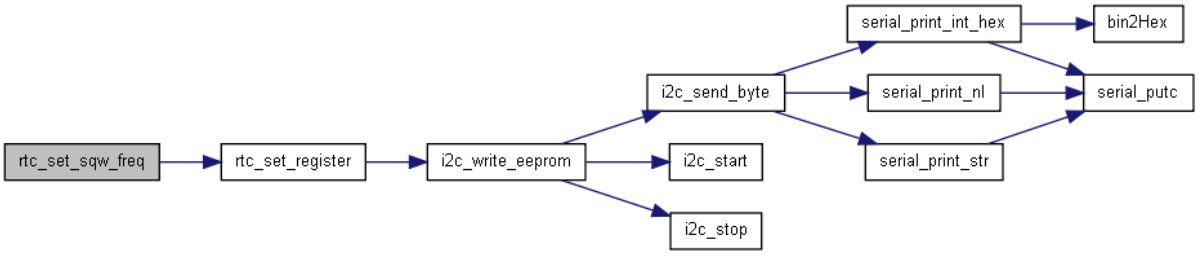
Use one of the following self explanitory defines:

rtc_sqw_freq_32768Hz rtc_sqw_freq_8192Hz rtc_sqw_freq_4096Hz rtc_sqw_freq_2048Hz rtc_sqw_freq_1024Hz rtc_sqw_freq_512Hz rtc_sqw_freq_256Hz rtc_sqw_freq_128Hz rtc_sqw_freq_64Hz rtc_sqw_freq_32Hz rtc_sqw_freq_16Hz rtc_sqw_freq_8Hz rtc_sqw_freq_4Hz rtc_sqw_freq_2Hz rtc_sqw_freq_1Hz

Note that on the m41t81s 18384Hz is not available.

```
125                                     {
126
127     freq = freq << 4;
128     rtc_set_register(m41t81s_sqw_reg, freq);
129 }
```

Here is the call graph for this function:

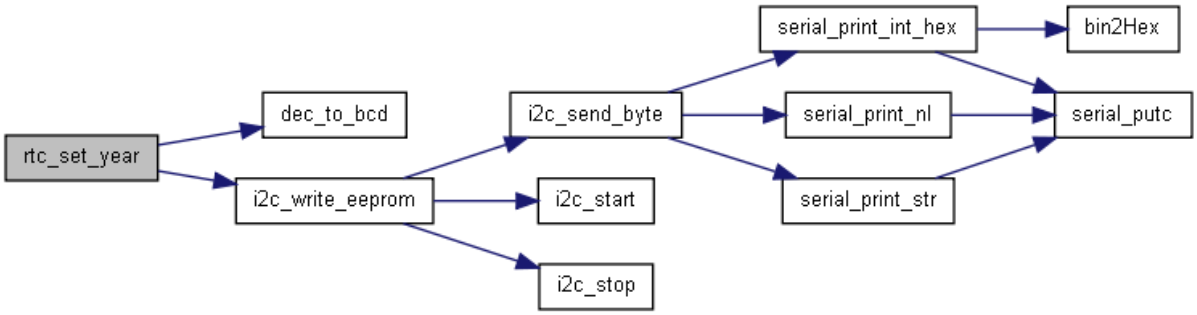


void rtc_set_year (uns8 *year*)

Changes the year in the m41t81s.

```
98                                      {
99     i2c_write_eeprom(m41t81s_device_addr, m41t81s_year_reg, dec_to_bcd(year));
100 }
```

Here is the call graph for this function:



void rtc_setup_io ()

Setup ports and pins for use in the m41t81s.

Calls i2c_setup() to configure ports and pins ready for use

```
143                     {
144     i2c_setup_io();
```

145 }

Here is the call graph for this function:

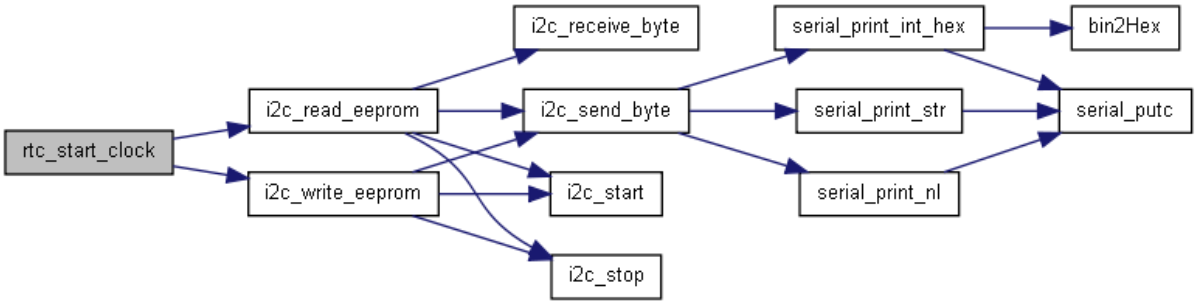
void rtc_start_clock ()

Starts the clock in the m41t81s.

Resume time in the ds1307

```
91                          {
92
93     i2c_write_eeprom(m41t81s_device_addr, m41t81s_alarm_hour_reg, 0b10111111 &
i2c_read_eeprom(m41t81s_device_addr, m41t81s_alarm_hour_reg)); //HT
94     i2c_write_eeprom(m41t81s_device_addr, m41t81s_seconds_reg, 0b01111111 &
i2c_read_eeprom(m41t81s_device_addr, m41t81s_seconds_reg)); //ST
95
96 }
```

Here is the call graph for this function:

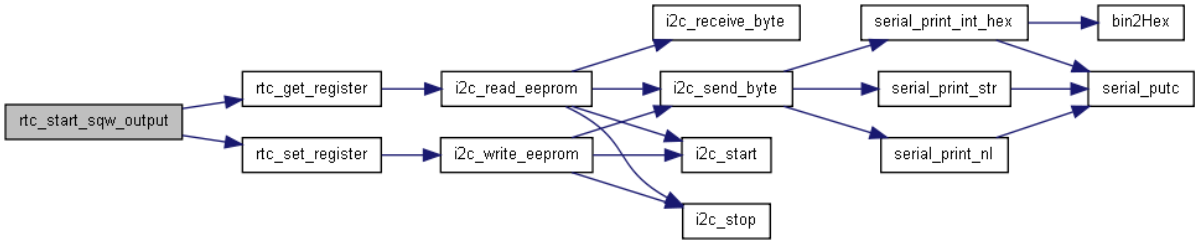


void rtc_start_sqw_output ()

Outputs desired frequency on the SQW output pin. To set the frequency, see rtc_set_sqw_freq(uns8 freq);

```
131                                {
132
133     rtc_set_register(m41t81s_alarm_month_reg, 0b01000000 |
rtc_get_register(m41t81s_alarm_month_reg));
134 }
```

Here is the call graph for this function:



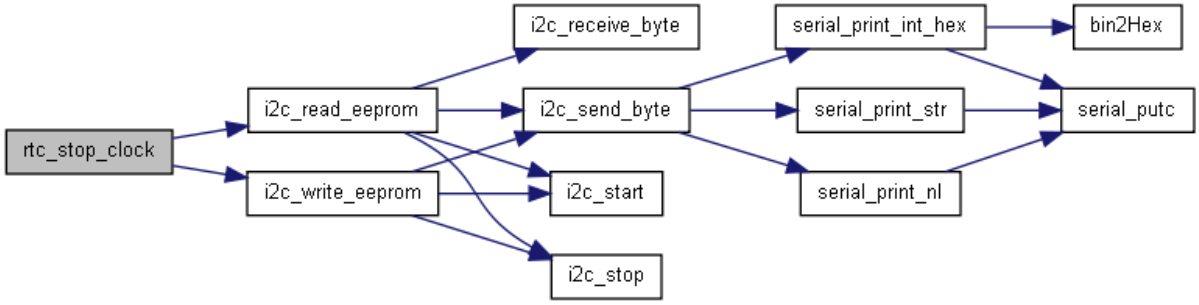
void rtc_stop_clock ()

Stop the clock in the m41t81s.

Pauses time in the ds1307

```
87                         {
88     i2c_write_eeprom(m41t81s_device_addr, m41t81s_seconds_reg, 0b10000000 |
i2c_read_eeprom(m41t81s_device_addr, m41t81s_seconds_reg));
89 }
```

Here is the call graph for this function:

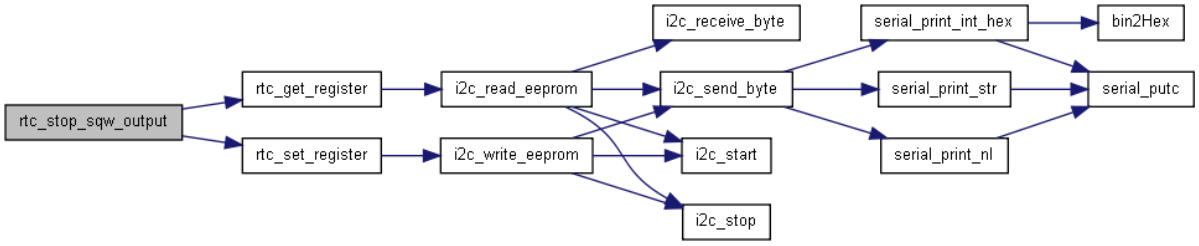


void rtc_stop_sqw_output ()

Stops square wave output

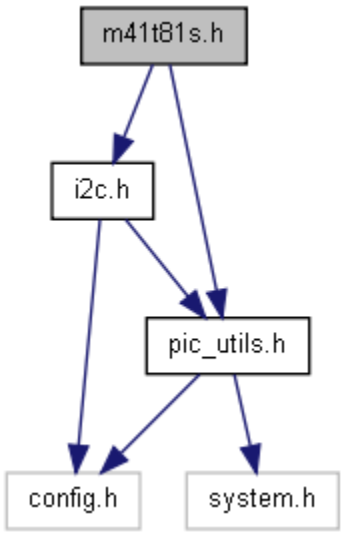
```
136                            {
137
138     rtc_set_register(m41t81s_alarm_month_reg, 0b10111111 |
rtc_get_register(m41t81s_alarm_month_reg));
139
140 }
```

Here is the call graph for this function:

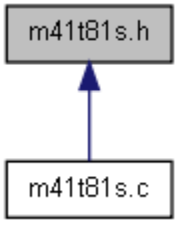


m41t81s.h File Reference

Routines for communicating with the m41t81s real time clock.
Include dependency graph for m41t81s.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define __m41t81s_h defined
- #define m41t81s_alarm_date_reg 0x0B
- #define m41t81s_alarm_hour_reg 0x0C
- #define m41t81s_alarm_min_reg 0x0D
- #define m41t81s_alarm_month_reg 0x0A
- #define m41t81s_alarm_seconds_reg 0x0E
- #define m41t81s_calibration_reg 0x08
- #define m41t81s_date_reg 0x05
- #define m41t81s_device_addr 0xD0
- #define m41t81s_dow_reg 0x04
- #define m41t81s_flags_reg 0x0F
- #define m41t81s_hours_reg 0x03
- #define m41t81s_minutes_reg 0x02

- #define m41t81s_month_reg 0x06
- #define m41t81s_part_seconds_reg 0x00
- #define m41t81s_reserved1_reg 0x10
- #define m41t81s_reserved2_reg 0x11
- #define m41t81s_reserved3_reg 0x12
- #define m41t81s_seconds_reg 0x01
- #define m41t81s_sqw_reg 0x13
- #define m41t81s_watchdog_reg 0x09
- #define m41t81s_year_reg 0x07
- #define rtc_setup() rtc_setup_io()
- #define rtc_sqw_freq_1024Hz 0b00000101
- #define rtc_sqw_freq_128Hz 0b00001000
- #define rtc_sqw_freq_16Hz 0b00001011
- #define rtc_sqw_freq_1Hz 0b00001111
- #define rtc_sqw_freq_2048Hz 0b00000100
- #define rtc_sqw_freq_256Hz 0b00000111
- #define rtc_sqw_freq_2Hz 0b00001110
- #define rtc_sqw_freq_32768Hz 0b00000001
- #define rtc_sqw_freq_32Hz 0b00001010
- #define rtc_sqw_freq_4096Hz 0b00000011
- #define rtc_sqw_freq_4Hz 0b00001101
- #define rtc_sqw_freq_512Hz 0b00000110
- #define rtc_sqw_freq_64Hz 0b00001001
- #define rtc_sqw_freq_8192Hz 0b00000010
- #define rtc_sqw_freq_8Hz 0b00001100

Functions

- uns8 rtc_get_date ()
- *Get the date register from the m41t81s.* uns8 rtc_get_dow ()
- *Get the day register from the m41t81s.* uns8 rtc_get_hours ()
- *Get the decoded hours register from the m41t81s.* uns8 rtc_get_minutes ()
- *Get the decoded minutes register from the m41t81s.* uns8 rtc_get_month ()
- *Get the month register from the m41t81s.* uns8 rtc_get_register (uns8 reg)
- uns8 rtc_get_seconds ()
- *Get the decoded seconds register from the m41t81s.* uns8 rtc_get_year ()
- *Get the year register from the m41t81s.* uns8 rtc_set_config (uns8 config)
- *Set the config register in the m41t81s.* void rtc_set_date (uns8 date)
- *Set the date register from the m41t81s.* void rtc_set_day (uns8 day)
- *Set the day of the week register from the m41t81s.* void rtc_set_hours (uns8 hours)
- *Set the hours register in the m41t81s.* void rtc_set_minutes (uns8 minutes)
- *Set the minutes register from the m41t81s.* void rtc_set_month (uns8 month)
- *Set the month register in the m41t81s.* void rtc_set_register (uns8 reg, uns8 data)
- void rtc_set_seconds (uns8 seconds)
- *Set the seconds register in the m41t81s.* void rtc_set_sqw_freq (uns8 freq)
- *Set the frequence of the square wave output pin.* void rtc_set_year (uns8 year)
- *Set the year register from the m41t81s.* void rtc_setup_io ()
- *Setup ports and pins for use in the m41t81s.* void rtc_start_clock ()
- *Starts the clock in the m41t81s.* void rtc_start_sqw_output ()
- *Start pulsing on square wave output pin.* void rtc_stop_clock ()
- *Stop the clock in the m41t81s.* void rtc_stop_sqw_output ()

Stop pulsing on square wave output pin.

Detailed Description

Define Documentation

#define __m41t81s_h defined

#define m41t81s_alarm_date_reg 0x0B

m41t81s alarm date register (D7=RPT4, D6=RPT5, D5-D4=ABE, D4=AL 10M, D3-D0=Alarm month)

#define m41t81s_alarm_hour_reg 0x0C

m41t81s alarm hour register (D7=RPT3, D6=HT, D5-D4=Alarm 10 Hour, D3-D0=Alarm Hour)

#define m41t81s_alarm_min_reg 0x0D

m41t81s alarm min register (D7=RPT2, D6-D4=Alarm 10 Minutes, D3-D0=Alarm Minutes)

#define m41t81s_alarm_month_reg 0x0A

m41t81s alarm month register (D7=AFE, D6=SQWE, D5=ABE, D4=AL 10M, D3-D0=Alarm month)

#define m41t81s_alarm_seconds_reg 0x0E

m41t81s alarm seconds register (D7=RPT1, D6-D4=Alarm 10 Seconds, D3-D0=Alarm Seconds)

#define m41t81s_calibration_reg 0x08

m41t81s calibration register (D7=OUT, D6=FT, D5=S D4-D0=Calibration)

#define m41t81s_date_reg 0x05

m41t81s date in month register

#define m41t81s_device_addr 0xD0

The m41t81s device address

#define m41t81s_dow_reg 0x04

m41t81s day of week register

#define m41t81s_flags_reg 0x0F

m41t81s flags register (D7=WDF, D6=AF, D5=0, D4=BL, D3=0, D2=OF, D1=0, D0=0 -D4=Alarm 10 Seconds, D3-D0=Alarm Seconds)

#define m41t81s_hours_reg 0x03

m41t81s hours register (D7=CEB, D6=CB)

#define m41t81s_minutes_reg 0x02

m41t81s minutes register

#define m41t81s_month_reg 0x06

m41t81s month register

#define m41t81s_part_seconds_reg 0x00

m41t81s tenths and hundreths of seconds register

#define m41t81s_reserved1_reg 0x10

m41t81s reserved register

#define m41t81s_reserved2_reg 0x11

m41t81s reserved register

#define m41t81s_reserved3_reg 0x12

m41t81s reserved register

#define m41t81s_seconds_reg 0x01

m41t81s seconds register (D7=ST)

#define m41t81s_sqw_reg 0x13

m41t81s SQW register (D7-D4=RS3-RS0)

#define m41t81s_watchdog_reg 0x09

m41t81s watchdog register (D7=OFIE, D6-D2=BMB, D1-D0=RB

#define m41t81s_year_reg 0x07

m41t81s year register

#define rtc_setup() rtc_setup_io()

#define rtc_sqw_freq_1024Hz 0b00000101

#define rtc_sqw_freq_128Hz 0b00001000

#define rtc_sqw_freq_16Hz 0b00001011

#define rtc_sqw_freq_1Hz 0b00001111

#define rtc_sqw_freq_2048Hz 0b00000100

#define rtc_sqw_freq_256Hz 0b00000111

#define rtc_sqw_freq_2Hz 0b00001110

#define rtc_sqw_freq_32768Hz 0b00000001

#define rtc_sqw_freq_32Hz 0b00001010

#define rtc_sqw_freq_4096Hz 0b00000011

#define rtc_sqw_freq_4Hz 0b00001101

#define rtc_sqw_freq_512Hz 0b00000110

#define rtc_sqw_freq_64Hz 0b00001001

#define rtc_sqw_freq_8192Hz 0b00000010

#define rtc_sqw_freq_8Hz 0b00001100

Function Documentation

uns8 rtc_get_date ()

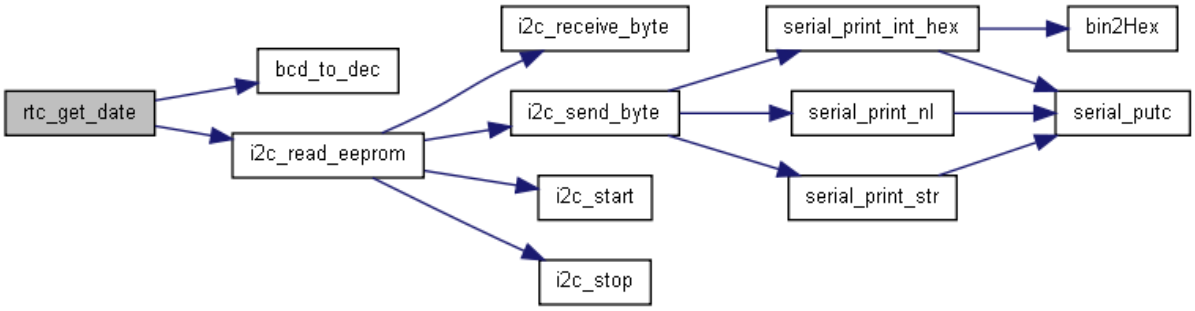
Returns the date in month from the m41t81s. The result is coverted to decimal from BCD and is ready to use. Range 1 through 28/29/30/31 depending on month

Get the date register from the m41t81s.

Returns the date in month from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 1 through 28/29/30/31 depending on month

```
65                      {
66     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_date_register));
67 }
```

Here is the call graph for this function:

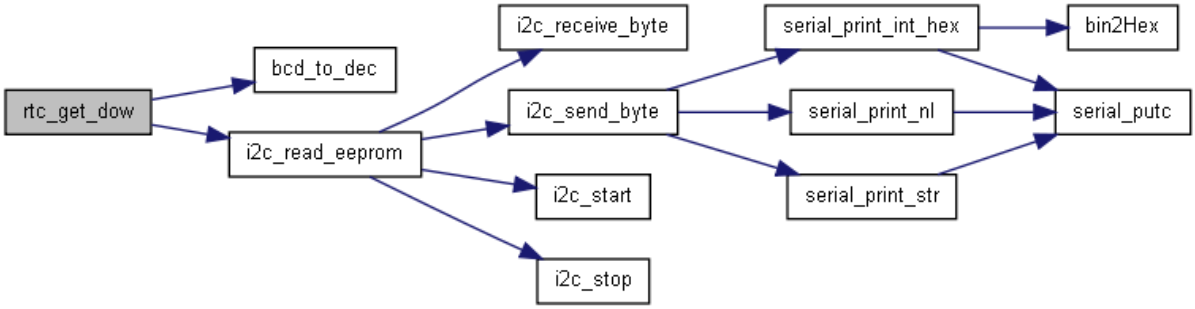


uns8 rtc_get_dow ()

Returns the day of the week from the m41t81s. The result is coverted to decimal from BCD and is ready to use. Range - 1 through 7

```
62                      {
63     return bcd_to_dec(i2c_read_eeprom(m41t81s_device_addr, m41t81s_dow_reg));
64 }
```

Here is the call graph for this function:



uns8 rtc_get_hours ()

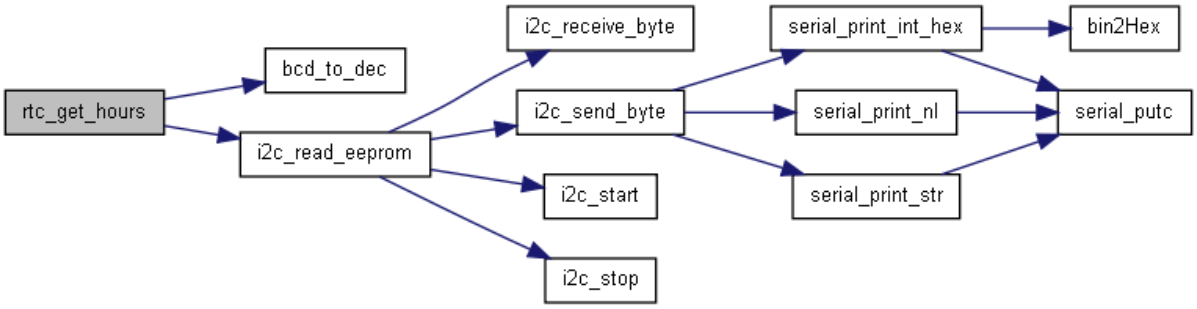
Returns hour from the m41t81s. The result is coverted to decimal from BCD and is ready to use. These routines assume the m41t81s is running in 24 hour mode. Range - 0 through 23

Get the decoded hours register from the m41t81s.

Returns hour from the ds1307. The result is coverted to decimal from BCD and is ready to use. These routines assume the ds1307 is running in 24 hour mode. Range - 0 through 23

```
50                      {
51
52 // Always assume it's in 24 hour mode
53
54     return bcd_to_dec(0b00111111 & i2c_read_eeprom(ds1307_device, ds1307_hours_register));
55 }
```

Here is the call graph for this function:



uns8 rtc_get_minutes ()

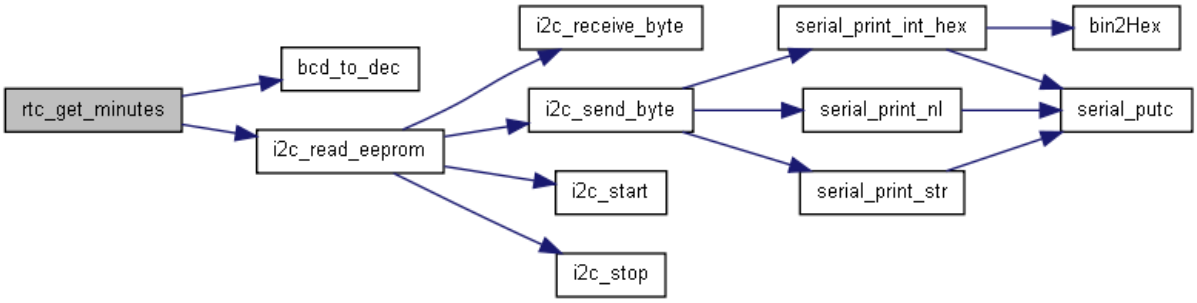
Returns the number of minutes past the hour from the m41t81s. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

Get the decoded minutes register from the m41t81s.

Returns the number of minutes past the hour from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

```
47                          {
48     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_minutes_register));
49 }
```

Here is the call graph for this function:



uns8 rtc_get_month ()

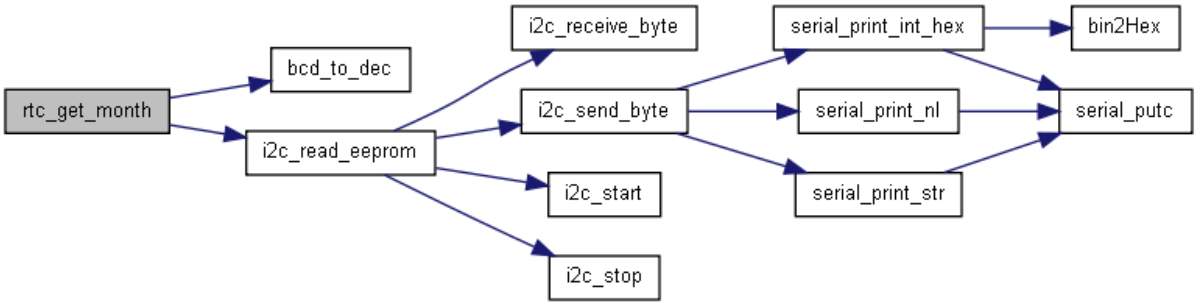
Returns the month of the year from the m41t81s. The result is coverted to decimal from BCD and is ready to use. Range 1 through 12

Get the month register from the m41t81s.

Returns the month of the year from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 1 through 12

```
69                          {
70     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_month_register));
71 }
```

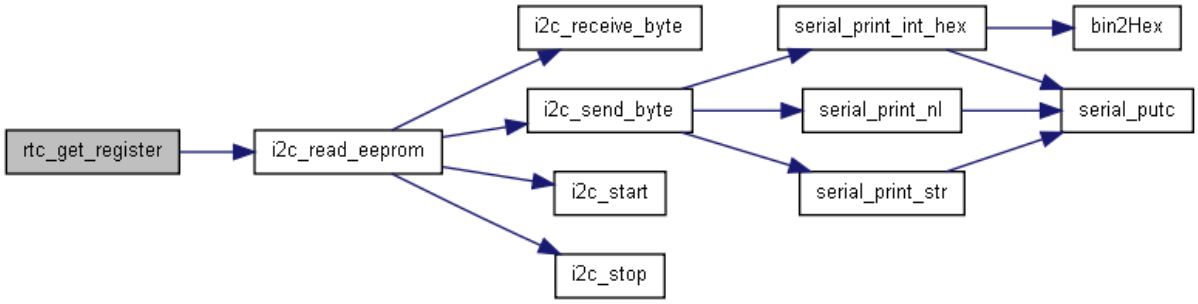
Here is the call graph for this function:



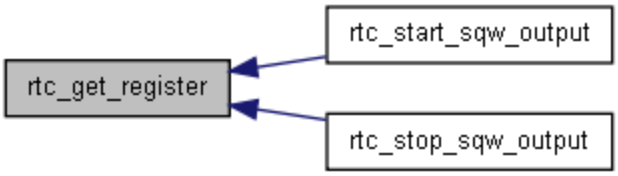
uns8 rtc_get_register (uns8 *reg*)

```
78                                  {
79     return i2c_read_eeprom(m41t81s_device_addr, reg);
80 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



uns8 rtc_get_seconds ()

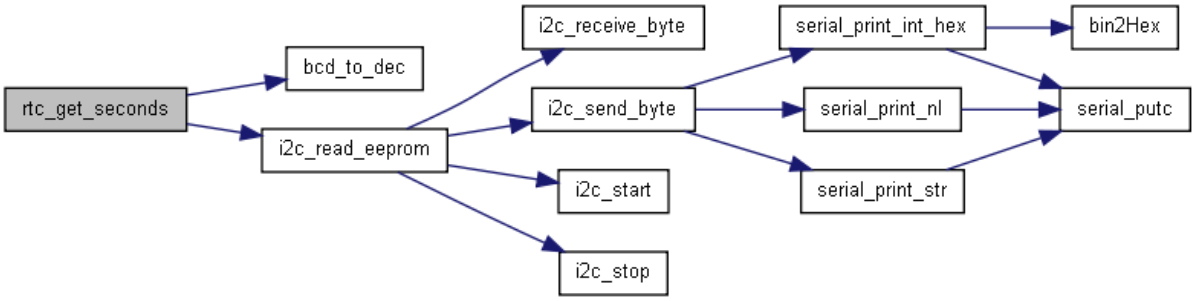
Returns seconds from the m41t81s. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

Get the decoded seconds register from the m41t81s.

Returns seconds from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range - 0 through 59

```
57                          {
58     return bcd_to_dec(0b01111111 & i2c_read_eeprom(ds1307_device, ds1307_seconds_register));
59 }
```

Here is the call graph for this function:



uns8 rtc_get_year ()

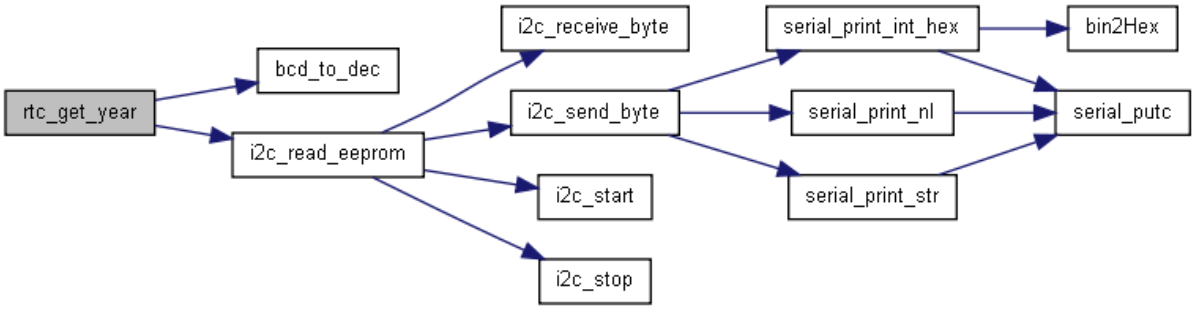
Returns the year from the m41t81s. The result is coverted to decimal from BCD and is ready to use. Range 0 through 99

Get the year register from the m41t81s.

Returns the year from the ds1307. The result is coverted to decimal from BCD and is ready to use. Range 0 through 99

```
72                      {
73     return bcd_to_dec(i2c_read_eeprom(ds1307_device, ds1307_year_register));
74 }
```

Here is the call graph for this function:



uns8 rtc_set_config (uns8 *config*)

Parameters:

config Value to set the config register to

Set the config register in the m41t81s.

Sets the config register in the ds1307.

Bit 7 - Out - Value on SQWE pin if not outputting square wave Bit 6 - 0 Bit 5 - 0 Bit 4 - SQWE - Enable square wave output Bit 3 - 0 Bit 2 - 0 Bit 1 - RS1 Bit 0 - RS0

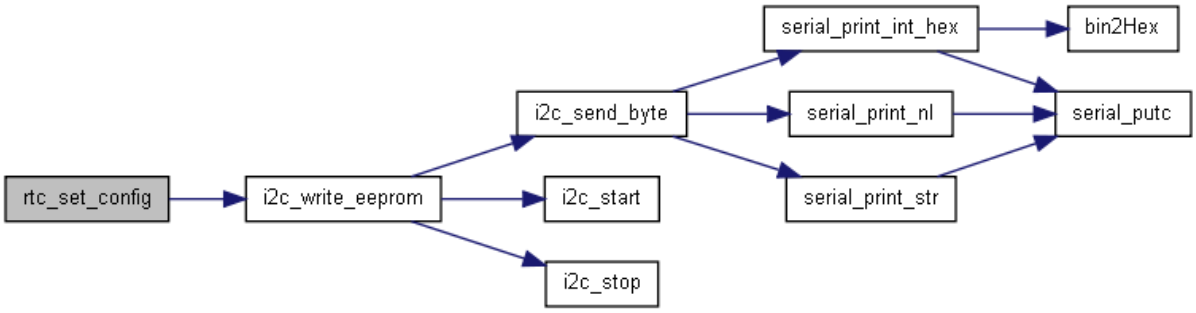
RS1/0 determin the speed of the square wave output. Set to 0/0 for 1 Hz.

Parameters:

config Value to set the config register to

```
80                                      {
81     i2c_write_eeprom(ds1307_device, ds1307_control_register, config);
82 }
```

Here is the call graph for this function:



void rtc_set_date (uns8 *date*)

Changes the date in the m41t81s.

Parameters:

seconds Value to set date to

Set the date register from the m41t81s.

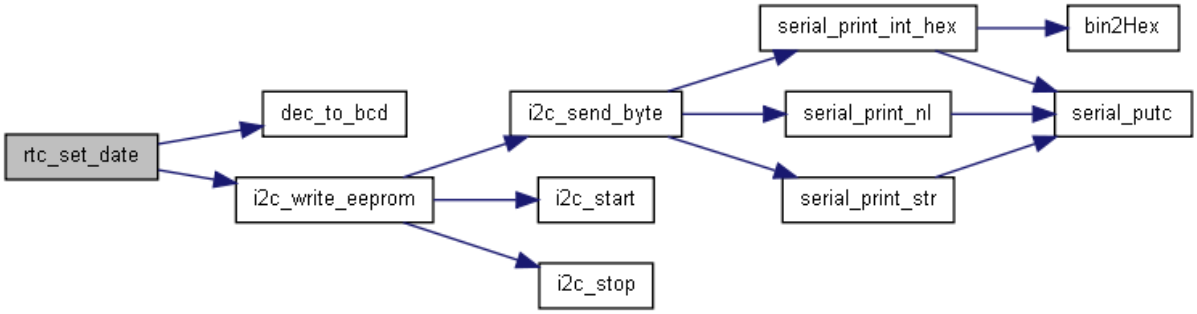
Changes the date in the ds1307.

Parameters:

seconds Value to set date to

```
102                                 {
103     i2c_write_eeprom(ds1307_device, ds1307_date_register, dec_to_bcd(date));
104 }
```

Here is the call graph for this function:



void rtc_set_day (uns8 *day*)

Changes the day of the week in the m41t81s.

Parameters:

seconds Value to set day to

Set the day of the week register from the m41t81s.

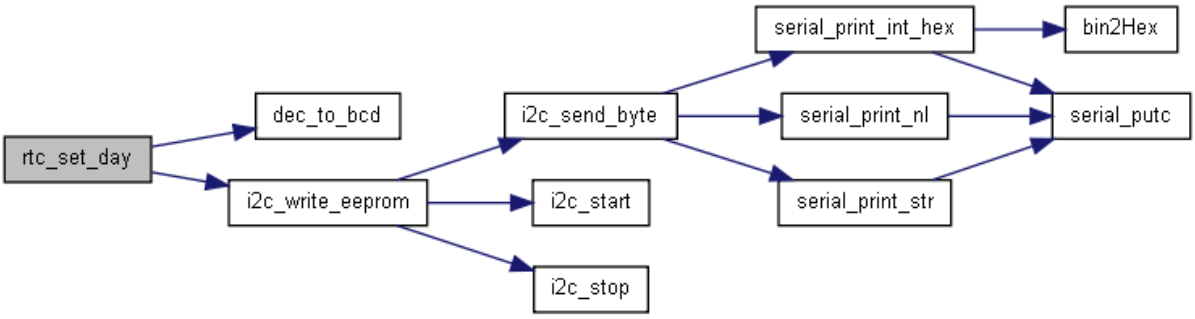
Changes the day of the week in the ds1307.

Parameters:

seconds Value to set day to

```
99                                  {
100     i2c_write_eeprom(ds1307_device, ds1307_day_register, dec_to_bcd(day));
101 }
```

Here is the call graph for this function:



void rtc_set_hours (uns8 *hours*)

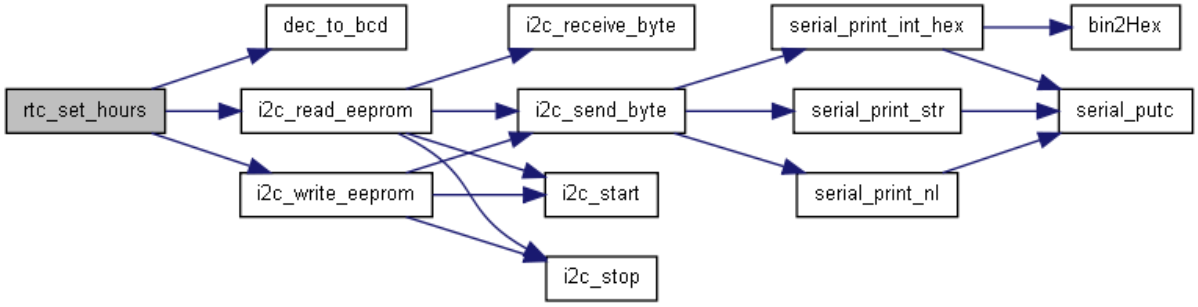
Changes the hours in the m41t81s. Forces the m41t81s into 24 hour mode.

Set the hours register in the m41t81s.

Changes the hours in the ds1307. Forces the ds1307 into 24 hour mode.

```
110                                 {
111     // by doing this we clear the 12/24 flag, making it 24 hour mode
112         i2c_write_eeprom(ds1307_device, ds1307_hours_register, dec_to_bcd(hours));
113 }
```

Here is the call graph for this function:



void rtc_set_minutes (uns8 *minutes*)

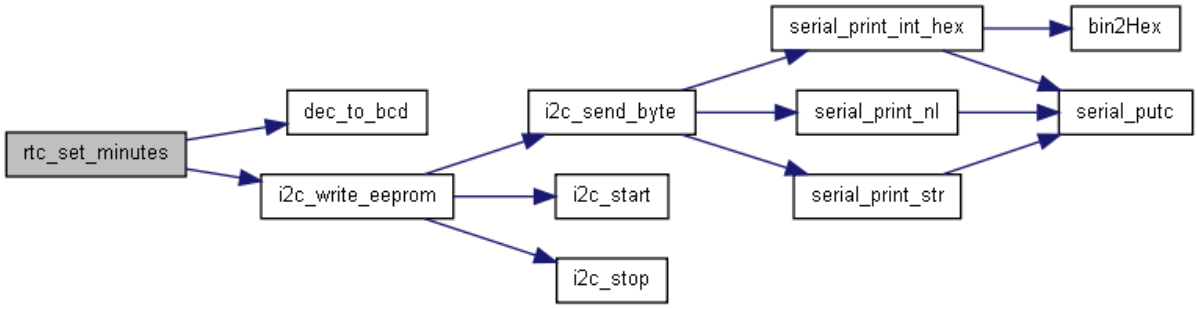
Changes the minutes in the m41t81s.

Parameters:

seconds Value to set minutes to

```
101                                 {
102     i2c_write_eeprom(m41t81s_device_addr, m41t81s_minutes_reg, dec_to_bcd(minutes));
103 }
```

Here is the call graph for this function:



void rtc_set_month (uns8 *month*)

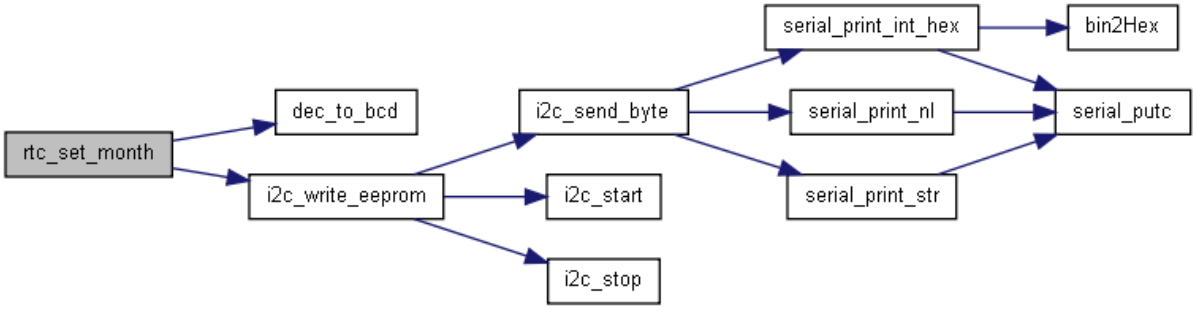
Changes the month in the m41t81s.

Set the month register in the m41t81s.

Changes the month in the ds1307.

```
115                                 {
116     i2c_write_eeprom(ds1307_device, ds1307_month_register, dec_to_bcd(month));
117 }
```

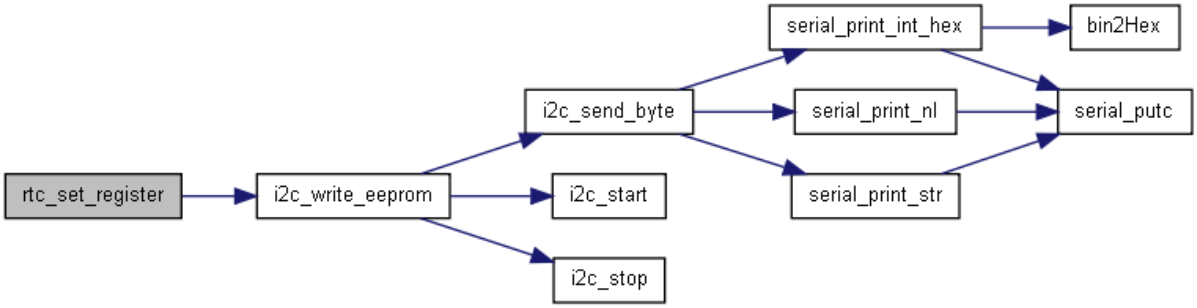
Here is the call graph for this function:



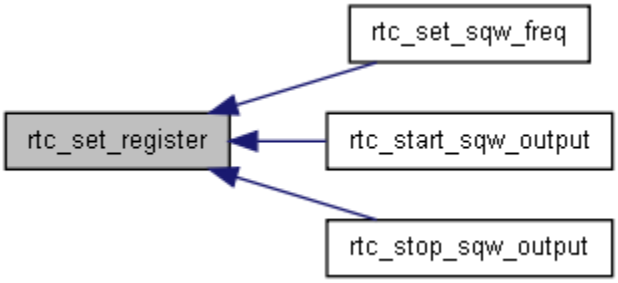
void rtc_set_register (uns8 *reg*, uns8 *data*)

```
82                                          {
83     i2c_write_eeprom(m41t81s_device_addr, reg, data);
84 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void rtc_set_seconds (uns8 *seconds*)

Changes the seconds in the m41t81s.

Parameters:

seconds Value to set seconds to

Set the seconds register in the m41t81s.

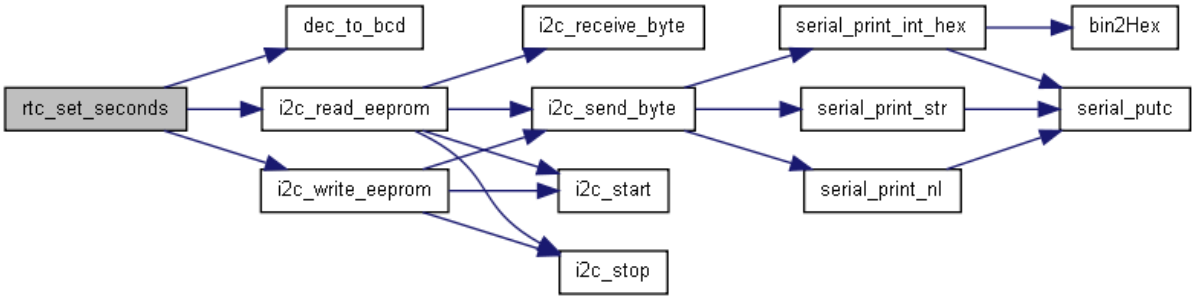
Changes the seconds in the ds1307.

Parameters:

seconds Value to set seconds to

```
106                                     {
107     i2c_write_eeprom(ds1307_device, ds1307_seconds_register, (0b10000000 &
i2c_read_eeprom(ds1307_device, ds1307_seconds_register)) + dec_to_bcd(seconds));
108 }
```

Here is the call graph for this function:



void rtc_set_sqw_freq (uns8 *freq*)

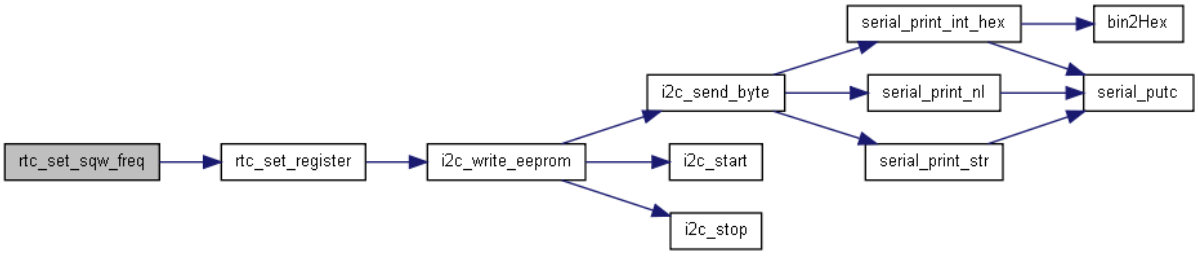
Use one of the following self explanitory defines:

rtc_sqw_freq_32768Hz rtc_sqw_freq_8192Hz rtc_sqw_freq_4096Hz rtc_sqw_freq_2048Hz rtc_sqw_freq_1024Hz rtc_sqw_freq_512Hz rtc_sqw_freq_256Hz rtc_sqw_freq_128Hz rtc_sqw_freq_64Hz rtc_sqw_freq_32Hz rtc_sqw_freq_16Hz rtc_sqw_freq_8Hz rtc_sqw_freq_4Hz rtc_sqw_freq_2Hz rtc_sqw_freq_1Hz

Note that on the m41t81s 18384Hz is not available.

```
125                                     {
126
127     freq = freq << 4;
128     rtc_set_register(m41t81s_sqw_reg, freq);
129 }
```

Here is the call graph for this function:

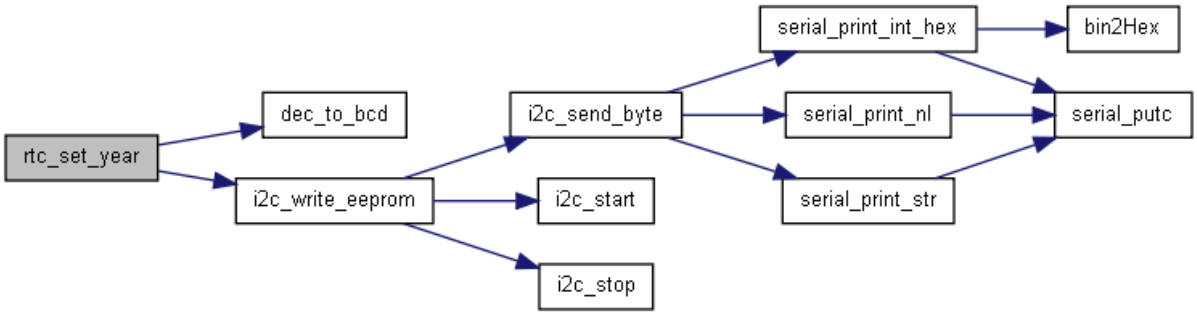


void rtc_set_year (uns8 *year*)

Changes the year in the m41t81s.

```
98                                  {
99      i2c_write_eeprom(m41t81s_device_addr, m41t81s_year_reg, dec_to_bcd(year));
100 }
```

Here is the call graph for this function:



void rtc_setup_io ()

Calls i2c_setup() to configure ports and pins ready for use

Setup ports and pins for use in the m41t81s.

Calls i2c_setup() to configure ports and pins ready for use

```
119                         {
120     i2c_setup_io();
```

121 }

Here is the call graph for this function:

void rtc_start_clock ()

Resume time in the m41t81s. Also resumes the paused time that happens upon non-battery backup start up (which allows you to read the time before "resuming" so you know how long the clock has been running on battery back up).

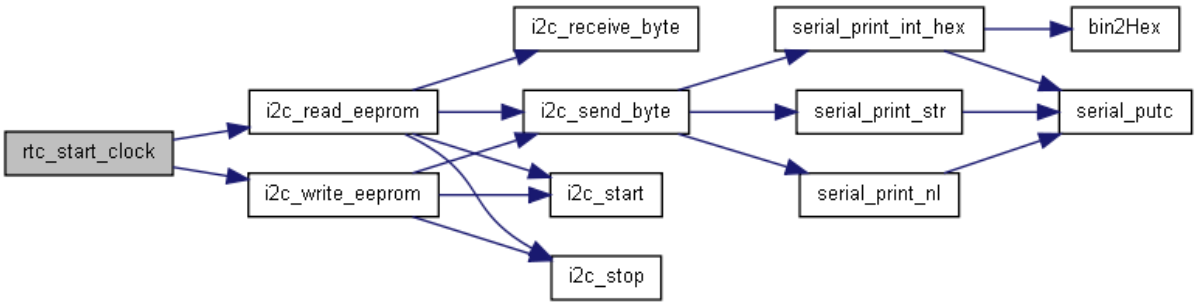
If you want to do this, read the time etc before calling rtc_start_clock();

Starts the clock in the m41t81s.

Resume time in the ds1307

```
89                          {
90      i2c_write_eeprom(ds1307_device, ds1307_seconds_register, 0b01111111 &
i2c_read_eeprom(ds1307_device, ds1307_seconds_register));
91 }
```

Here is the call graph for this function:

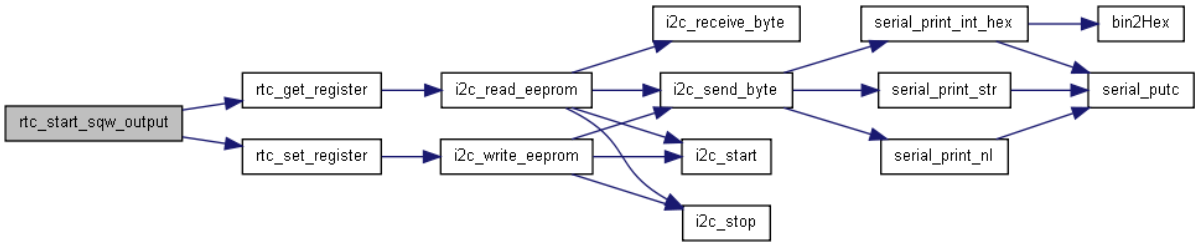


void rtc_start_sqw_output ()

Outputs desired frequency on the SQW output pin. To set the frequency, see rtc_set_sqw_freq(uns8 freq);

```
131                                {
132
133     rtc_set_register(m41t81s_alarm_month_reg, 0b01000000 |
rtc_get_register(m41t81s_alarm_month_reg));
134 }
```

Here is the call graph for this function:



void rtc_stop_clock ()

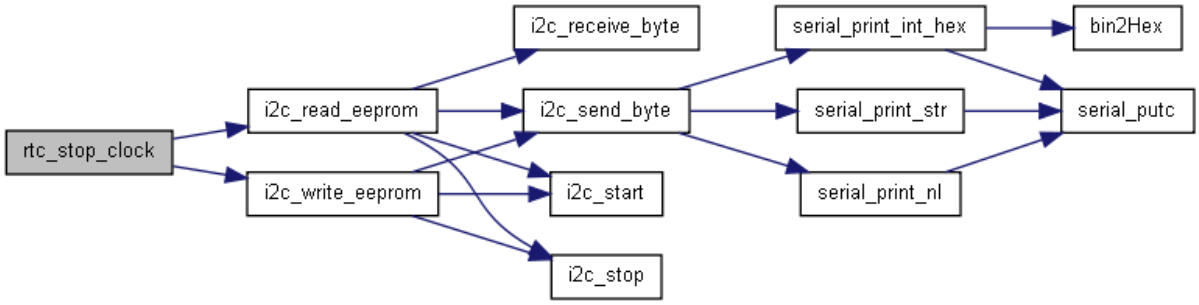
Pauses time in the m41t81s

Stop the clock in the m41t81s.

Pauses time in the ds1307

```
85                          {
86     i2c_write_eeprom(ds1307_device, ds1307_seconds_register, 0b10000000 |
i2c_read_eeprom(ds1307_device, ds1307_seconds_register));
87 }
```

Here is the call graph for this function:

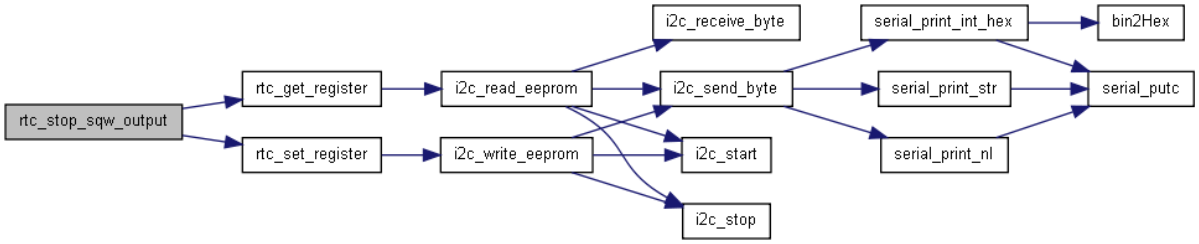


void rtc_stop_sqw_output ()

Stops square wave output

```
136                                {
137
138     rtc_set_register(m41t81s_alarm_month_reg, 0b10111111 |
rtc_get_register(m41t81s_alarm_month_reg));
139
140 }
```

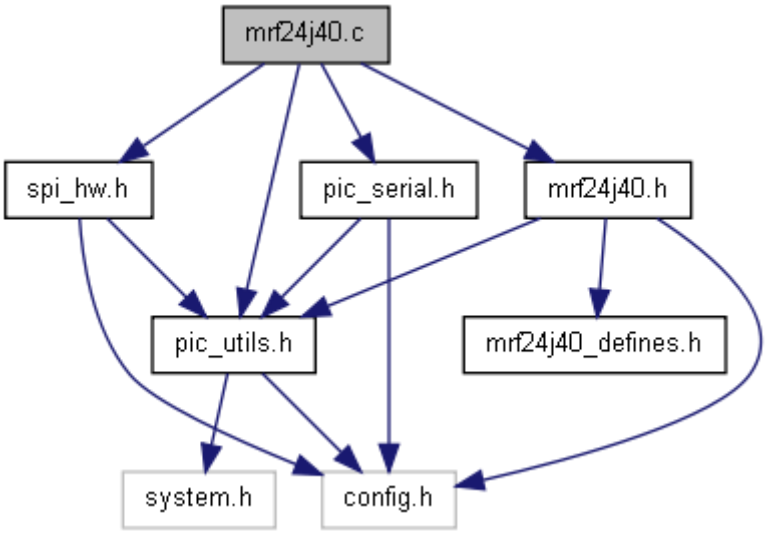
Here is the call graph for this function:



---

mrf24j40.c File Reference

Include dependency graph for mrf24j40.c:



Functions

- void mrf24h40_pan_association_requested ()
- void mrf24j40_active_channel_scan ()
- void mrf24j40_associate_to_pan ()
- void mrf24j40_flush_receive_buffer ()
- *Flush receive buffer of mrf24j40.* void mrf24j40_handle_isr ()
- *Interrupt service routine for mrf24j40 chip.* void mrf24j40_init ()
- *Initialises mrf24j40 chip ready for use.* void mrf24j40_init_coordinator ()
- uns8 mrf24j40_long_addr_read (uns16 addr)
- *Read data from long address of memory location in mrf24j40.* void mrf24j40_long_addr_write (uns16 addr, uns8 data)
- *Write data to long address memory location.* void mrf24j40_orphan_channel_scan ()
- void mrf24j40_realign_pan ()
- uns8 mrf24j40_receive (uns8 *data, uns8 bytes_to_receive)
- *Pull received data from buffer.* uns8 mrf24j40_scan_for_lowest_channel_ed ()
- *Scan all channels for lowest RF energy.* void mrf24j40_set_channel (uns8 channel)
- *Change channels.* void mrf24j40_set_extended_address (uns8 *_extended_address)
- *Set extended address.* void mrf24j40_set_pan_id (uns16 _pan_id)

- *Set PAN id.* void mrf24j40_set_short_address (uns16 _short_address)
- *Set short address.* void mrf24j40_setup_io ()
- *Setup ports/pins as inputs/outputs ready for use.* uns8 mrf24j40_short_addr_read (uns8 addr)
- *Read data from short address of memory location in mrf24j40.* void mrf24j40_short_addr_write (uns8 addr, uns8 data)
- *Write data to short address memory location.* void mrf24j40_start_pan ()
- void mrf24j40_transmit (uns8 *data, uns8 bytes_to_transmit)
- *Transmit raw data.* void mrf24j40_transmit_to_extended_address (uns8 frame_type, uns16 dest_pan_id, uns8 *dest_extended_address, uns8 *data, uns8 data_length, uns8 ack)
- *Transmit packet to extended address.* void mrf24j40_transmit_to_short_address (uns8 frame_type, uns16 dest_pan_id, uns16 dest_short_address, uns8 *data, uns8 bytes_to_transmit, uns8 ack)

Transmit packet to short address. Variables

- uns8 current_channel = 0
- uns8 data_sequence_number
- uns8 extended_address [8] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff}
- uns16 pan_id = 0xffff
- uns16 short_address = 0xffff

Function Documentation

void mrf24h40_pan_association_requested ()

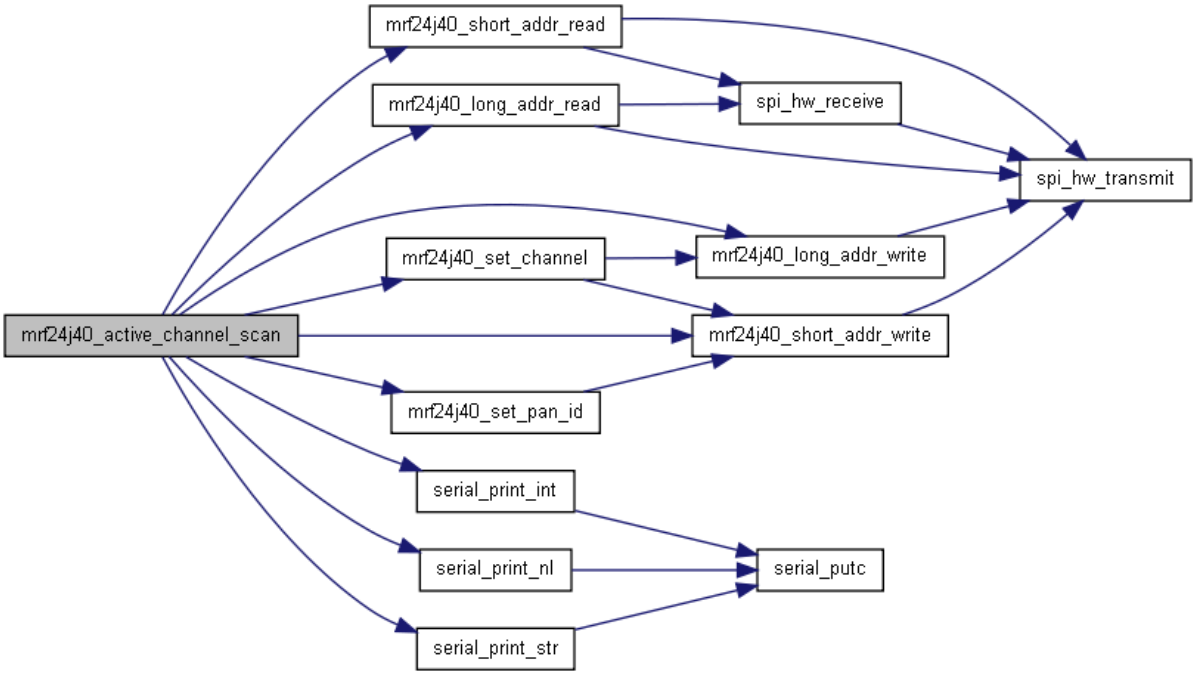
```
242                                             {
243     // allocate 16bit short address
244         // fffe stuff in 7.5.3.1
245     // generate association response command (see 7.3.2)
246     // send it indirect, using 7.5.6.3 method
247     //  add to pending transactions
248 }
```

void mrf24j40_active_channel_scan ()

```
134                                         {
135
136 uns8 rxflush;
137 uns16 prev_pan_id;
138 uns8 channel;
139 uns8 highest_on_channel;
140 uns16 scan_count;
141 uns8 bbreg6;
142 uns8 rssi;
143
144     // receive only beacon frames
145
146     rxflush = mrf24j40_short_addr_read(RXFLUSH);
147     set_bit(rxflush, RXFLUSH_BCNONLY);
148     mrf24j40_short_addr_write(RXFLUSH, rxflush);
149
150
151     // ignore pending data
152
153
154     // store pan id
155     prev_pan_id = pan_id;
156
157     // set pan id to ffff
158     mrf24j40_set_pan_id(0xffff);
```

```
159
160     for (channel = MRF_FIRST_CHANNEL; channel <= MRF_LAST_CHANNEL; channel++) {
161         serial_print_str("Searching on channel: ");
162         serial_print_int(channel);
163         serial_print_nl();
164 
165         // switch channel
166         mrf24j40_set_channel(channel);
167 
168         // send beacon request
169         uns8 fc_msb = 0b11001100;   // 64 bit dest (10,11) 64 bit src (14,15)
170                     // 0b00000011;
171         uns8 fc_lsb = 0b00000001;   // data, no pan id compression
172 
173         data_sequence_number++;
174         uns8 bytes_to_transmit = 5;//??
175         uns8 header_length = 3+8+8+2+2; // Just two bytes of frame control + sequence number
176         uns8 frame_length = header_length + bytes_to_transmit;
177 
178          //       TxBuffer[0] = 0x03;   // fc lsb
179          //       TxBuffer[1] = 0x08;   // fc msb
180          //       TxBuffer[2] = IEEESeqNum++;     //sequence number
181          //       TxBuffer[3] = 0xFF;
182          //       TxBuffer[4] = 0xFF;
183           //      TxBuffer[5] = 0xFF;
184           //      TxBuffer[6] = 0xFF;
185            //     TxBuffer[7] = 0x07;
186 
187     mrf24j40_long_addr_write(0x00, header_length);
188     mrf24j40_long_addr_write(0x01, frame_length);
189     mrf24j40_long_addr_write(0x02, fc_lsb); // swapped
190     mrf24j40_long_addr_write(0x03, fc_msb);
191     mrf24j40_long_addr_write(0x04, data_sequence_number);
192 
193     mrf24j40_long_addr_write(0x05, 0x05);   // dest pan id LSB
194     mrf24j40_long_addr_write(0x06, 0x00);   // MSB
195 
196 
197         // wait [aBaseSuperframeDuration * (2^n + 1)] symbols, n=scanduration parameter
198         // store info on pan in pan description structure
199         // beacon is unique if pan id and source address haven't been seen before on current
channel
200 
201         highest_on_channel = 0;
202         for (scan_count = 0; scan_count < 1000; scan_count++) {
203             mrf24j40_short_addr_write(BBREG6, 1 << BBREG6_RSSIMODE1);
204             do {
205                 bbreg6 = mrf24j40_short_addr_read(BBREG6);
206             } while (!test_bit(bbreg6, BBREG6_RSSIRDY));
207             rssi = mrf24j40_long_addr_read(RSSI);
208             if (rssi > highest_on_channel) {
209                 highest_on_channel = rssi;
210             }
211         }
212         serial_print_str("Highest on channel = ");
213         serial_print_int(highest_on_channel);
214         serial_print_nl();
215 
216 
217 
218     }
219     // restore pan id
220     // switch channel
221     //.. receive all frames
222     serial_print_str("/-----\n");
223 
224 }
```

Here is the call graph for this function:



void mrf24j40_associate_to_pan ()

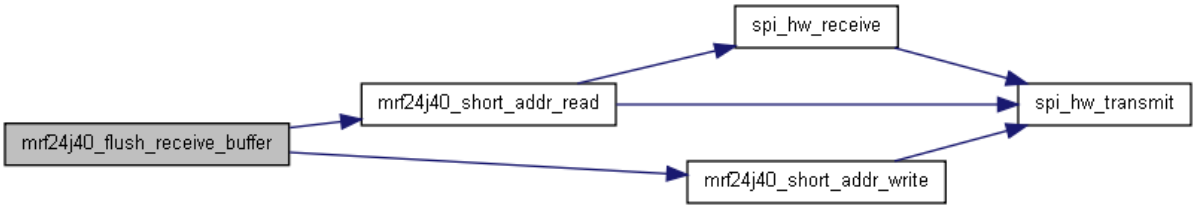
```
236                                     {
237     // send associate to pan command
238     // wait macResponseWaitTime
239
240 }
```

void mrf24j40_flush_receive_buffer ()

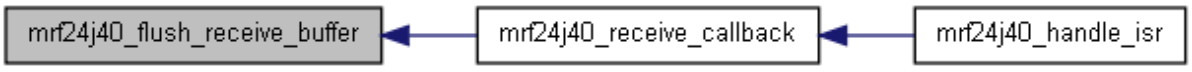
No need to call this routine normally, except according to mrf24j40 errata (promiscuous mode)

```
52                                      {
53
54     uns8 rxflush;
55
56     rxflush = mrf24j40_short_addr_read(RXFLUSH);     // Get rxflush
57     set_bit(rxflush, RXFLUSH_RXFLUSH);  // Set flush bit
58     mrf24j40_short_addr_write(RXFLUSH, rxflush);     // Push it back to the mrf
59 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

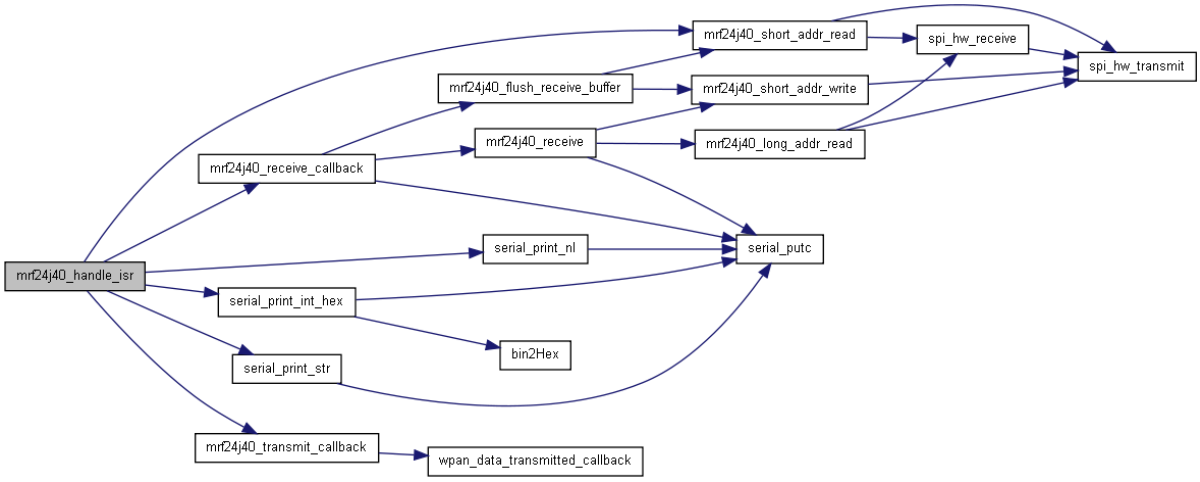


void mrf24j40_handle_isr ()

Call this routine when the mrf24j40 indicates an interrupt condition, or called regularly.

```
287                             {
288
289 uns8 intstat;
290
291     // first we need to get a copy of intstat to find out what happened
292     intstat = mrf24j40_short_addr_read(INTSTAT);
293
294     // mrf24j40 intstat register is cleared on read
295     if (test_bit(intstat, INTSTAT_RXIF)) {
296         serial_print_str("R");
297         // handle receive
298         mrf24j40_receive_callback();
299     }
300     if (test_bit(intstat, INTSTAT_TXNIF)) {
301         serial_print_str("Tx complete txstat=0x");
302         uns8 stat = mrf24j40_short_addr_read(TXSTAT);
303         mrf24j40_transmit_callback(stat & 0b00000001,   // success = 0
304                                    stat >> 6,    // retries
305                                    test_bit(stat, TXSTAT_CCAFAIL)); // due to cca
failure?
306         serial_print_int_hex(stat);
307         serial_print_nl();
308     }
309
310 }
```

Here is the call graph for this function:



void mrf24j40_init ()

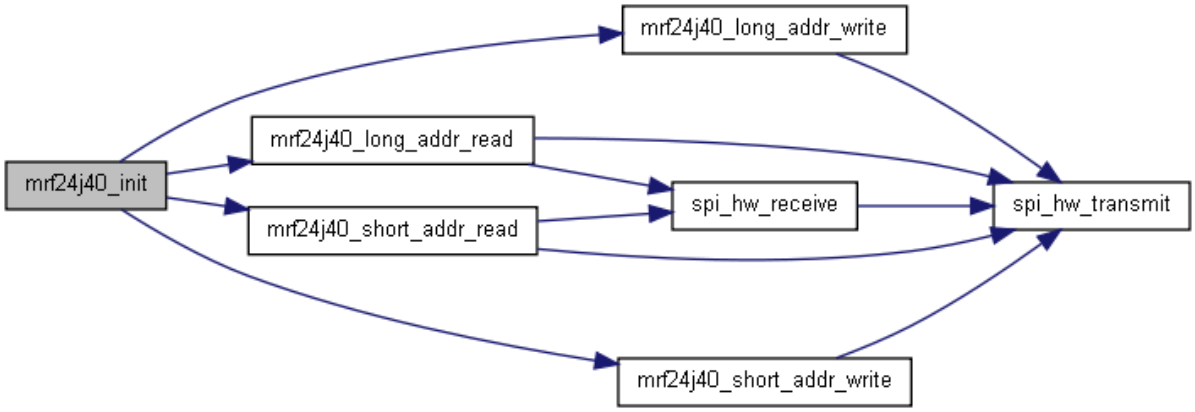
Sets int pin as input and cs pin as output.

```
499                         {
500
501 uns8 check;
502
503     // Example steps to initialize the MRF24J40:
504
505     // 1. SOFTRST (0x2A) = 0x07 - Perform a software Reset. The bits will be automatically
cleared to '0' by hardware.
506     mrf24j40_short_addr_write(SOFTRST, 0x07);
507
508     // Wait for soft reset to complete
```

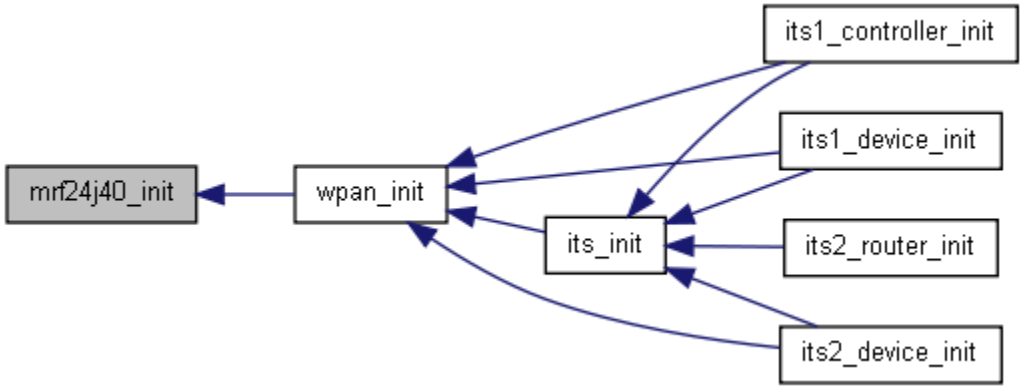
```
509
510     do {
511         check = mrf24j40_short_addr_read(SOFTRST);
512     } while (check != 0);
513
514     #if defined(ENABLE_PA_LNA) && defined(MRF24J40MB)
515         // Turn on PA/LNA if we have one
516         mrf24j40_long_addr_write(TESTMODE, 0x0f);
517     #endif
518
519     // 2. PACON2 (0x18) = 0x98 - Initialize FIFOEN = 1 and TXONTS = 0x6.
520     mrf24j40_short_addr_write(PACON2, 0x98);
521     // 3. TXSTBL (0x2E) = 0x95 - Initialize RFSTBL = 0x9.
522     mrf24j40_short_addr_write(TXSTBL, 0x95);
523
524     // wait for mrf to be in receive mode
525
526     do {
527         check = mrf24j40_long_addr_read(RFSTATE);
528     } while (check & 0xa0 != 0xa0);
529
530     mrf24j40_long_addr_write(RFCON0, 0x03); // or three?
531     // mrf24j40_long_addr_write(RFCON1, 0x02);  // VCO control mode (1 or 2?)
532     // miwi stack source is at odds with data sheet here
533
534     // 4. RFCON1 (0x201) = 0x01 - Initialize VCOOPT = 0x01.
535     mrf24j40_long_addr_write(RFCON1, 0x01);
536     // 5. RFCON2 (0x202) = 0x80 - Enable PLL (PLLEN = 1).
537     mrf24j40_long_addr_write(RFCON2, 0x80);
538     // 6. RFCON6 (0x206) = 0x90 - Initialize TXFIL = 1 and 20MRECVR = 1.
539     mrf24j40_long_addr_write(RFCON6, 0x90);
540     // 7. RFCON7 (0x207) = 0x80 - Initialize SLPCLKSEL = 0x2 (100 kHz Internal oscillator).
541     mrf24j40_long_addr_write(RFCON7, 0x08);
542     // 8. RFCON8 (0x208) = 0x10 - Initialize RFVCO = 1.
543     mrf24j40_long_addr_write(RFCON8, 0x10);
544     // 9. SLPCON1 (0x220) = 0x21 - Initialize CLKOUTEN = 1 and SLPCLKDIV = 0x01.
545     mrf24j40_long_addr_write(SLPCON1, 0x21);
546
547     //Configuration for nonbeacon-enabled devices
548     //(see Section MRF datasheet 3.8 “Beacon-Enabled and Nonbeacon-Enabled Networks”):
549     //10. BBREG2 (0x3A) = 0x80 - Set CCA mode to ED.
550     mrf24j40_short_addr_write(BBREG2, 0x80);
551     // 11. RSSITHCCA (0x3F) = 0x60 - Set CCA ED threshold.
552     // IH: I think the datasheet is wrong, the mem address is "CCAEDTH", even though
553     // it's referred to as RSSITHCCA and CCAMODE in the documentation...
554
555     mrf24j40_short_addr_write(CCAEDTH, 0x60);
556     // 12. BBREG6 (0x3E) = 0x40 - Set appended RSSI value to RXFIFO.
557     mrf24j40_short_addr_write(BBREG6, 1 << BBREG6_RSSIMODE2);
558     // 13. Enable interrupts - See Section 3.3 “Interrupts”.
559     mrf24j40_short_addr_write(MRF_INTCON, (1 << MRF_INTCON_RXIE) & (1 << MRF_INTCON_TXNIE));
560     // 14. Set channel - See Section 3.4 “Channel Selection”.
561     mrf24j40_long_addr_write(RFCON0, 11);
562
563     #if defined(ENABLE_PA_LNA) && defined(MRF24J40MB)
564     // There are special MRF24J40 transceiver output power
565     // setting for Microchip MRF24J40MB module.
566     // To do: Correct settings for other locations
567
568         #if APPLICATION_SITE == EUROPE
569             // MRF24J40 output power set to be -14.9dB
570             mrf24j40_long_addr_write(RFCON3, 0x70);
571         #else
572             // MRF24J40 output power set to be -1.9dB - Stock setting
573             // mrf24j40_long_addr_writeRFCON3, 0x18);
574
575             // MRF24J40 output power set to be -1.2dB
576             // mrf24j40_long_addr_write(RFCON3, 0x10);
577
578             // MRF24J40 output power set to be -0.5dB
579             // mrf24j40_long_addr_write(RFCON3, 0x08);
```

```
580
581             // MRF24J40 output power set to be -0dB -> 22,5 dBm output power
582             mrf24j40_long_addr_write(RFCON3, 0x00);
583         #endif
584     #else
585         // power level to be 0dBms
586         mrf24j40_long_addr_write(RFCON3,0x00);
587     #endif
588
589     // 15. RFCTL (0x36) = 0x04 - Reset RF state machine.
590     mrf24j40_short_addr_write(RFCTL, 0x04);
591
592     // 16. RFCTL (0x36) = 0x00.
593     mrf24j40_short_addr_write(RFCTL, 0x00);
594     // 17. Delay at least 192 is.
595     delay_us(200);
596
597     //  RXMCR.PANCOORD = 1 if coordinator, 0 is default (not coordinator)
598
599     //mrf24j40_short_addr_write(RXMCR, 0x01); // promiscuous mode, no coord
600
601     data_sequence_number = 0;
602
603 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



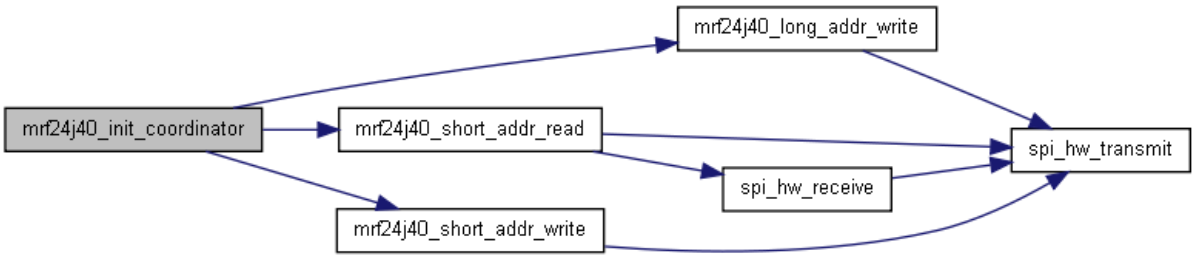
void mrf24j40_init_coordinator ()

```
606                                 {
```

```
607 uns8 val;
608
609     // Example steps to initialize the MRF24J40:
610
611     // 1. SOFTRST (0x2A) = 0x07 – Perform a software Reset. The bits will be automatically
cleared to ‘0’ by hardware.
612     mrf24j40_short_addr_write(SOFTRST, 0x07);
613     // 2. PACON2 (0x18) = 0x98 – Initialize FIFOEN = 1 and TXONTS = 0x6.
614     mrf24j40_short_addr_write(PACON2, 0x98);
615     // 3. TXSTBL (0x2E) = 0x95 – Initialize RFSTBL = 0x9.
616     mrf24j40_short_addr_write(TXSTBL, 0x95);
617     // 4. RFCON1 (0x201) = 0x01 – Initialize VCOOPT = 0x01.
618     mrf24j40_long_addr_write(RFCON1, 0x01);
619     // 5. RFCON2 (0x202) = 0x80 – Enable PLL (PLLEN = 1).
620     mrf24j40_long_addr_write(RFCON2, 0x008);
621     // 6. RFCON6 (0x206) = 0x90 – Initialize TXFIL = 1 and 20MRECVR = 1.
622     mrf24j40_long_addr_write(RFCON6, 0x90);
623     // 7. RFCON7 (0x207) = 0x80 – Initialize SLPCLKSEL = 0x2 (100 kHz Internal oscillator).
624     mrf24j40_long_addr_write(RFCON7, 0x08);
625     // 8. RFCON8 (0x208) = 0x10 – Initialize RFVCO = 1.
626     mrf24j40_long_addr_write(RFCON8, 0x10);
627     // 9. SLPCON1 (0x220) = 0x21 – Initialize CLKOUTEN = 1 and SLPCLKDIV = 0x01.
628     mrf24j40_long_addr_write(SLPCON1, 0x21);
629
630     //Configuration for nonbeacon-enabled devices
631     //(see Section MRF datasheet 3.8 “Beacon-Enabled and Nonbeacon-Enabled Networks”):
632     //10. BBREG2 (0x3A) = 0x80 – Set CCA mode to ED.
633     mrf24j40_short_addr_write(BBREG2, 0x80);
634     // 11. RSSITHCCA (0x3F) = 0x60 – Set CCA ED threshold.
635     // IH: I think the datasheet is wrong, the mem address is "CCAEDTH", even though
636     // it's referred to as RSSITHCCA and CCAMODE in the documentation...
637     mrf24j40_short_addr_write(CCAEDTH, 0x60);
638     // 12. BBREG6 (0x3E) = 0x40 – Set appended RSSI value to RXFIFO.
639     mrf24j40_short_addr_write(BBREG6, 0x40);
640     // 13. Enable interrupts – See Section 3.3 “Interrupts”.
641     mrf24j40_short_addr_write(MRF_INTCON, (1 << MRF_INTCON_RXIE) & (1 << MRF_INTCON_TXNIE));
642     // 14. Set channel – See Section 3.4 “Channel Selection”.
643     mrf24j40_long_addr_write(RFCON0, 11);
644     // 15. RFCTL (0x36) = 0x04 – Reset RF state machine.
645     mrf24j40_short_addr_write(RFCTL, 0x04);
646
647     // 16. RFCTL (0x36) = 0x00.
648     mrf24j40_short_addr_write(RFCTL, 0x00);
649     // 17. Delay at least 192 ìs.
650     delay_us(200);
651
652 //  RXMCR.PANCOORD = 1 if coordinator, 0 is default (not coordinator)
653 //  TXMCR.SLOTTED = 1 for slotted CSMA-CA mode, 0 is default (unslotted CSMA_CA mode)
654
655     //mrf24j40_short_addr_write(RXMCR, 0x01); // promiscuous mode, no coord
656
657     //Set the PANCOORD (RXMCR 0x00<3>) bit = 1 to configure as PAN coordinator.
658     val = mrf24j40_short_addr_read(RXMCR);
659     set_bit(val, RXMCR_PANCOORD);
660     mrf24j40_short_addr_write(RXMCR, val);
661
662     //2. Set the SLOTTED (TXMCR 0x11<5>) bit = 1 to use Slotted CSMA-CA mode.
663     // turn on slotted
664     val = mrf24j40_short_addr_read(TXMCR);
665     set_bit(val, TXMCR_SLOTTED);
666     mrf24j40_short_addr_write(TXMCR, val);
667
668     // 3. Load the beacon frame into the TXBFIFO (0x080-0x0FF).
669     uns8 header_length = 2+1+10;
670     uns8 frame_length = header_length + 6;
671     uns8 superframe_spec_field_l = 0b10001000; // bo = 8, so = 8
672     uns8 superframe_spec_field_h = 0b11001000; // AP/PC/R/BLE/final cap slot
673
674     mrf24j40_long_addr_write(0x080+0, header_length);
675     mrf24j40_long_addr_write(0x080+1, frame_length);
676     mrf24j40_long_addr_write(0x080+2, 0x00);   //frame control - beacon
```

```
677     mrf24j40_long_addr_write(0x080+3, 0x80);   //addressing mode - source only
678     mrf24j40_long_addr_write(0x080+4, data_sequence_number++);   //seq number
679     mrf24j40_long_addr_write(0x080+5, 5);   //PANID LSB
680     mrf24j40_long_addr_write(0x080+6, 0);   //PANID MSB
681     mrf24j40_long_addr_write(0x080+7, extended_address[7]); // LSB
682     mrf24j40_long_addr_write(0x080+8, extended_address[6]);
683     mrf24j40_long_addr_write(0x080+9, extended_address[5]);
684     mrf24j40_long_addr_write(0x080+10, extended_address[4]);
685     mrf24j40_long_addr_write(0x080+11, extended_address[3]);
686     mrf24j40_long_addr_write(0x080+12, extended_address[2]);
687     mrf24j40_long_addr_write(0x080+13, extended_address[1]);
688     mrf24j40_long_addr_write(0x080+14, extended_address[0]);    // MSB
689
690     mrf24j40_long_addr_write(0x080+15, superframe_spec_field_l);
691     mrf24j40_long_addr_write(0x080+16, superframe_spec_field_h);
692
693     mrf24j40_long_addr_write(0x080+17, 0);  // GTS
694     mrf24j40_long_addr_write(0x080+18, 0);  // Pending addresses
695     mrf24j40_long_addr_write(0x080+19, 0x4d);   // Protocol ID
696     mrf24j40_long_addr_write(0x080+20, 0x10);   // version
697     // no payload
698
699
700     // 4. Set the TXBMSK (TXBCON1 0x25<7>) bit = 1 to mask the beacon interrupt mask.
701     //val = mrf24j40_short_addr_read(TXBCON1);
702     //set_bit(val, TXBCON1_TXBMSK);
703     //mrf24j40_short_addr_write(RXMCR);
704
705     //5. Program the CAP end slot (ESLOTG1 0x13<3:0>) value. If the coordinator supports
706     //Guaranteed Time Slot operation, refer to Section 3.8.1.5 "Configuring Beacon-Enabled
707     //GTS Settings for PAN Coordinator" below.
708
709     //6. Calibrate the Sleep Clock (SLPCLK) frequency. Refer to Section 3.15.1.2 "Sleep Clock
Calibration".
710     //7. Set WAKECNT (SLPACK 0x35<6:0>) value = 0x5F to set the main oscillator (20 MHz)
711     //start-up timer value.
712     //8. Program the Beacon Interval into the Main Counter,
713     //MAINCNT (0x229<1:0>, 0x228, 0x227, 0x226),
714     //and Remain Counter, REMCNT (0x225, 0x224), according to BO and SO values. Refer to
715     //Section 3.15.1.3 "Sleep Mode Counters".
716     //9. Configure the BO (ORDER 0x10<7:4>) and SO (ORDER 0x10<3:0>) values. After configuring
717     //BO and SO, the beacon frame will be sent immediately.
718             //BO3(1) BO2(1) BO1(1) BO0(1) SO3(1) SO2(1) SO1(1) SO0(1)
719     //mrf24j40 short addr write(ORDER, 0b10001000);
720
721     data_sequence_number = 0;
722
723 }
```

Here is the call graph for this function:



uns8 mrf24j40_long_addr_read (uns16 *addr*)

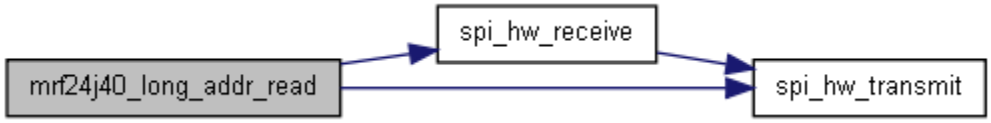
Returns the value in the memory location specified.

Parameters:

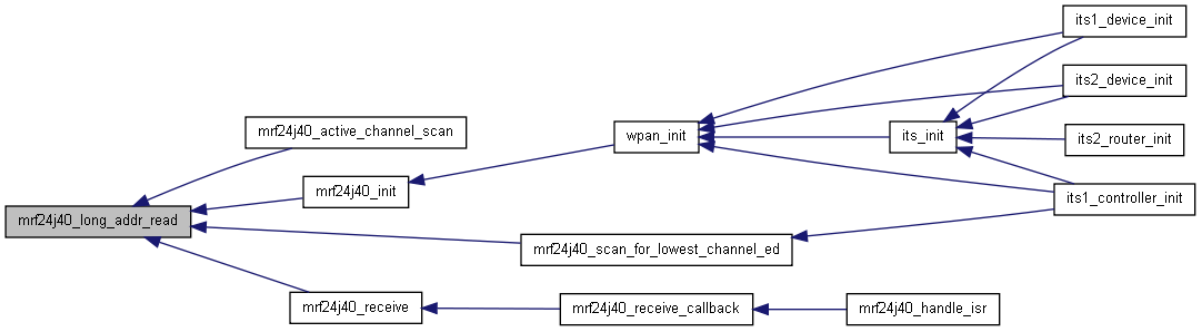
addr Long address memory location (see mrf24h40_defines.h)

```
749                                             {
750
751     uns8 result;
752
753     clear_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
754
755     addr = addr & 0b0000001111111111;   // <9:0> bits
756     addr = addr << 5;
757     set_bit(addr, 15);  // long addresss
758     spi_hw_transmit(addr >> 8);
759     spi_hw_transmit(addr & 0x00ff);
760     result = spi_hw_receive();
761
762     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
763
764     return result;
765 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_long_addr_write (uns16 *addr*, uns8 *data*)

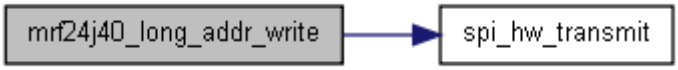
Sets the memory location to the value specified

Parameters:

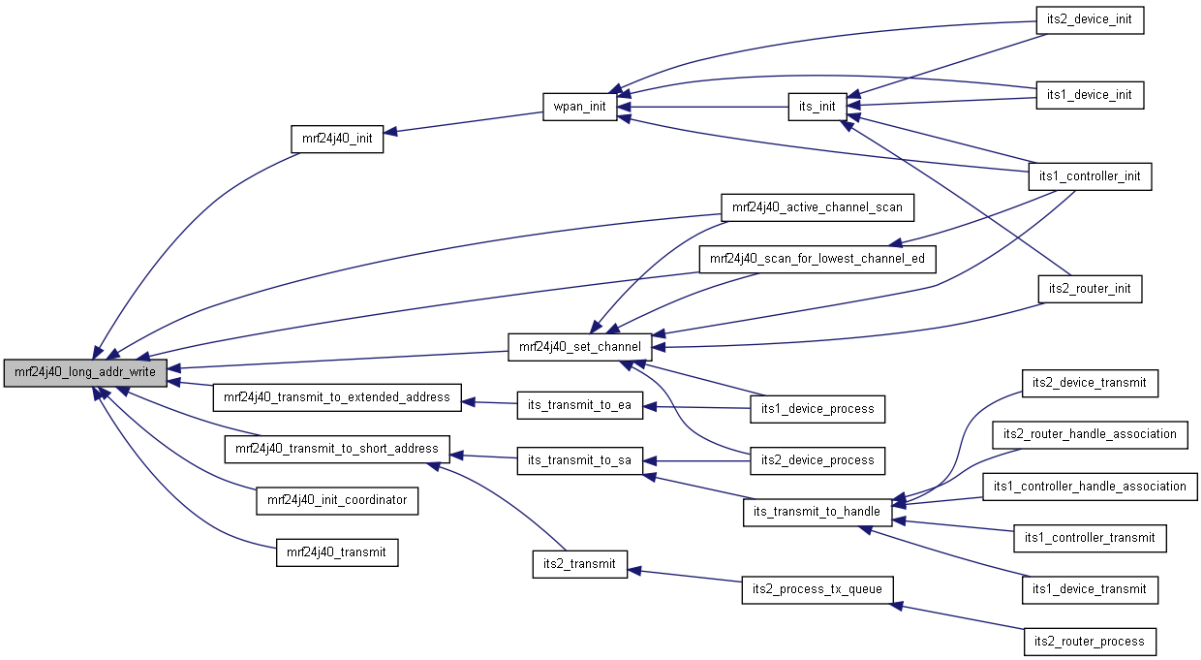
addr Long address memory location (see mrf24h40_defines.h)
data Value that the memory location should be set to

```
767                                                     {
768
769     clear_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
770
771     addr = addr & 0b0000001111111111;   // <9:0> bits
772     addr = addr << 5;
773
774     set_bit(addr, 15);  // long addresss
775     set_bit(addr, 4);   // set for write
776
777     spi_hw_transmit(addr >> 8 );
778     spi_hw_transmit(addr & 0x00ff);
779     spi_hw_transmit(data);
780
781     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
782 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_orphan_channel_scan ()

```
226                                     {
227 }
```

void mrf24j40_realign_pan ()

```
233                                 {
234 }
```

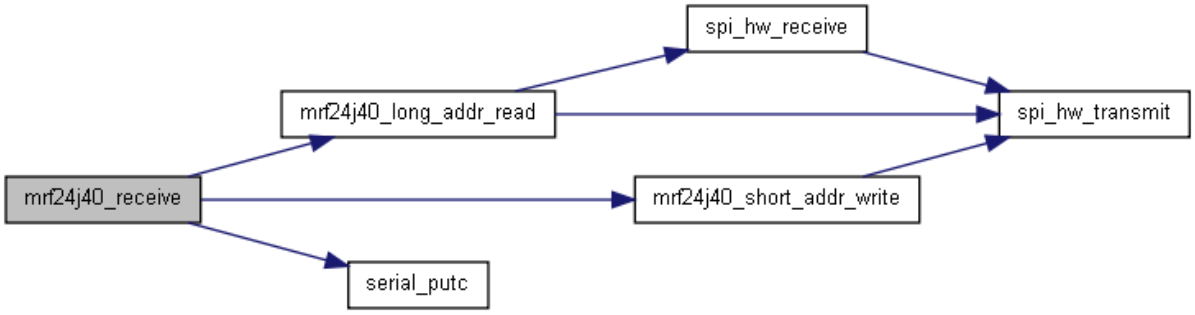
uns8 mrf24j40_receive (uns8 * *data*, uns8 *bytes_to_receive*)

Once an interrupt has occured and we know it is because a packet has been received, this routine will pull the data from the mrf receive buffer. This needs to be done quickly so that the next packet is not lost.

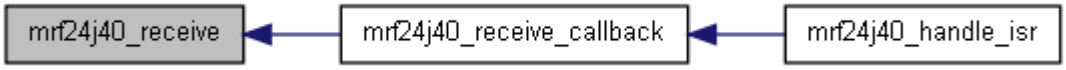
```
312                                                         {
313
314 uns8 frame_length;
315 uns16 frame pos;
316 uns8 buffer_count;
317 /*
318 1. Receive RXIF interrupt.
319 2. Disable host microcontroller interrupts.
320 3. Set RXDECINV = 1; disable receiving packets off air.
321 4. Read address, 0x300; get RXFIFO frame length value.
```

```
322 5. Read RXFIFO addresses, 0x301 through (0x300 + Frame Length + 2); read packet data plus LQI
and RSSI.
323 6. Clear RXDECINV = 0; enable receiving packets.
324 7. Enable host microcontroller interrupts.
325 */
326     serial_putc('a');
327     // Disable reading packets off air
328     mrf24j40_short_addr_write(BBREG1, 1 << BBREG1_RXDECINV);
329     serial_putc('b');
330 
331     frame_pos = 0x300;
332     frame_length = mrf24j40_long_addr_read(frame_pos++);
333     buffer_count = 0;
334     serial_putc('c');
335 
336     while ((buffer_count <= bytes_to_receive) && (buffer_count <= frame_length + 2)) {
337         //0x301 through (0x300 + Frame Length + 2 ); read packet data plus LQI and RSSI.
338         data[buffer_count++] = mrf24j40_long_addr_read(frame_pos++);
339     }
340     serial_putc('d');
341 
342     // Re-enable reading packets off air
343     mrf24j40_short_addr_write(BBREG1, 0);
344     serial_putc('e');
345 
346     return buffer_count - 1;
347 
348 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



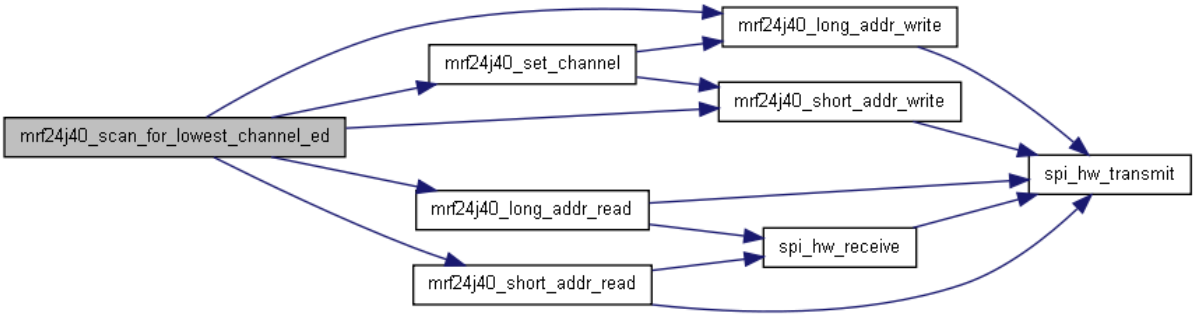
uns8 mrf24j40_scan_for_lowest_channel_ed ()

Scans through all channels and reports the channel with the lowest Energy Detection level.

```
77                                          {
78 
79 uns8 rssi;
80 uns8 channel;
81 uns16 scan_count;
82 uns8 highest_on_channel;
83 uns8 lowest_channel = MRF_LAST_CHANNEL;
84 uns8 lowest_ed = 0xff;
85 uns8 bbreg6;
86 
87     // We should really ignore all packets here (do to)
88 
89     #if defined(ENABLE_PA_LNA)
```

```
90          mrf24j40_long_addr_write(TESTMODE, 0x08);               // Disable automatic switch on
PA/LNA
91          mrf24j40_short_addr_write(TRISGPIO, 0x0F);         // Set GPIO direction
92          mrf24j40_short_addr_write(GPIO, 0x0C);            // Enable LNA
93      #endif
94
95
96      for (channel = MRF_FIRST_CHANNEL; channel <= MRF_LAST_CHANNEL; channel++) {
97
98          // switch channel
99          mrf24j40_set_channel(channel);
100
101          highest_on_channel = 0;
102          for (scan_count = 0; scan_count < 1000; scan_count++) {
103              mrf24j40_short_addr_write(BBREG6, 1 << BBREG6_RSSIMODE1);
104              do {
105                  bbreg6 = mrf24j40_short_addr_read(BBREG6);
106              } while (!test_bit(bbreg6, BBREG6_RSSIRDY));
107              rssi = mrf24j40_long_addr_read(RSSI);
108              if (rssi > highest_on_channel) {
109                  highest_on_channel = rssi;
110              }
111          }
112
113          if (highest_on_channel < lowest_ed) {
114              lowest_ed = highest_on_channel;
115              lowest_channel = channel;
116          }
117      }
118      mrf24j40_short_addr_write(BBREG6, 1 << BBREG6_RSSIMODE2);    // Back to mode 2 (rssi in
pkt)
119
120      // Should stop ignoring all packets now if we started ignoring them earlier
121      // (to do)
122
123      #if defined(ENABLE_PA_LNA)
124          mrf24j40_short_addr_write(GPIO, 0);
125          mrf24j40_short_addr_write(TRISGPIO, 0x00);
126          mrf24j40_long_addr_write(TESTMODE, 0x0F);
127      #endif
128
129
130      return lowest_channel;
131  }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void mrf24j40_set_channel (uns8 *channel*)

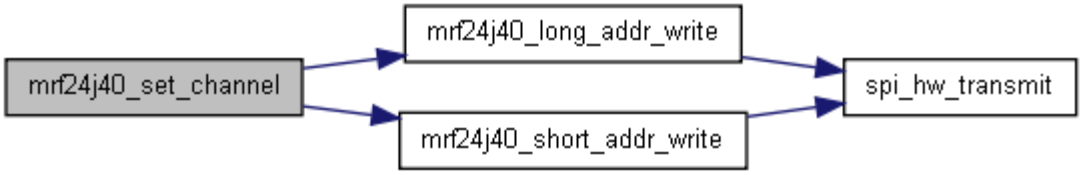
Change to the specified channel, in the range MRF_FIRST_CHANNEL to MRF_LAST_CHANNEL.

Parameters:

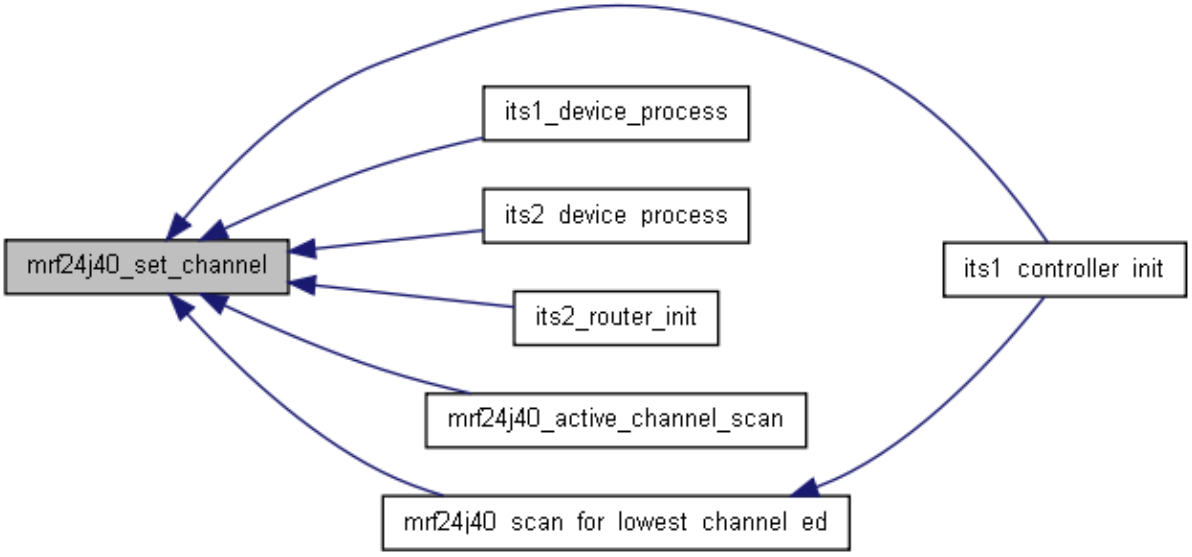
channel Channel to change to.

```
63                                          {
64
65      current_channel = channel;
66      channel = channel - 11;
67      channel = 0x02 + 0x10 * channel;
68
69      mrf24j40_long_addr_write(RFCON0, channel);  // Set channel
70      mrf24j40_short_addr_write(RFCTL, 0x04); // RFCTL (0x36) = 0x04 - Reset RF state machine.
71      mrf24j40_short_addr_write(RFCTL, 0x00); // RFCTL (0x36) = 0x00
72
73      delay_us(200);  // Delay at least 192us
74  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_set_extended_address (uns8 * *_extended_address*)

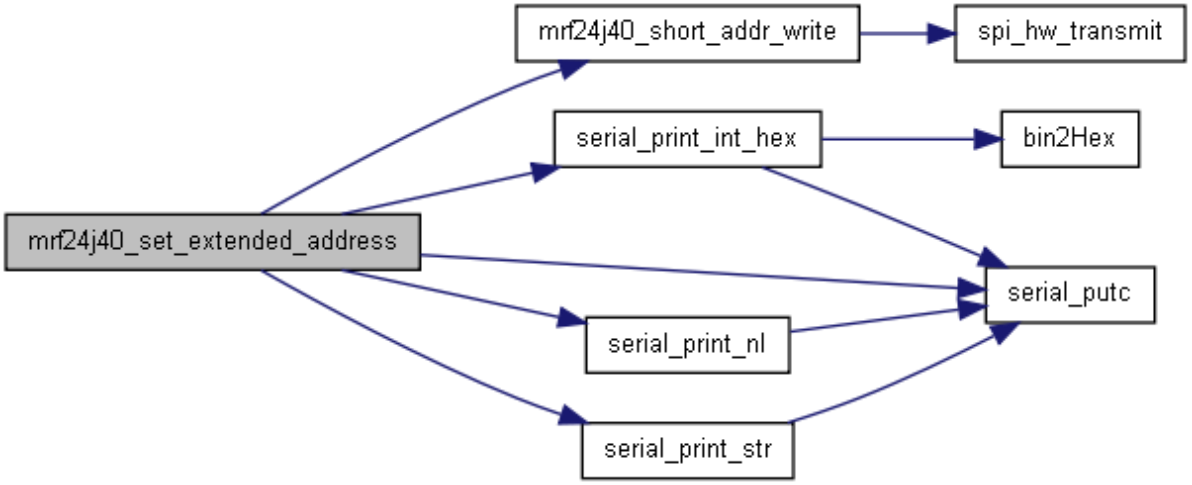
Set IEEE 802.15.4 extended address.

Sets the 64 bit extended address of the module. Pass a pointer to an 8 byte uns8 array containing the address.

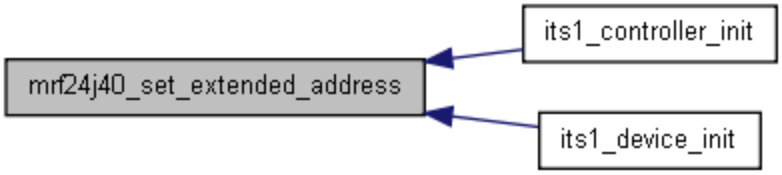
```
259                                                         {
260     uns8 count;
261     uns8 mem_pos = EADR7;
262     serial_print_str("Setting EA to: ");
263     for (count = 0; count < 8; count++) {
264         extended_address[count] = _extended_address[count];
265         mrf24j40_short_addr_write(mem_pos--, extended_address[count]);
```

```
266         serial_print_int_hex(extended_address[count]);
267         serial_putc(' ');
268 
269     }
270     serial_print_nl();
271 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

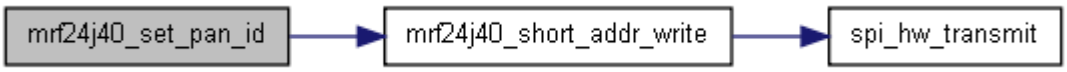


void mrf24j40_set_pan_id (uns16 *_pan_id*)

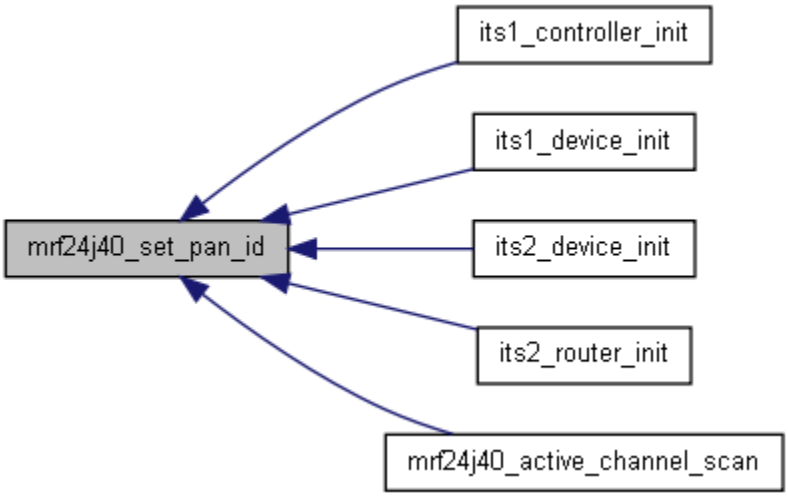
Sets the 16 bit PAN id of the module.

```
252                                     {
253     pan_id = _pan_id;
254     mrf24j40_short_addr_write(PANIDL, pan_id & 0xff);
255     mrf24j40_short_addr_write(PANIDH, pan_id >> 8);
256 
257 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

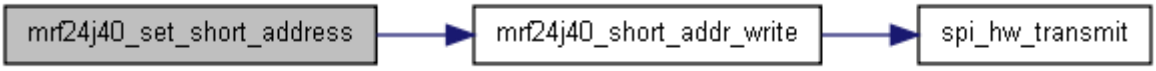


void mrf24j40_set_short_address (uns16 *_short_address*)

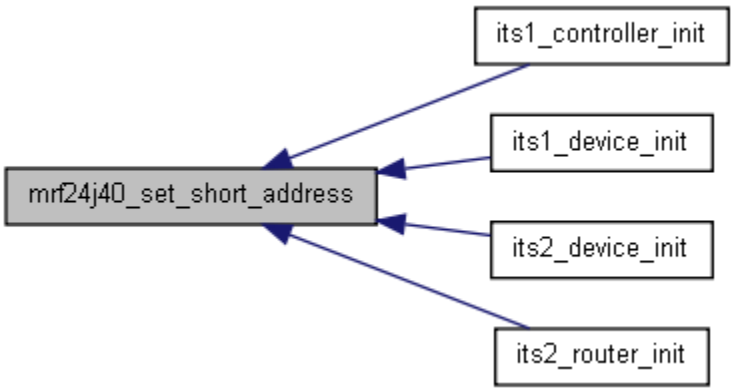
Sets the 16 bit short address of the module.

```
273                                                 {
274
275     // Keep hold of the short address
276     short_address = _short_address;
277
278     // Tell the mrf about it
279     mrf24j40_short_addr_write(SADRL, short_address & 0xff);
280     mrf24j40_short_addr_write(SADRH, short_address >> 8);
281
282 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



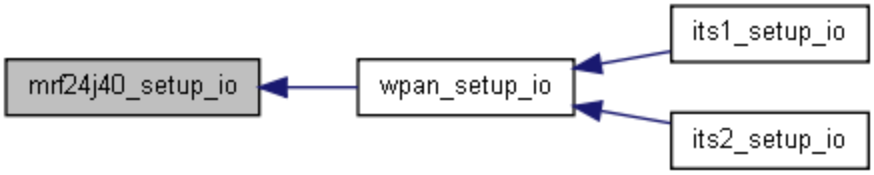
void mrf24j40_setup_io ()

Sets int pin as input and cs pin as output.

```
785                         {
786
```

```
787     make_input(mrf24j40_int_port, mrf24j40_int_pin);
788     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin); // keep high
789     make_output(mrf24j40_cs_port, mrf24j40_cs_pin);
790
791 }
```

Here is the caller graph for this function:



uns8 mrf24j40_short_addr_read (uns8 *addr*)

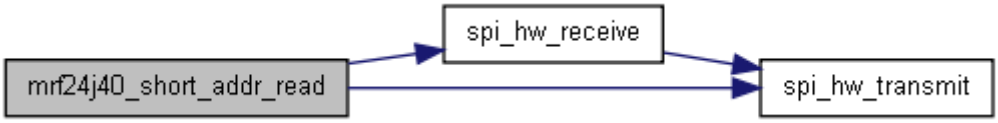
Returns the value in the memory location specified.

Parameters:

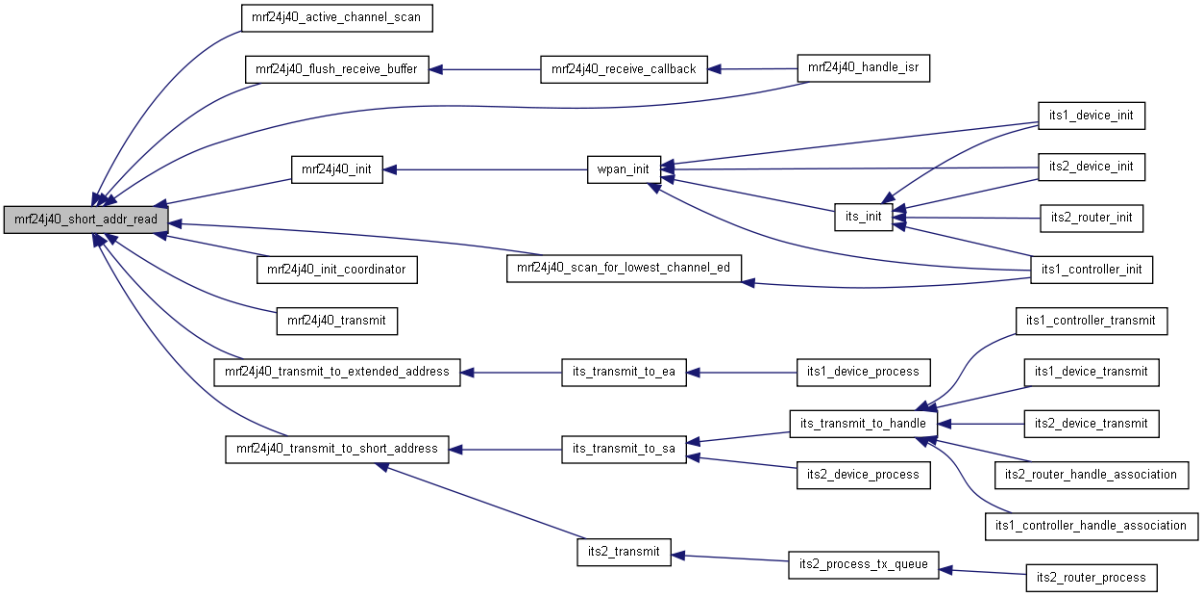
addr Short address memory location (see mrf24h40_defines.h)

```
725                                         {
726
727     clear_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
728     addr = addr & 0b00111111;   // <5:0> bits
729     addr = addr << 1;   // LSB = 0, means read
730     spi_hw_transmit(addr);
731     uns8 result = spi_hw_receive();
732
733     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
734     return result;
735 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_short_addr_write (uns8 *addr*, uns8 *data*)

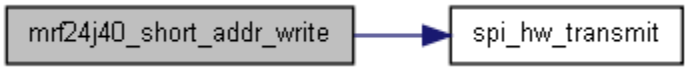
Sets the memory location to the value specified

Parameters:

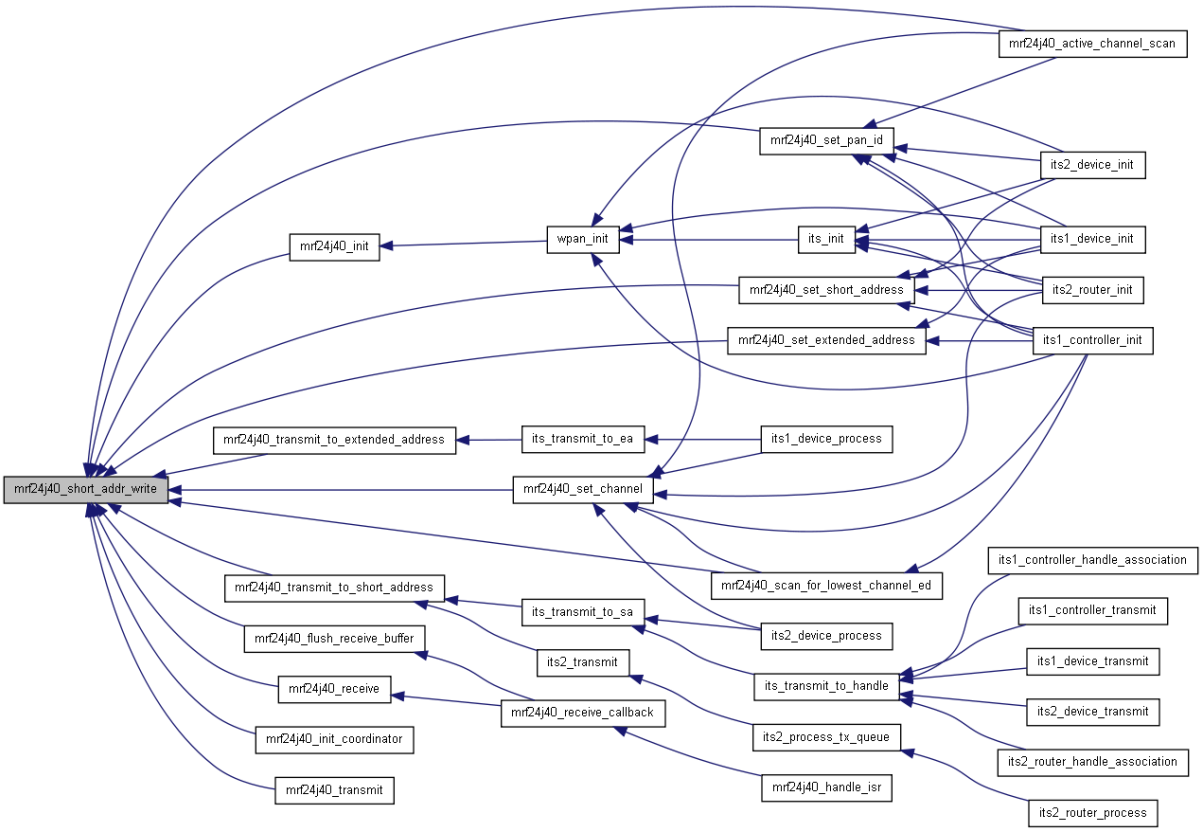
addr Short address memory location (see mrf24h40_defines.h)
data Value that the memory location should be set to

```
737                                                     {
738     clear_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
739     addr = addr & 0b00111111;   // <5:0> bits
740     addr = addr << 1;   // LSB = 0, means read
741     set_bit(addr, 0);   // write
742
743     spi_hw_transmit(addr);
744     spi_hw_transmit(data);
745
746     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
747 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_start_pan ()

```
229                             {
230
231 }
```

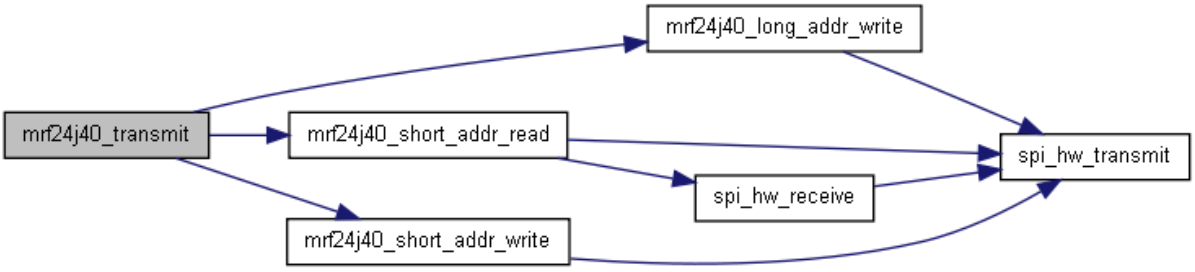
void mrf24j40_transmit (uns8 * *data*, uns8 *bytes_to_transmit*)

Transmit a raw packet - this assumes you have already created an 802.15.4 compatible packet. Normally you would use transmit_to_extended_adddress or transmit_to_short_address instead.

```
470                                                             {
471
472
473     uns8 fc_lsb = 0b01000001;
474     uns8 fc_msb = 0b00000000;
475
476
477     data_sequence_number++;
478     uns8 header_length = 3; // Just two bytes of frame control + sequence number, no addrs
479     uns8 frame_length = header_length + bytes_to_transmit;
480
481     mrf24j40_long_addr_write(0x00, header_length);
482     mrf24j40_long_addr_write(0x01, frame_length);
483     mrf24j40_long_addr_write(0x02, fc_lsb);
484     mrf24j40_long_addr_write(0x03, fc_msb);
485     mrf24j40_long_addr_write(0x04, data_sequence_number);
486
487     for (uns8 count=0;  count < bytes_to_transmit; count++) { // check format here
488         mrf24j40_long_addr_write(count+header_length+2,  data[count]);
489     }
```

```
490
491     uns8 txncon = mrf24j40_short_addr_read(TXNCON);
492     set_bit(txncon, TXNCON_TXNTRIG);
493     mrf24j40_short_addr_write(TXNCON, txncon);
494
495
496 }
```

Here is the call graph for this function:



void mrf24j40_transmit_to_extended_address (uns8 *frame_type*, uns16 *dest_pan_id*, uns8 * *dest_extended_address*, uns8 * *data*, uns8 *data_length*, uns8 *ack*)

Requests that the mrf24j40 transmit the packet to the specified extended address

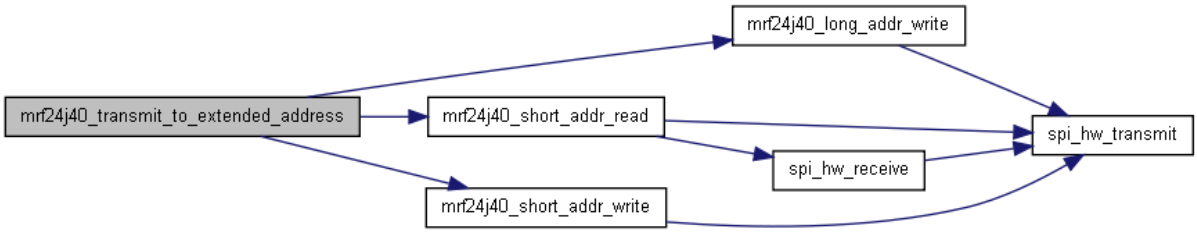
Parameters:

frame_type 802.15.4 frame type
dest_pan_id PAN id of destination
dest_extended_address Pointer to uns8 array indicating extended address of destination
data Pointer to uns8 array of bytes to transmit
bytes_to_transmit
ack Either MRF_ACK or MRF_NO_ACK depending if you want hardware acknowledgement

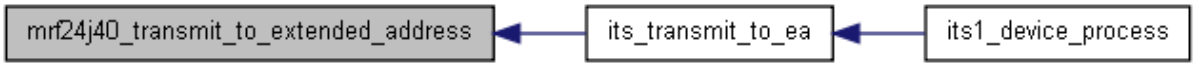
```
351                                                                                 {
352
353     // See notes below on frame control bytes format:
354     uns8 fc_msb = 0b11001100;   // 64 bit dest (10,11) 64 bit src (14,15)
355     uns8 fc_lsb = 0b00000000 | frame_type;  // pan id compression=0, data
356
357     if (ack) {
358         set_bit(fc_lsb, 5); // ack bit
359     }
360
361     data_sequence_number++;
362
363     uns8 header_length = 3+8+8+2+2;
364     uns8 frame_length = header_length + data_length;
365
366     // Write out data to mrf
367
368     mrf24j40_long_addr_write(0x00, header_length);
369     mrf24j40_long_addr_write(0x01, frame_length);
370
371     mrf24j40_long_addr_write(0x02, fc_lsb);
372     mrf24j40_long_addr_write(0x03, fc_msb);
373     mrf24j40_long_addr_write(0x04, data_sequence_number);
374
375     mrf24j40_long_addr_write(0x05, dest_pan_id & 0xff); // dest pan id LSB
376     mrf24j40_long_addr_write(0x06, dest_pan_id >> 8);   // MSB
377
378     mrf24j40_long_addr_write(0x07, dest_extended_address[7]); // LSB
379     mrf24j40_long_addr_write(0x08, dest_extended_address[6]);
380     mrf24j40_long_addr_write(0x09, dest_extended_address[5]);
381     mrf24j40_long_addr_write(0x0a, dest_extended_address[4]);
382     mrf24j40_long_addr_write(0x0b, dest_extended_address[3]);
383     mrf24j40_long_addr_write(0x0c, dest_extended_address[2]);
```

```
384     mrf24j40_long_addr_write(0x0d, dest_extended_address[1]);
385     mrf24j40_long_addr_write(0x0e, dest_extended_address[0]); // MSB
386
387     mrf24j40_long_addr_write(0x0f, pan_id & 0xff);  // src pan id LSB
388     mrf24j40_long_addr_write(0x10, pan_id >> 8);    // MSB
389
390
391     mrf24j40_long_addr_write(0x11, extended_address[7]);    // LSB
392     mrf24j40_long_addr_write(0x12, extended_address[6]);
393     mrf24j40_long_addr_write(0x13, extended_address[5]);
394     mrf24j40_long_addr_write(0x14, extended_address[4]);
395     mrf24j40_long_addr_write(0x15, extended_address[3]);
396     mrf24j40_long_addr_write(0x16, extended_address[2]);
397     mrf24j40_long_addr_write(0x17, extended_address[1]);
398     mrf24j40_long_addr_write(0x18, extended_address[0]);    // MSB
399
400     for (uns8 count=0;  count < data_length; count++) {
401         mrf24j40_long_addr_write(count+header_length+2,  data[count]);
402     }
403
404     uns8 txncon = mrf24j40_short_addr_read(TXNCON);
405
406     set_bit(txncon, TXNCON_TXNTRIG);
407     if (ack) {
408         set_bit(txncon, TXNCON_TXNACKREQ);
409     }
410     mrf24j40_short_addr_write(TXNCON, txncon);
411
412
413 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_transmit_to_short_address (uns8 *frame_type*, uns16 *dest_pan_id*, uns16 *dest_short_address*, uns8 * *data*, uns8 *bytes_to_transmit*, uns8 *ack*)

Requests that the mrf24j40 transmit the packet to the specified short address

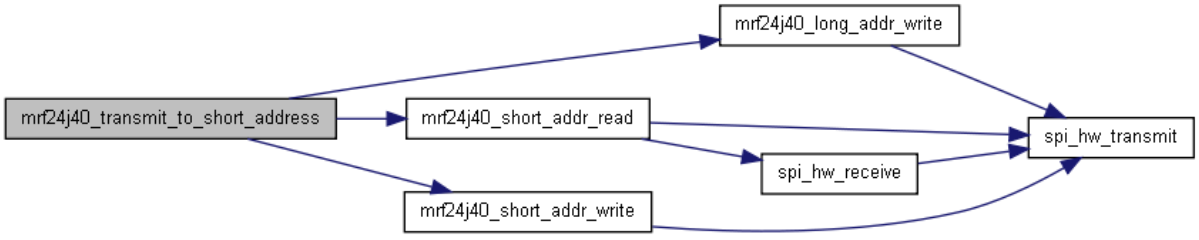
Parameters:

frame_type 802.15.4 frame type
dest_pan_id PAN id of destination
dest_short_address 16 bit short address of destination
data Pointer to uns8 array of bytes to transmit
bytes_to_transmit
ack Either MRF_ACK or MRF_NO_ACK depending if you want hardware acknowledgement

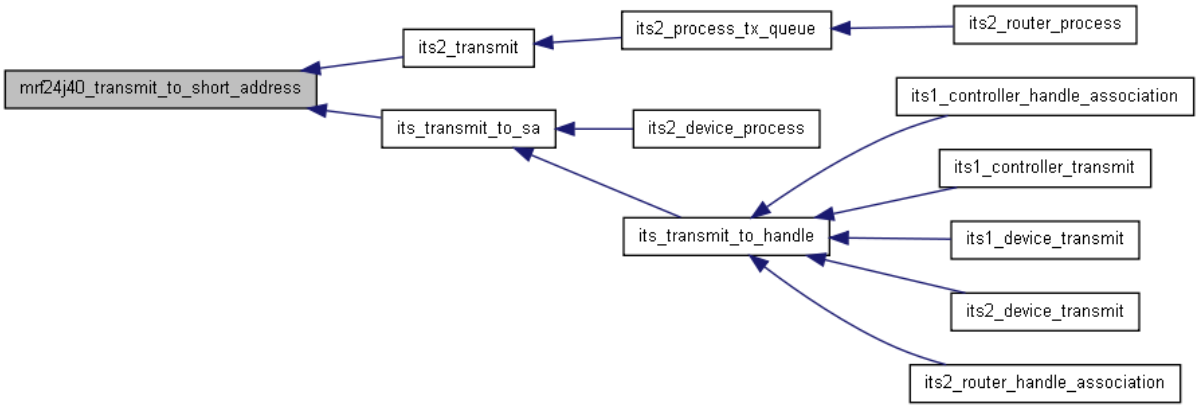
```
416
{
417
418     uns8 fc_msb = 0b10001000;   // short dest (10,11) short src (14,15)
```

```
419     uns8 fc_lsb = 0b00000000 | frame_type;  // data, pan id compression (only have dest pan
id)
420     // To do:
421     // Not smart enough for this yet:
422     //if (dest_pan_id == pan_id) {
423     //  set_bit(fc_lsb, 6);     // pan compression
424     //}
425     if (ack) {
426         set_bit(fc_lsb, 5); // ack bit
427     }
428
429     data_sequence_number++;
430
431     uns8 header_length = 3+2+2+2+2;
432     uns8 frame_length = header_length + bytes_to_transmit;
433
434     mrf24j40_long_addr_write(0x00, header_length);
435     mrf24j40_long_addr_write(0x01, frame_length);
436
437     mrf24j40_long_addr_write(0x02, fc_lsb);
438     mrf24j40_long_addr_write(0x03, fc_msb);
439     mrf24j40_long_addr_write(0x04, data_sequence_number);
440
441     mrf24j40_long_addr_write(0x05, dest_pan_id & 0xff); // dest pan id  LSB
442     mrf24j40_long_addr_write(0x06, dest_pan_id >> 8);   // MSB
443
444     mrf24j40_long_addr_write(0x07, dest_short_address & 0xff); // LSB
445     mrf24j40_long_addr_write(0x08, dest_short_address >> 8);    // MSB
446
447     mrf24j40_long_addr_write(0x09, pan_id & 0xff);  // src pan id  (=ours) LSB
448     mrf24j40_long_addr_write(0x0a, pan_id >> 8);    // MSB
449
450
451     mrf24j40_long_addr_write(0x0b, short_address & 0xff);   // LSB
452     mrf24j40_long_addr_write(0x0c, short_address >> 8);
453
454     for (uns8 count=0;  count < bytes_to_transmit; count++) { // check format here
455         mrf24j40_long_addr_write(count+header_length+2,  data[count]);
456     }
457
458     uns8 txncon = mrf24j40_short_addr_read(TXNCON);
459     set_bit(txncon, TXNCON_TXNTRIG);
460     if (ack) {
461         set bit(txncon, TXNCON_TXNACKREQ);
462     }
463     mrf24j40_short_addr_write(TXNCON, txncon);
464
465
466 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



Variable Documentation

uns8 current_channel = 0

uns8 data_sequence_number

uns8 extended_address[8] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff}

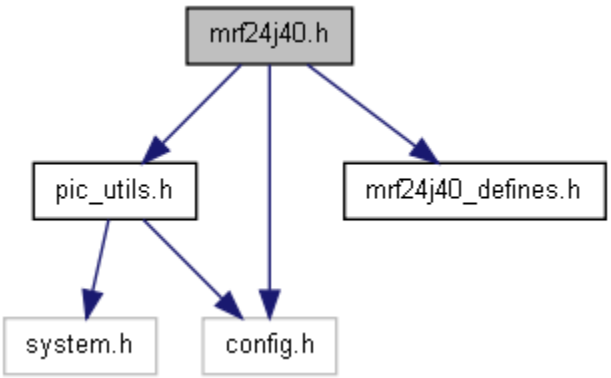
uns16 pan_id = 0xffff

uns16 short_address = 0xffff

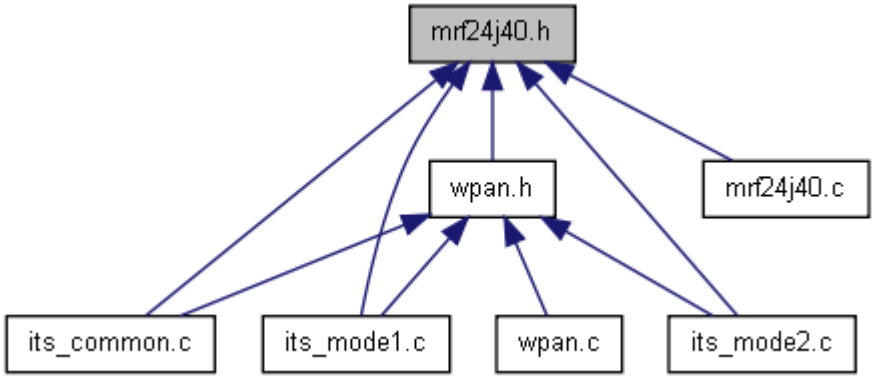
---

mrf24j40.h File Reference

Microchip mrf24j40 IEEE 802.15.4 module routines.
Include dependency graph for mrf24j40.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define LOC_CANADA 0x02
- #define LOC_EUROPE 0x03
- #define LOC_UNDEFINED 0x00
- #define LOC_UNITED_STATES 0x01
- #define MRF_ACK 1
- #define MRF_FIRST_CHANNEL 11
- #define MRF_LAST_CHANNEL 26
- #define MRF_NO_ACK 0

Functions

- void mrf24j40_flush_receive_buffer ()
- *Flush receive buffer of mrf24j40.* void mrf24j40_handle_isr ()
- *Interrupt service routine for mrf24j40 chip.* void mrf24j40_init ()
- *Initialises mrf24j40 chip ready for use.* uns8 mrf24j40_long_addr_read (uns16 addr)
- *Read data from long address of memory location in mrf24j40.* void mrf24j40_long_addr_write (uns16 addr, uns8 data)
- *Write data to long address memory location.* uns8 mrf24j40_receive (uns8 *data, uns8 bytes_to_receive)
- *Pull received data from buffer.* void mrf24j40_receive_callback ()
- *Callback is actioned when mrf24j40 has a packet received off air.* uns8 mrf24j40_scan_for_lowest_channel_ed ()
- *Scan all channels for lowest RF energy.* void mrf24j40_set_channel (uns8 channel)
- *Change channels.* void mrf24j40_set_extended_address (uns8 *_extended_address)
- *Set IEEE 802.15.4 extended address.* void mrf24j40_set_pan_id (uns16 _pan_id)
- *Set PAN id.* void mrf24j40_set_short_address (uns16 _short_address)
- *Set short address.* void mrf24j40_setup_io ()
- *Setup ports/pins as inputs/outputs ready for use.* uns8 mrf24j40_short_addr_read (uns8 addr)
- *Read data from short address of memory location in mrf24j40.* void mrf24j40_short_addr_write (uns8 addr, uns8 data)
- *Write data to short address memory location.* void mrf24j40_transmit (uns8 *data, uns8 bytes_to_transmit)
- *Transmit raw data.* void mrf24j40_transmit_callback (uns8 status, uns8 retries, uns8 channel_busy)
- *Callback is actioned when mrf24j40 has finished transmitting a packet, or failed to do so.* void mrf24j40_transmit_to_extended_address (uns8 frame_type, uns16 dest_pan_id, uns8 *dest_extended_address, uns8 *data, uns8 data_length, uns8 ack)
- *Transmit packet to extended address.* void mrf24j40_transmit_to_short_address (uns8 frame_type, uns16 dest_pan_id, uns16 dest_short_address, uns8 *data, uns8 bytes_to_transmit, uns8 ack)

Transmit packet to short address.

Detailed Description

Define Documentation

#define LOC_CANADA 0x02

Module located in Canada (?? power)

#define LOC_EUROPE 0x03

Module located in Europe (-14.9dB power)

#define LOC_UNDEFINED 0x00

Module located in undefined location (full power)

#define LOC_UNITED_STATES 0x01

Module located in United States (?? power)

#define MRF_ACK 1

Send mrf packet and request acknowledgement

#define MRF_FIRST_CHANNEL 11

First available mrf channel

#define MRF_LAST_CHANNEL 26

Last availablre mrf channel

#define MRF_NO_ACK 0

Send mrf packet without acknowledgement

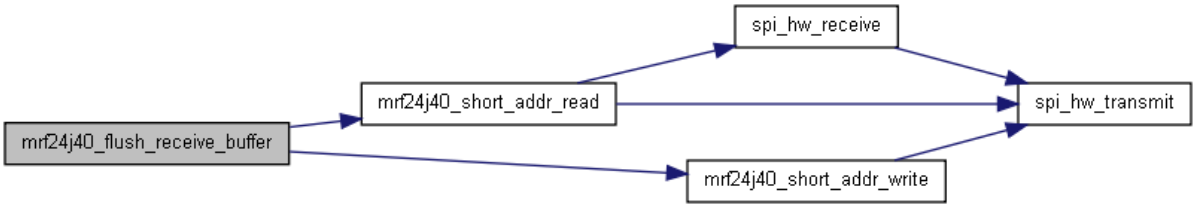
Function Documentation

void mrf24j40_flush_receive_buffer ()

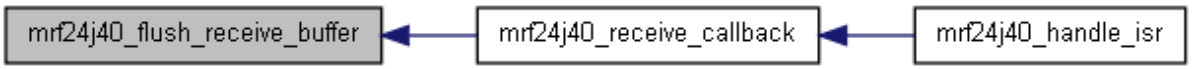
No need to call this routine normally, except according to mrf24j40 errata (promiscuous mode)

```
52                                      {
53
54     uns8 rxflush;
55
56     rxflush = mrf24j40_short_addr_read(RXFLUSH);    // Get rxflush
57     set_bit(rxflush, RXFLUSH_RXFLUSH);  // Set flush bit
58     mrf24j40_short_addr_write(RXFLUSH, rxflush);    // Push it back to the mrf
59 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

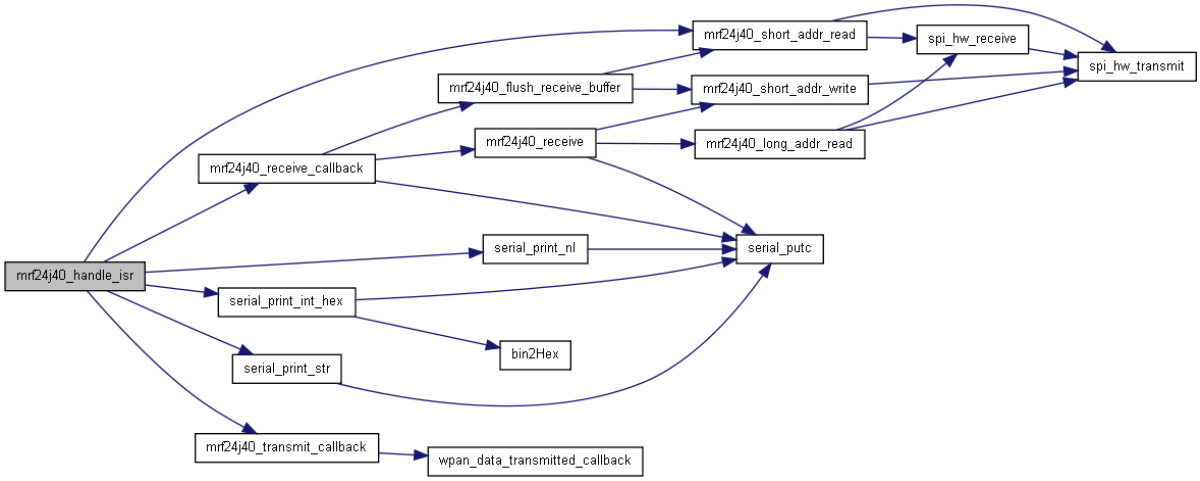


void mrf24j40_handle_isr ()

Call this routine when the mrf24j40 indicates an interrupt condition, or called regularly.

```
287                             {
288
289 uns8 intstat;
290
291     // first we need to get a copy of intstat to find out what happened
292     intstat = mrf24j40_short_addr_read(INTSTAT);
293
294     // mrf24j40 intstat register is cleared on read
295     if (test_bit(intstat, INTSTAT_RXIF)) {
296         serial_print_str("R");
297         // handle receive
298         mrf24j40_receive_callback();
299     }
300     if (test_bit(intstat, INTSTAT_TXNIF)) {
301         serial_print_str("Tx complete txstat=0x");
302         uns8 stat = mrf24j40_short_addr_read(TXSTAT);
303         mrf24j40_transmit_callback(stat & 0b00000001,   // success = 0
304                                    stat >> 6,    // retries
305                                    test_bit(stat, TXSTAT_CCAFAIL)); // due to cca
failure?
306         serial_print_int_hex(stat);
307         serial_print_nl();
308     }
309
310 }
```

Here is the call graph for this function:



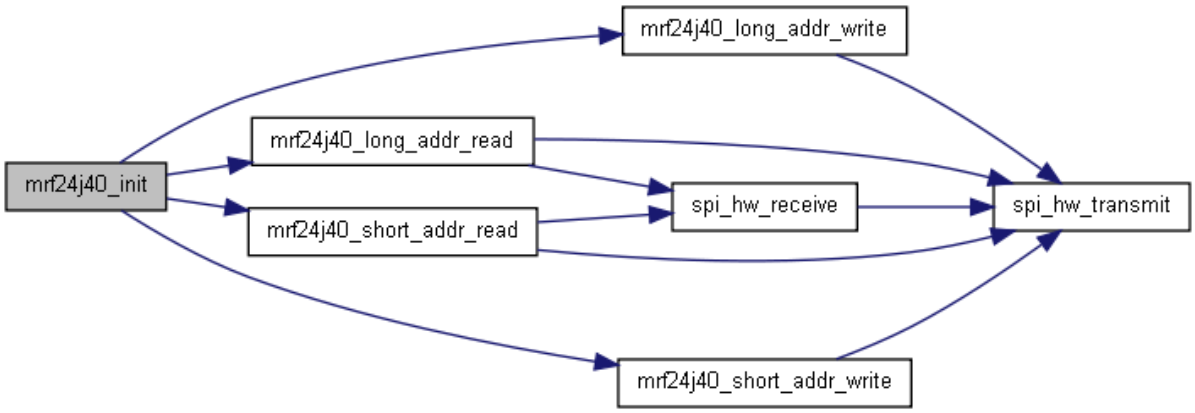
void mrf24j40_init ()

Sets int pin as input and cs pin as output.

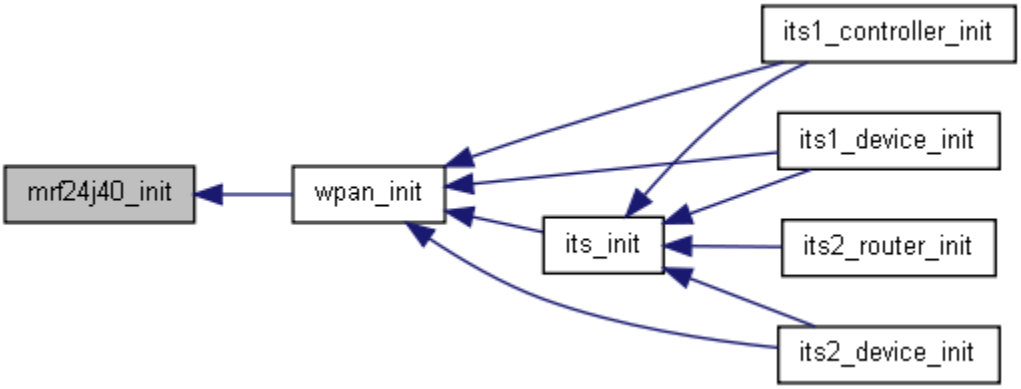
```
499                     {
500
501 uns8 check;
502
503     // Example steps to initialize the MRF24J40:
504
505     // 1. SOFTRST (0x2A) = 0x07 - Perform a software Reset. The bits will be automatically
cleared to '0' by hardware.
506     mrf24j40 short addr write(SOFTRST, 0x07);
507
508     // Wait for soft reset to complete
509
510     do {
511         check = mrf24j40 short addr read(SOFTRST);
512     } while (check != 0);
513
514     #if defined(ENABLE_PA_LNA) && defined(MRF24J40MB)
515         // Turn on PA/LNA if we have one
516         mrf24j40 long addr write(TESTMODE, 0x0f);
517     #endif
518
519     // 2. PACON2 (0x18) = 0x98 - Initialize FIFOEN = 1 and TXONTS = 0x6.
520     mrf24j40 short addr write(PACON2, 0x98);
521     // 3. TXSTBL (0x2E) = 0x95 - Initialize RFSTBL = 0x9.
522     mrf24j40 short addr write(TXSTBL, 0x95);
523
524     // wait for mrf to be in receive mode
525
526     do {
527         check = mrf24j40 long addr read(RFSTATE);
528     } while (check & 0xa0 != 0xa0);
529
530     mrf24j40 long addr write(RFCON0, 0x03); // or three?
531     // mrf24j40_long_addr_write(RFCON1, 0x02);  // VCO control mode (1 or 2?)
532     // miwi stack source is at odds with data sheet here
533
534     // 4. RFCON1 (0x201) = 0x01 - Initialize VCOOPT = 0x01.
535     mrf24j40 long addr write(RFCON1, 0x01);
536     // 5. RFCON2 (0x202) = 0x80 - Enable PLL (PLLEN = 1).
537     mrf24j40 long addr write(RFCON2, 0x80);
538     // 6. RFCON6 (0x206) = 0x90 - Initialize TXFIL = 1 and 20MRECVR = 1.
539     mrf24j40 long addr write(RFCON6, 0x90);
540     // 7. RFCON7 (0x207) = 0x80 - Initialize SLPCLKSEL = 0x2 (100 kHz Internal oscillator).
541     mrf24j40 long addr write(RFCON7, 0x08);
542     // 8. RFCON8 (0x208) = 0x10 - Initialize RFVCO = 1.
543     mrf24j40 long addr write(RFCON8, 0x10);
544     // 9. SLPCON1 (0x220) = 0x21 - Initialize CLKOUTEN = 1 and SLPCLKDIV = 0x01.
545     mrf24j40 long addr write(SLPCON1, 0x21);
546
547     //Configuration for nonbeacon-enabled devices
548     //(see Section MRF datasheet 3.8 "Beacon-Enabled and Nonbeacon-Enabled Networks"):
549     //10. BBREG2 (0x3A) = 0x80 - Set CCA mode to ED.
550     mrf24j40 short addr write(BBREG2, 0x80);
551     // 11. RSSITHCCA (0x3F) = 0x60 - Set CCA ED threshold.
552     // IH: I think the datasheet is wrong, the mem address is "CCAEDTH", even though
553     // it's referred to as RSSITHCCA and CCAMODE in the documentation...
554
555     mrf24j40 short addr write(CCAEDTH, 0x60);
556     // 12. BBREG6 (0x3E) = 0x40 - Set appended RSSI value to RXFIFO.
557     mrf24j40 short addr write(BBREG6, 1 << BBREG6 RSSIMODE2);
558     // 13. Enable interrupts - See Section 3.3 "Interrupts".
559     mrf24j40 short addr write(MRF INTCON, (1 << MRF INTCON RXIE) & (1 << MRF INTCON TXNIE));
560     // 14. Set channel - See Section 3.4 "Channel Selection".
561     mrf24j40 long addr write(RFCON0, 11);
562
563     #if defined(ENABLE_PA_LNA) && defined(MRF24J40MB)
564     // There are special MRF24J40 transceiver output power
565     // setting for Microchip MRF24J40MB module.
```

```
566     // To do: Correct settings for other locations
567
568         #if APPLICATION SITE == EUROPE
569             // MRF24J40 output power set to be -14.9dB
570             mrf24j40 long addr write(RFCON3, 0x70);
571         #else
572             // MRF24J40 output power set to be -1.9dB - Stock setting
573             // mrf24j40_long_addr_writeRFCON3, 0x18);
574
575             // MRF24J40 output power set to be -1.2dB
576             // mrf24j40 long addr write(RFCON3, 0x10);
577
578             // MRF24J40 output power set to be -0.5dB
579             // mrf24j40_long_addr_write(RFCON3, 0x08);
580
581             // MRF24J40 output power set to be -0dB -> 22,5 dBm output power
582             mrf24j40 long addr write(RFCON3, 0x00);
583         #endif
584     #else
585         // power level to be 0dBms
586         mrf24j40 long addr write(RFCON3,0x00);
587     #endif
588
589     // 15. RFCTL (0x36) = 0x04 - Reset RF state machine.
590     mrf24j40 short addr write(RFCTL, 0x04);
591
592     // 16. RFCTL (0x36) = 0x00.
593     mrf24j40 short addr write(RFCTL, 0x00);
594     // 17. Delay at least 192 ìs.
595     delay_us(200);
596
597     //  RXMCR.PANCOORD = 1 if coordinator, 0 is default (not coordinator)
598
599     //mrf24j40_short_addr_write(RXMCR, 0x01); // promiscuous mode, no coord
600
601     data sequence number = 0;
602
603 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



uns8 mrf24j40_long_addr_read (uns16 *addr*)

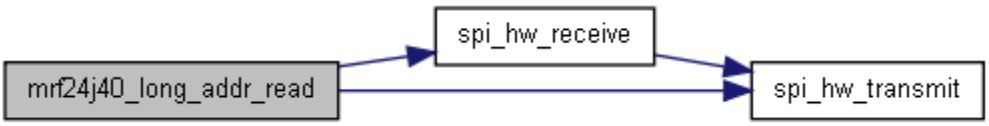
Returns the value in the memory location specified.

Parameters:

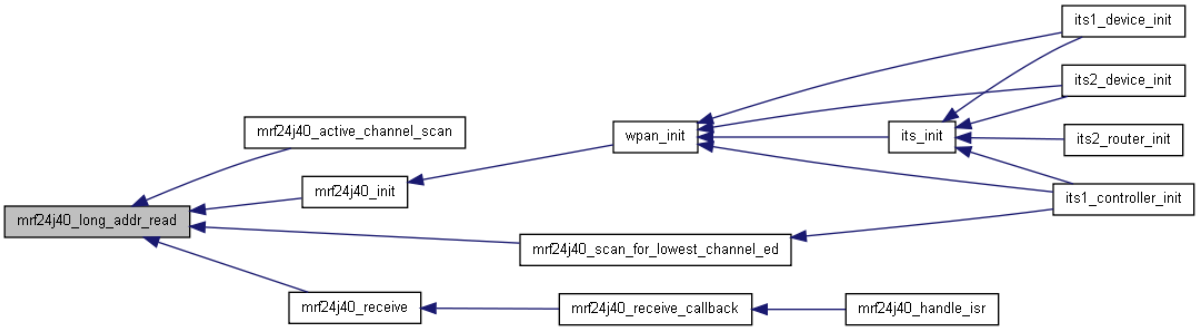
addr Long address memory location (see mrf24h40_defines.h)

```
749                                         {
750
751     uns8 result;
752
753     clear_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
754
755     addr = addr & 0b0000001111111111;   // <9:0> bits
756     addr = addr << 5;
757     set_bit(addr, 15);  // long addresss
758     spi_hw_transmit(addr >> 8);
759     spi_hw_transmit(addr & 0x00ff);
760     result = spi_hw_receive();
761
762     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
763
764     return result;
765 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_long_addr_write (uns16 *addr*, uns8 *data*)

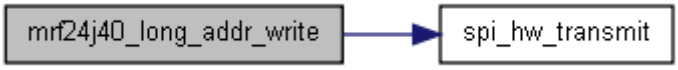
Sets the memory location to the value specified

Parameters:

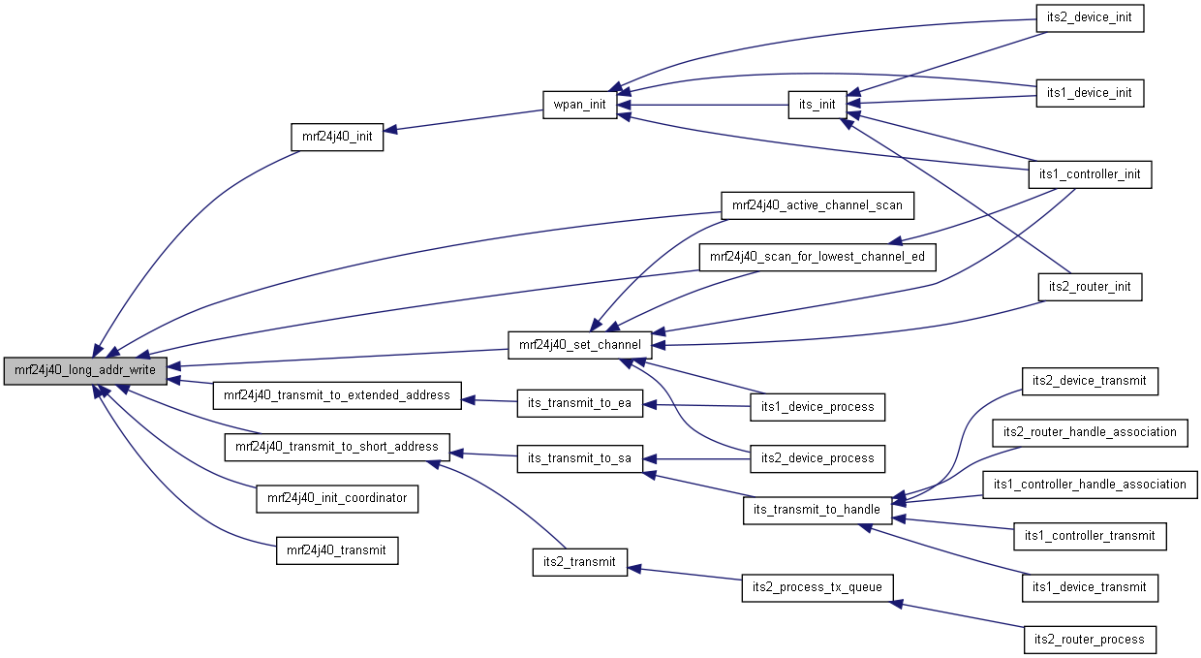
addr Long address memory location (see mrf24h40_defines.h)
data Value that the memory location should be set to

```
767                                                     {
768
769     clear_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
770
771     addr = addr & 0b0000001111111111;   // <9:0> bits
772     addr = addr << 5;
773
774     set_bit(addr, 15);  // long addresss
775     set_bit(addr, 4);   // set for write
776
777     spi_hw_transmit(addr >> 8 );
778     spi_hw_transmit(addr & 0x00ff);
779     spi_hw_transmit(data);
780
781     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
782 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



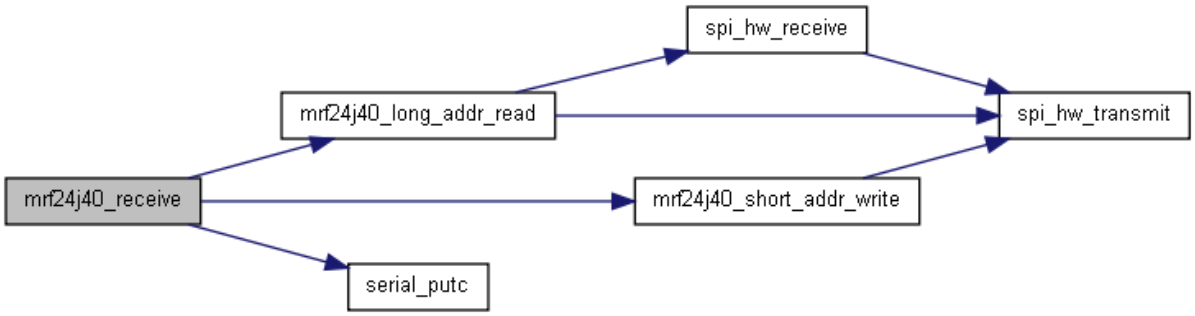
uns8 mrf24j40_receive (uns8 * *data*, uns8 *bytes_to_receive*)

Once an interrupt has occured and we know it is because a packet has been received, this routine will pull the data from the mrf receive buffer. This needs to be done quickly so that the next packet is not lost.

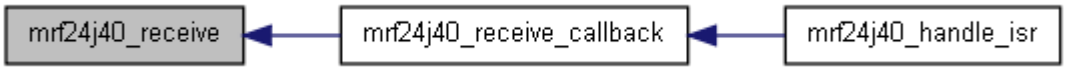
```
312                                                     {
313
```

```
314 uns8 frame_length;
315 uns16 frame_pos;
316 uns8 buffer count;
317 /*
318 1. Receive RXIF interrupt.
319 2. Disable host microcontroller interrupts.
320 3. Set RXDECINV = 1; disable receiving packets off air.
321 4. Read address, 0x300; get RXFIFO frame length value.
322 5. Read RXFIFO addresses, 0x301 through (0x300 + Frame Length + 2); read packet data plus LQI
and RSSI.
323 6. Clear RXDECINV = 0; enable receiving packets.
324 7. Enable host microcontroller interrupts.
325 */
326     serial_putc('a');
327     // Disable reading packets off air
328     mrf24j40_short_addr_write(BBREG1, 1 << BBREG1_RXDECINV);
329     serial_putc('b');
330
331     frame_pos = 0x300;
332     frame_length = mrf24j40_long_addr_read(frame_pos++);
333     buffer_count = 0;
334     serial_putc('c');
335
336     while ((buffer_count <= bytes_to_receive) && (buffer_count <= frame_length + 2)) {
337         //0x301 through (0x300 + Frame Length + 2 ); read packet data plus LQI and RSSI.
338         data[buffer_count++] = mrf24j40_long_addr_read(frame_pos++);
339     }
340     serial_putc('d');
341
342     // Re-enable reading packets off air
343     mrf24j40_short_addr_write(BBREG1, 0);
344     serial_putc('e');
345
346     return buffer_count - 1;
347
348 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



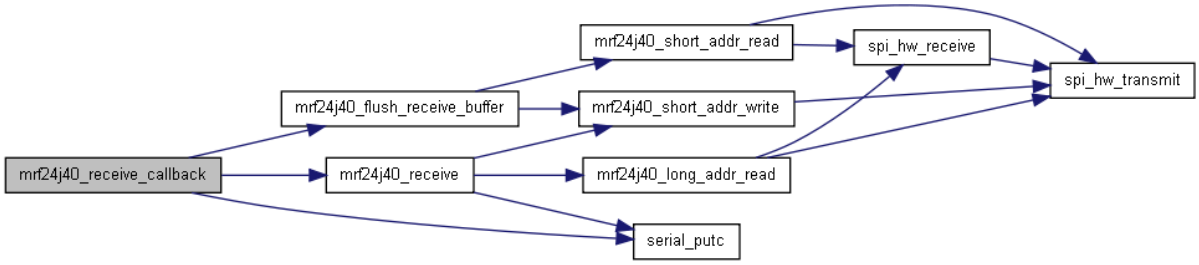
void mrf24j40_receive_callback ()

Callback indicating the mrf24j40 has a packet ready.

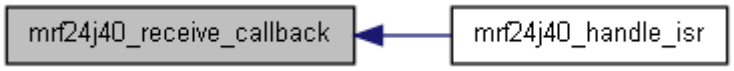
```
56                                  {
57     serial_putc('r');
58     if (pkt_received) {
59         receive_lost++;
60         debug_str("<Lost receive>");
61         mrf24j40_flush_receive_buffer();
```

```
62      } else {
63          pkt_received++;
64          wpan_rx_count = mrf24j40_receive(&wpan_rx_buffer, sizeof(wpan_rx_buffer));
65
66          debug_str("Rx: ");
67          debug_int(wpan_rx_count);
68          debug_str(" bytes ");
69      }
70
71      // this will get picked up in wpan_process
72
73  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



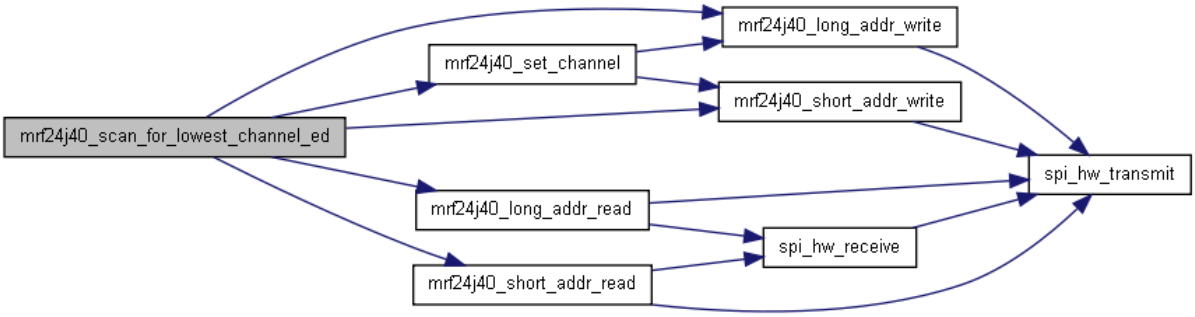
uns8 mrf24j40_scan_for_lowest_channel_ed ()

Scans through all channels and reports the channel with the lowest Energy Detection level.

```
77                                          {
78
79  uns8 rssi;
80  uns8 channel;
81  uns16 scan_count;
82  uns8 highest_on_channel;
83  uns8 lowest_channel = MRF_LAST_CHANNEL;
84  uns8 lowest_ed = 0xff;
85  uns8 bbreg6;
86
87      // We should really ignore all packets here (do to)
88
89      #if defined(ENABLE_PA_LNA)
90          mrf24j40_long_addr_write(TESTMODE, 0x08);              // Disable automatic switch on
PA/LNA
91          mrf24j40_short_addr_write(TRISGPIO, 0x0F);         // Set GPIO direction
92          mrf24j40_short_addr_write(GPIO, 0x0C);            // Enable LNA
93      #endif
94
95
96      for (channel = MRF_FIRST_CHANNEL; channel <= MRF_LAST_CHANNEL; channel++) {
97
98          // switch channel
99          mrf24j40_set_channel(channel);
100
101          highest_on_channel = 0;
102          for (scan_count = 0; scan_count < 1000; scan_count++) {
103              mrf24j40_short_addr_write(BBREG6, 1 << BBREG6_RSSIMODE1);
104              do {
105                  bbreg6 = mrf24j40_short_addr_read(BBREG6);
106              } while (!test_bit(bbreg6, BBREG6_RSSIRDY));
107              rssi = mrf24j40_long_addr_read(RSSI);
```

```
108             if (rssi > highest_on_channel) {
109                 highest_on_channel = rssi;
110             }
111         }
112 
113         if (highest_on_channel < lowest_ed) {
114             lowest_ed = highest_on_channel;
115             lowest_channel = channel;
116         }
117     }
118     mrf24j40_short_addr_write(BBREG6, 1 << BBREG6_RSSIMODE2);   // Back to mode 2 (rssi in
pkt)
119 
120     // Should stop ignoring all packets now if we started ignoring them earlier
121     // (to do)
122 
123     #if defined(ENABLE_PA_LNA)
124         mrf24j40_short_addr_write(GPIO, 0);
125         mrf24j40_short_addr_write(TRISGPIO, 0x00);
126         mrf24j40_long_addr_write(TESTMODE, 0x0F);
127     #endif
128 
129 
130     return lowest_channel;
131 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void mrf24j40_set_channel (uns8 *channel*)

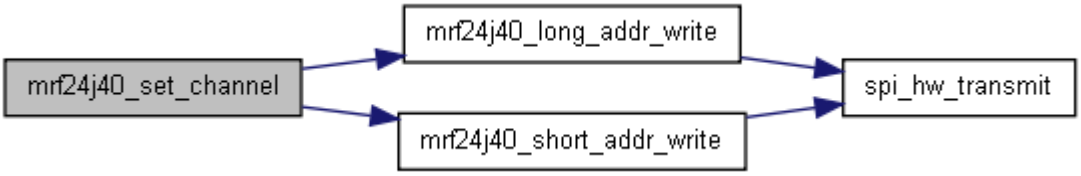
Change to the specified channel, in the range MRF_FIRST_CHANNEL to MRF_LAST_CHANNEL.

Parameters:

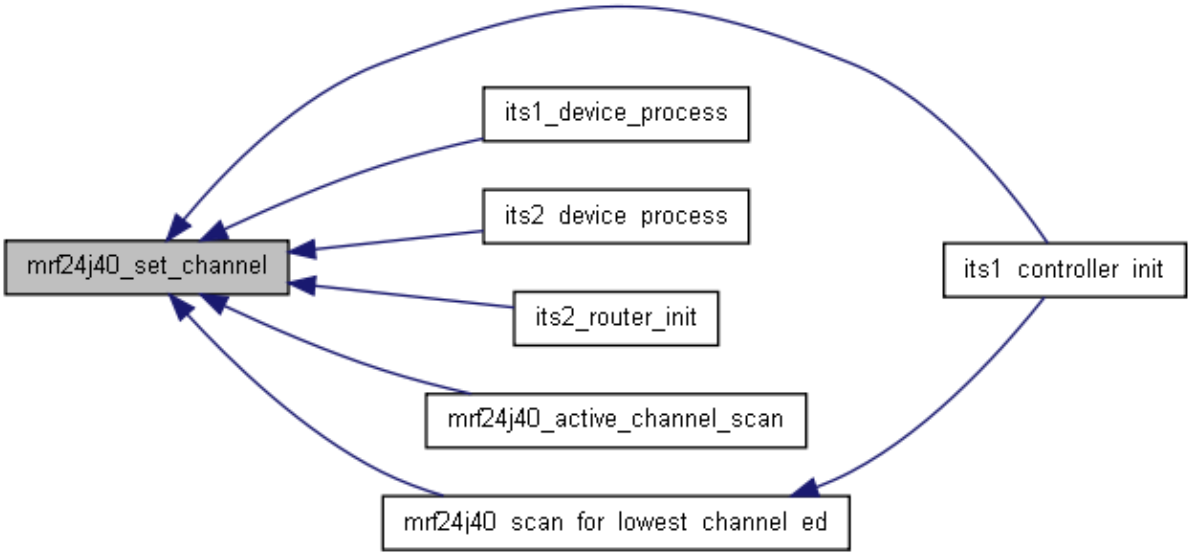
channel Channel to change to.

```
63                                      {
64 
65     current_channel = channel;
66     channel = channel - 11;
67     channel = 0x02 + 0x10 * channel;
68 
69     mrf24j40_long_addr_write(RFCON0, channel);  // Set channel
70     mrf24j40_short_addr_write(RFCTL, 0x04); // RFCTL (0x36) = 0x04 - Reset RF state machine.
71     mrf24j40_short_addr_write(RFCTL, 0x00); // RFCTL (0x36) = 0x00
72 
73     delay_us(200);  // Delay at least 192us
74 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_set_extended_address (uns8 * *_extended_address*)

Set extended address.

Pass a pointer to an 8 byte uns8 array with the address embedded as MSB leftmost byte and LSB rightmost byte, eg

uns8 EA_3[8] = { 0, 0, 0, 0, 0, 0, 0, 3 };

Set extended address to 0x0000000000000003 (64 bit address) mrf24j40_set_extended_address(&EA_3);

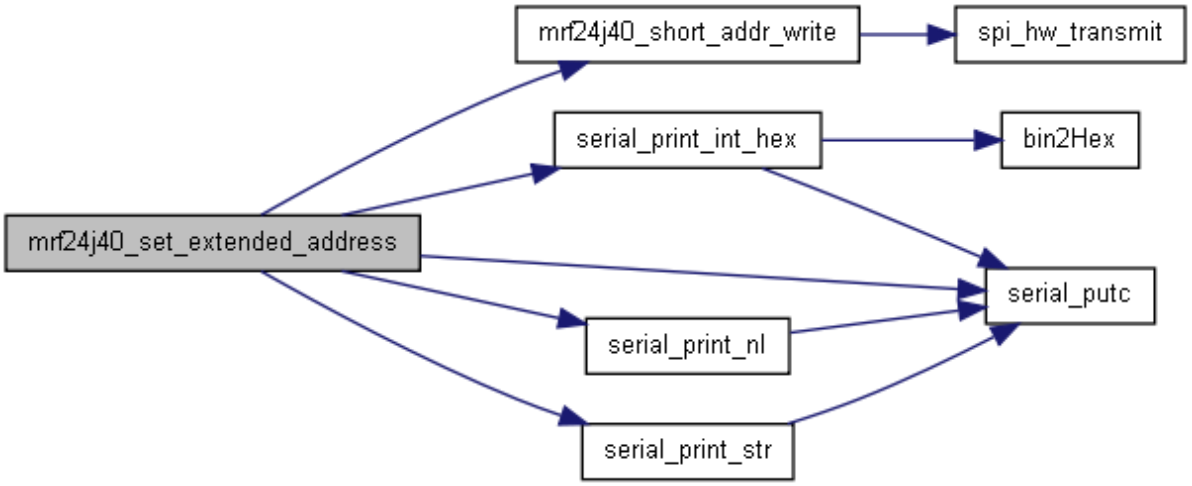
Sets the 64 bit extended address of the module. Pass a pointer to an 8 byte uns8 array containing the address.

Set IEEE 802.15.4 extended address.

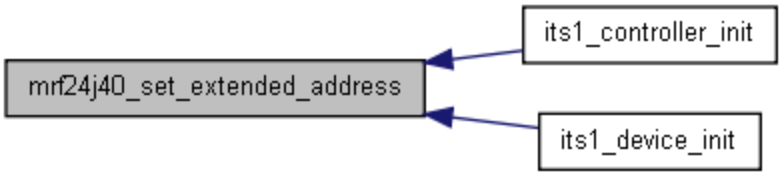
Sets the 64 bit extended address of the module. Pass a pointer to an 8 byte uns8 array containing the address.

```
259                                                     {
260     uns8 count;
261     uns8 mem_pos = EADR7;
262     serial_print_str("Setting EA to: ");
263     for (count = 0; count < 8; count++) {
264         extended_address[count] =  extended_address[count];
265         mrf24j40_short_addr_write(mem_pos--, extended_address[count]);
266         serial_print_int_hex(extended_address[count]);
267         serial_putc(' ');
268 
269     }
270     serial_print_nl();
271 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

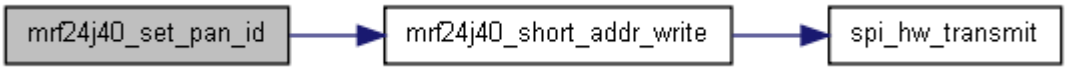


void mrf24j40_set_pan_id (uns16 *_pan_id*)

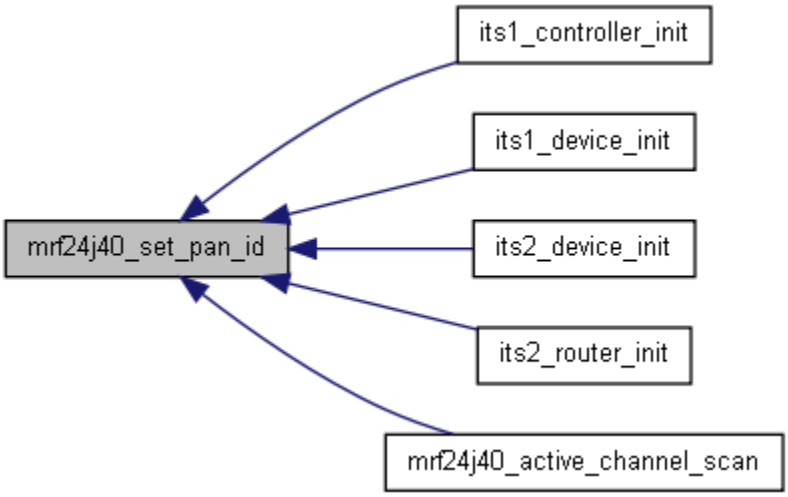
Sets the 16 bit PAN id of the module.

```
252                                         {
253     pan_id = _pan_id;
254     mrf24j40_short_addr_write(PANIDL, pan_id & 0xff);
255     mrf24j40_short_addr_write(PANIDH, pan_id >> 8);
256 
257 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

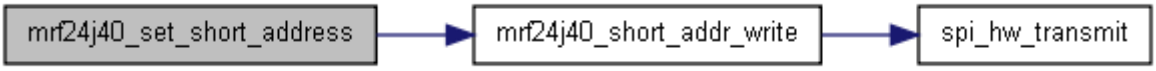


void mrf24j40_set_short_address (uns16 *_short_address*)

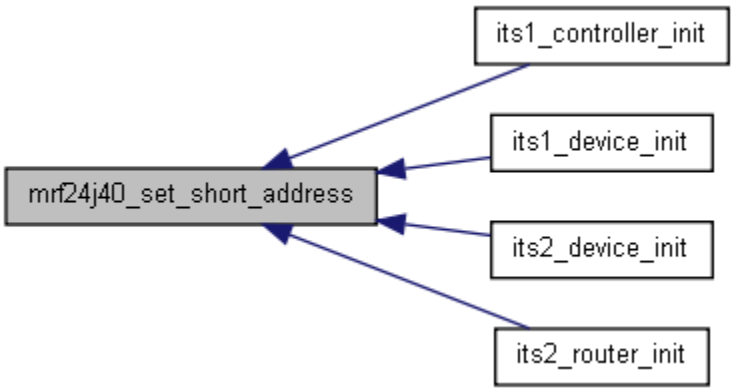
Sets the 16 bit short address of the module.

```
273                                                 {
274
275     // Keep hold of the short address
276     short_address = _short_address;
277
278     // Tell the mrf about it
279     mrf24j40_short_addr_write(SADRL, short_address & 0xff);
280     mrf24j40_short_addr_write(SADRH, short_address >> 8);
281
282 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



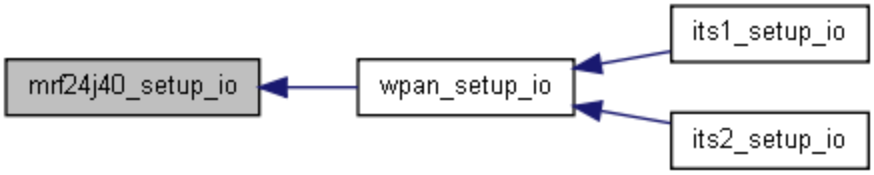
void mrf24j40_setup_io ()

Sets int pin as input and cs pin as output.

```
785                         {
786
```

```
787     make_input(mrf24j40_int_port, mrf24j40_int_pin);
788     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin); // keep high
789     make_output(mrf24j40_cs_port, mrf24j40_cs_pin);
790
791 }
```

Here is the caller graph for this function:



uns8 mrf24j40_short_addr_read (uns8 *addr*)

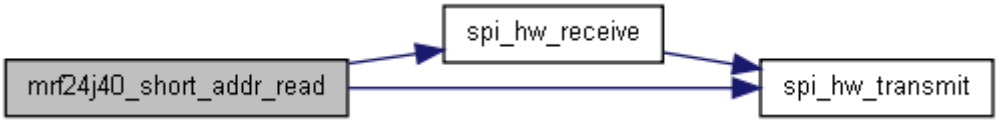
Returns the value in the memory location specified.

Parameters:

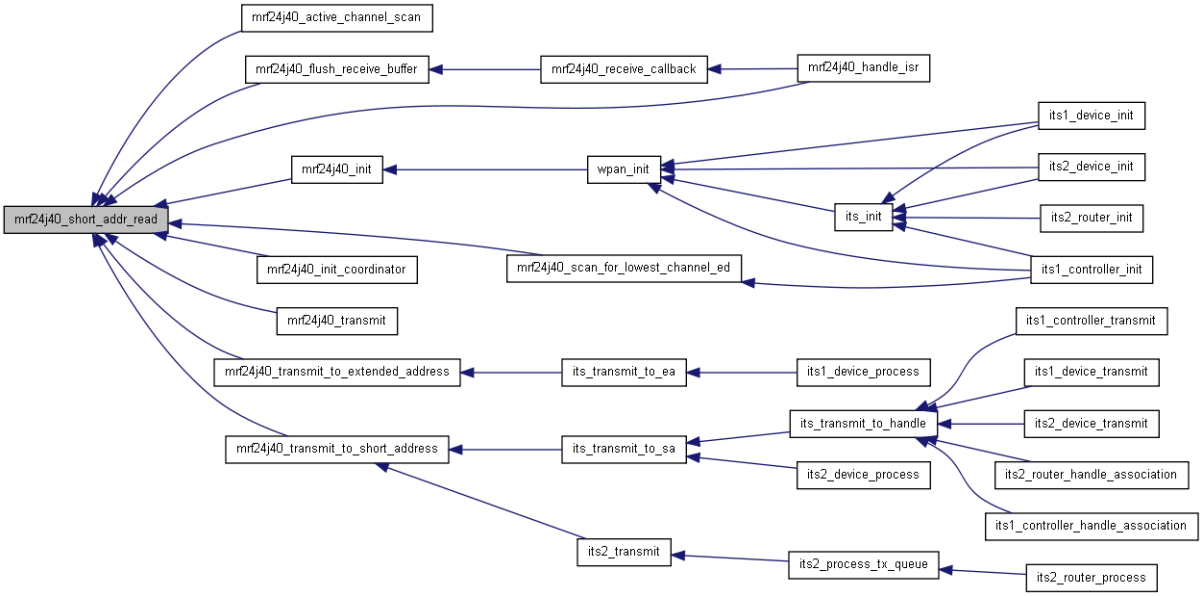
addr Short address memory location (see mrf24h40_defines.h)

```
725                                         {
726
727     clear_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
728     addr = addr & 0b00111111;   // <5:0> bits
729     addr = addr << 1;   // LSB = 0, means read
730     spi_hw_transmit(addr);
731     uns8 result = spi_hw_receive();
732
733     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
734     return result;
735 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_short_addr_write (uns8 *addr*, uns8 *data*)

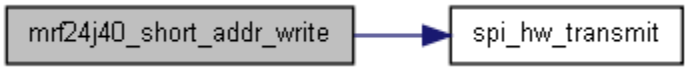
Sets the memory location to the value specified

Parameters:

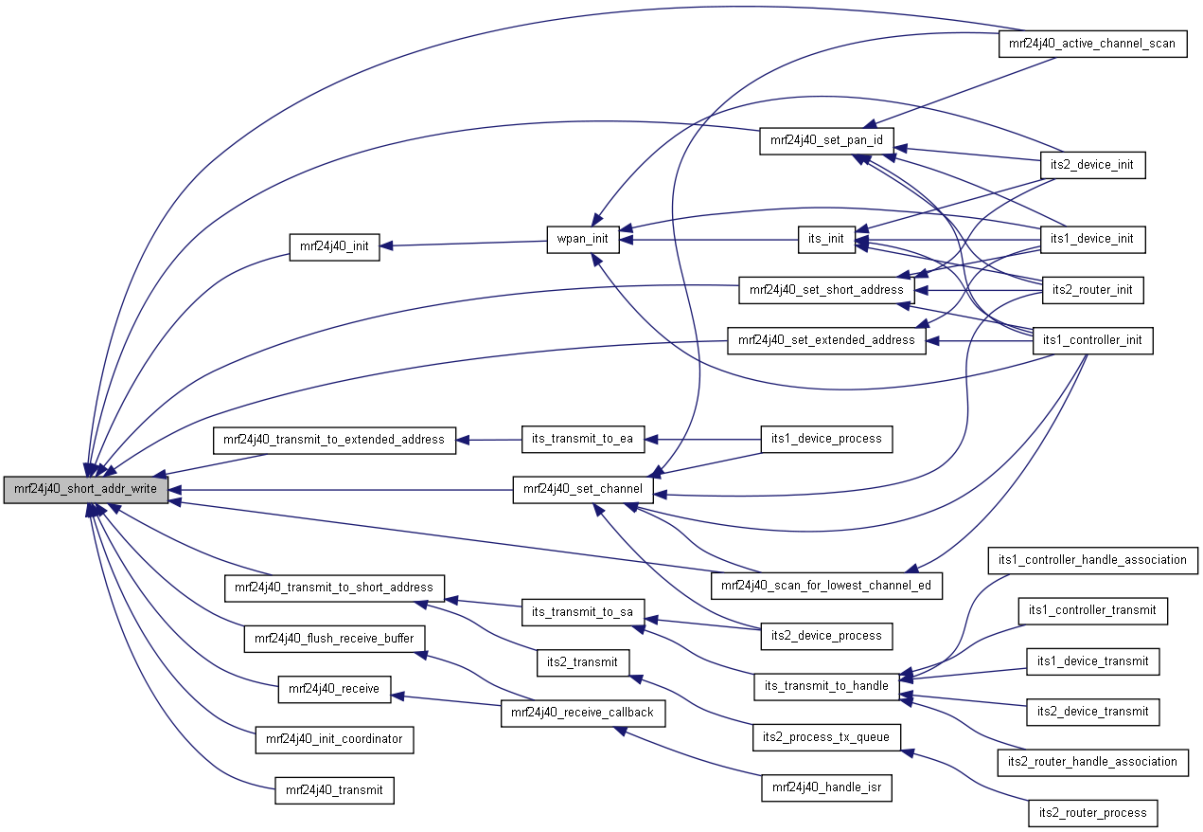
addr Short address memory location (see mrf24h40_defines.h)
data Value that the memory location should be set to

```
737                                                     {
738     clear_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
739     addr = addr & 0b00111111;   // <5:0> bits
740     addr = addr << 1;   // LSB = 0, means read
741     set_bit(addr, 0);   // write
742
743     spi_hw_transmit(addr);
744     spi_hw_transmit(data);
745
746     set_pin(mrf24j40_cs_port, mrf24j40_cs_pin);
747 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

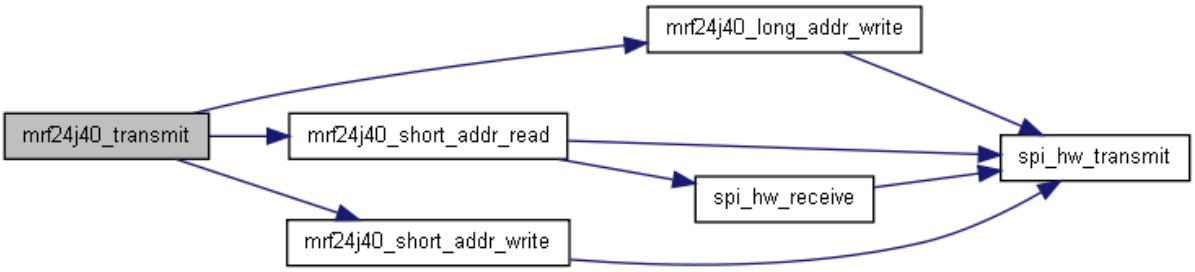


void mrf24j40_transmit (uns8 * *data*, uns8 *bytes_to_transmit*)

Transmit a raw packet - this assumes you have already created an 802.15.4 compatible packet. Normally you would use transmit_to_extended_adddress or transmit_to_short_address instead.

```
470                                                     {
471
472
473     uns8 fc_lsb = 0b01000001;
474     uns8 fc_msb = 0b00000000;
475
476
477     data_sequence_number++;
478     uns8 header_length = 3; // Just two bytes of frame control + sequence number, no addrs
479     uns8 frame_length = header_length + bytes_to_transmit;
480
481     mrf24j40_long_addr_write(0x00, header_length);
482     mrf24j40_long_addr_write(0x01, frame_length);
483     mrf24j40_long_addr_write(0x02, fc_lsb);
484     mrf24j40_long_addr_write(0x03, fc_msb);
485     mrf24j40_long_addr_write(0x04, data_sequence_number);
486
487     for (uns8 count=0;  count < bytes_to_transmit; count++) { // check format here
488         mrf24j40_long_addr_write(count+header_length+2,  data[count]);
489     }
490
491     uns8 txncon = mrf24j40_short_addr_read(TXNCON);
492     set_bit(txncon, TXNCON_TXNTRIG);
493     mrf24j40_short_addr_write(TXNCON, txncon);
494
495
496 }
```

Here is the call graph for this function:



void mrf24j40_transmit_callback (uns8 *status*, uns8 *retries*, uns8 *channel_busy*)

Callback indicating the mrf24j40 has finished the transmission sequence.

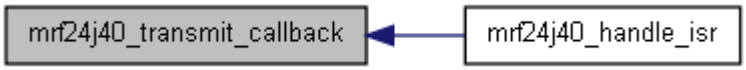
Parameters:

status Set to 0 for success or 1 for failure
retries Set to the number of retries
channel_busy Set to 1 if failure was due to the channel being busy.

```
77                                                  { // 1 if fail due to channel busy
78     debug_str(" <Tx: ");
79     if (!status) {
80         debug_str(" Good");
81     } else {
82         debug_str(" Fail");
83         if (channel_busy) {
84             debug_str(" (channel busy)");
85         }
86     }
87     debug_str(" Retries: ");
88     debug_int(retries);
89     debug_str(">\n");
90 
91     wpan_data_transmitted_callback(status, retries, channel_busy);
92 
93 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void mrf24j40_transmit_to_extended_address (uns8 *frame_type*, uns16 *dest_pan_id*, uns8 * *dest_extended_address*, uns8 * *data*, uns8 *data_length*, uns8 *ack*)

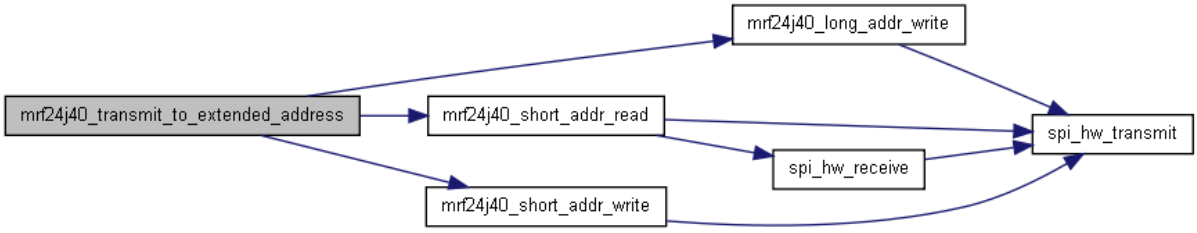
Requests that the mrf24j40 transmit the packet to the specified extended address

Parameters:

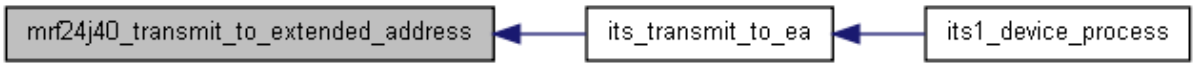
frame_type 802.15.4 frame type
dest_pan_id PAN id of destination
dest_extended_address Pointer to uns8 array indicating extended address of destination
data Pointer to uns8 array of bytes to transmit
bytes_to_transmit
ack Either MRF_ACK or MRF_NO_ACK depending if you want hardware acknowledgement

```
351                                                                                                 {
352
353     // See notes below on frame control bytes format:
354     uns8 fc_msb = 0b11001100;   // 64 bit dest (10,11) 64 bit src (14,15)
355     uns8 fc_lsb = 0b00000000 | frame_type;  // pan id compression=0, data
356
357     if (ack) {
358         set_bit(fc_lsb, 5); // ack bit
359     }
360
361     data sequence number++;
362
363     uns8 header_length = 3+8+8+2+2;
364     uns8 frame_length = header_length + data_length;
365
366     // Write out data to mrf
367
368     mrf24j40 long addr write(0x00, header_length);
369     mrf24j40 long addr write(0x01, frame_length);
370
371     mrf24j40 long addr write(0x02, fc_lsb);
372     mrf24j40 long addr write(0x03, fc_msb);
373     mrf24j40 long addr write(0x04, data sequence number);
374
375     mrf24j40 long addr write(0x05, dest_pan_id & 0xff); // dest pan id LSB
376     mrf24j40 long addr write(0x06, dest_pan_id >> 8);   // MSB
377
378     mrf24j40 long addr write(0x07, dest extended address[7]); // LSB
379     mrf24j40 long addr write(0x08, dest_extended_address[6]);
380     mrf24j40 long addr write(0x09, dest_extended_address[5]);
381     mrf24j40 long addr write(0x0a, dest_extended_address[4]);
382     mrf24j40 long addr write(0x0b, dest extended address[3]);
383     mrf24j40 long addr write(0x0c, dest_extended_address[2]);
384     mrf24j40 long addr write(0x0d, dest_extended_address[1]);
385     mrf24j40 long addr write(0x0e, dest_extended_address[0]); // MSB
386
387     mrf24j40 long addr write(0x0f, pan id & 0xff);  // src pan id LSB
388     mrf24j40 long addr write(0x10, pan id >> 8);    // MSB
389
390
391     mrf24j40 long addr write(0x11, extended address[7]);    // LSB
392     mrf24j40 long addr write(0x12, extended address[6]);
393     mrf24j40 long addr write(0x13, extended address[5]);
394     mrf24j40 long addr write(0x14, extended address[4]);
395     mrf24j40 long addr write(0x15, extended address[3]);
396     mrf24j40 long addr write(0x16, extended address[2]);
397     mrf24j40 long addr write(0x17, extended address[1]);
398     mrf24j40 long addr write(0x18, extended address[0]);    // MSB
399
400     for (uns8 count=0;  count < data_length; count++) {
401         mrf24j40 long addr write(count+header_length+2,  data[count]);
402     }
403
404     uns8 txncon = mrf24j40 short addr read(TXNCON);
405
406     set_bit(txncon, TXNCON TXNTRIG);
407     if (ack) {
408         set_bit(txncon, TXNCON TXNACKREQ);
409     }
410     mrf24j40 short addr write(TXNCON, txncon);
411
412
413 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_transmit_to_short_address (uns8 *frame_type*, uns16 *dest_pan_id*, uns16 *dest_short_address*, uns8 * *data*, uns8 *bytes_to_transmit*, uns8 *ack*)

Requests that the mrf24j40 transmit the packet to the specified short address

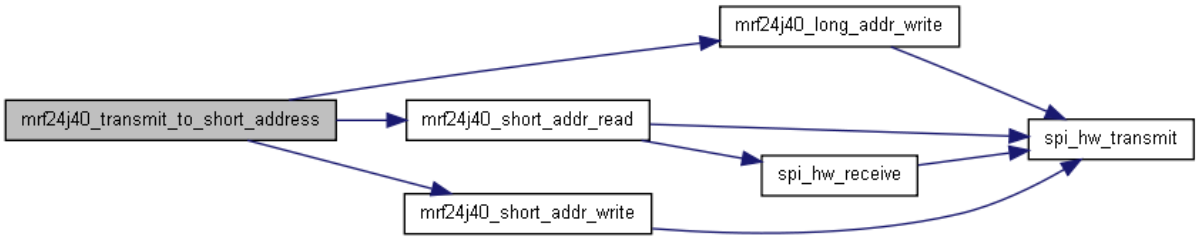
Parameters:

frame_type 802.15.4 frame type
dest_pan_id PAN id of destination
dest_short_address 16 bit short address of destination
data Pointer to uns8 array of bytes to transmit
bytes_to_transmit
ack Either MRF_ACK or MRF_NO_ACK depending if you want hardware acknowledgement

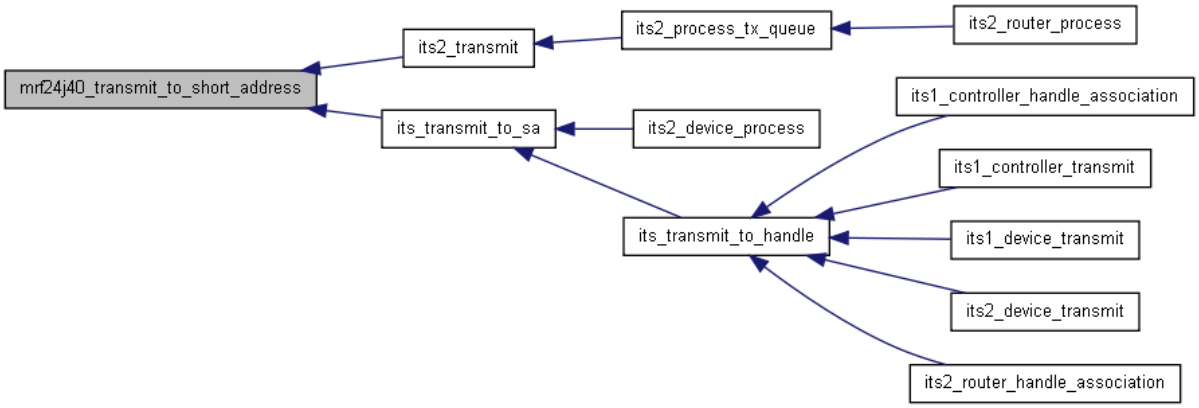
```
416
{
417
418     uns8 fc_msb = 0b10001000;   // short dest (10,11) short src (14,15)
419     uns8 fc_lsb = 0b00000000 | frame_type;  // data, pan id compression (only have dest pan
id)
420     // To do:
421     // Not smart enough for this yet:
422     //if (dest_pan_id == pan_id) {
423     //  set_bit(fc_lsb, 6);     // pan compression
424     //}
425     if (ack) {
426         set_bit(fc_lsb, 5); // ack bit
427     }
428
429     data sequence number++;
430
431     uns8 header length = 3+2+2+2+2;
432     uns8 frame_length = header_length + bytes_to_transmit;
433
434     mrf24j40 long addr write(0x00, header_length);
435     mrf24j40 long addr write(0x01, frame_length);
436
437     mrf24j40 long addr write(0x02, fc_lsb);
438     mrf24j40 long addr write(0x03, fc_msb);
439     mrf24j40 long addr write(0x04, data sequence number);
440
441     mrf24j40 long addr write(0x05, dest_pan_id & 0xff); // dest pan id  LSB
442     mrf24j40 long addr write(0x06, dest_pan_id >> 8);   // MSB
443
444     mrf24j40 long addr write(0x07, dest_short_address & 0xff); // LSB
445     mrf24j40 long addr write(0x08, dest_short_address >> 8);    // MSB
446
447     mrf24j40 long addr write(0x09, pan id & 0xff);  // src pan id  (=ours) LSB
448     mrf24j40 long addr write(0x0a, pan id >> 8);    // MSB
449
450
451     mrf24j40 long addr write(0x0b, short address & 0xff);   // LSB
```

```
452     mrf24j40_long_addr_write(0x0c, short_address >> 8);
453
454     for (uns8 count=0;  count < bytes_to_transmit; count++) { // check format here
455         mrf24j40_long_addr_write(count+header_length+2,  data[count]);
456     }
457
458     uns8 txncon = mrf24j40_short_addr_read(TXNCON);
459     set_bit(txncon, TXNCON_TXNTRIG);
460     if (ack) {
461         set_bit(txncon, TXNCON_TXNACKREQ);
462     }
463     mrf24j40_short_addr_write(TXNCON, txncon);
464
465
466 }
```

Here is the call graph for this function:



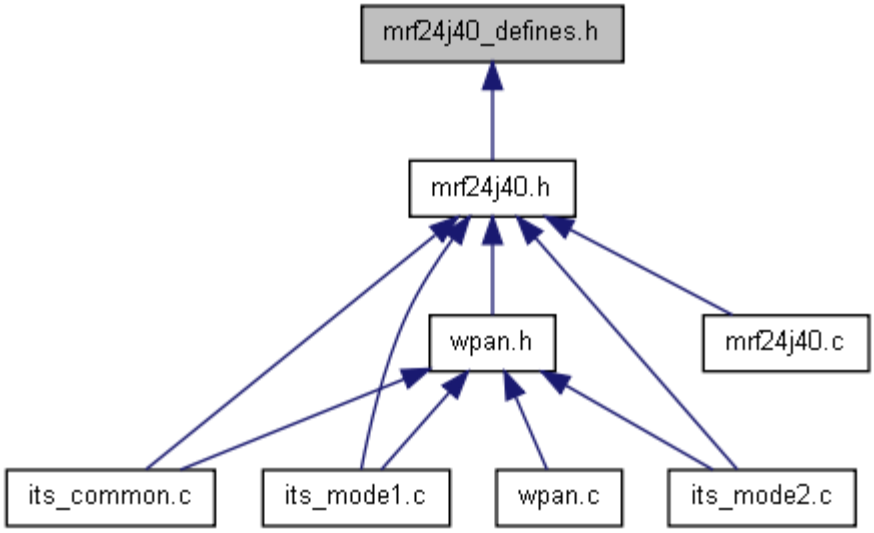
Here is the caller graph for this function:



---

mrf24j40_defines.h File Reference

Defines for MRF24J40 chip - generated from the datasheet.
This graph shows which files directly or indirectly include this file:



Defines

- #define ACKTMOUT 0x12
- #define ACKTMOUT_DRPACK 7
- #define ACKTMOUT_MAWD0 0
- #define ACKTMOUT_MAWD1 1
- #define ACKTMOUT_MAWD2 2
- #define ACKTMOUT_MAWD3 3
- #define ACKTMOUT_MAWD4 4
- #define ACKTMOUT_MAWD5 5
- #define ACKTMOUT_MAWD6 6
- #define ASSOEADR0 0x230
- #define ASSOEADR1 0x231
- #define ASSOEADR2 0x232
- #define ASSOEADR3 0x233
- #define ASSOEADR4 0x234
- #define ASSOEADR5 0x235
- #define ASSOEADR6 0x236
- #define ASSOEADR7 0x237
- #define ASSOSADR0 0x238
- #define ASSOSADR1 0x239
- #define BBREG0 0x38
- #define BBREG0_TURBO 0
- #define BBREG1 0x39
- #define BBREG1_RXDECINV 2
- #define BBREG2 0x3A
- #define BBREG2_CCACSTH0 2
- #define BBREG2_CCACSTH1 3
- #define BBREG2_CCACSTH2 4
- #define BBREG2_CCACSTH3 5
- #define BBREG2_CCAMODE0 6
- #define BBREG2_CCAMODE1 7
- #define BBREG3 0x3B
- #define BBREG3_PREDETTH0 1

- #define BBREG3_PREDETTH1 2
- #define BBREG3_PREDETTH2 3
- #define BBREG3_PREVALIDTH0 4
- #define BBREG3_PREVALIDTH1 5
- #define BBREG3_PREVALIDTH2 6
- #define BBREG3_PREVALIDTH3 7
- #define BBREG4 0x3C
- #define BBREG4_CSTH0 5
- #define BBREG4_CSTH1 6
- #define BBREG4_CSTH2 7
- #define BBREG4_PRECNT0 2
- #define BBREG4_PRECNT1 3
- #define BBREG4_PRECNT2 4
- #define BBREG6 0x3E
- #define BBREG6_RSSIMODE1 7
- #define BBREG6_RSSIMODE2 6
- #define BBREG6_RSSIRDY 0
- #define CCAEDTH 0x3F
- #define CCAEDTH_CCAEDTH0 0
- #define CCAEDTH_CCAEDTH1 1
- #define CCAEDTH_CCAEDTH2 2
- #define CCAEDTH_CCAEDTH3 3
- #define CCAEDTH_CCAEDTH4 4
- #define CCAEDTH_CCAEDTH5 5
- #define CCAEDTH_CCAEDTH6 6
- #define CCAEDTH_CCAEDTH7 7
- #define EADR0 0x05
- #define EADR1 0x06
- #define EADR2 0x07
- #define EADR3 0x08
- #define EADR4 0x09
- #define EADR5 0x0A
- #define EADR6 0x0B
- #define EADR7 0x0C
- #define ESLOTG1 0x13
- #define ESLOTG23 0x1E
- #define ESLOTG45 0x1F
- #define ESLOTG67 0x20
- #define FRMOFFSET 0x23
- #define FRMOFFSET_OFFSET0 0
- #define FRMOFFSET_OFFSET1 1
- #define FRMOFFSET_OFFSET2 2
- #define FRMOFFSET_OFFSET3 3
- #define FRMOFFSET_OFFSET4 4
- #define FRMOFFSET_OFFSET5 5
- #define FRMOFFSET_OFFSET6 6
- #define FRMOFFSET_OFFSET7 7
- #define GATECLK 0x26
- #define GATECLK_GTSON 3
- #define GPIO 0x33
- #define GPIO_GPIO0 0
- #define GPIO_GPIO1 1
- #define GPIO_GPIO2 2

- #define GPIO_GPIO3 3
- #define GPIO_GPIO4 4
- #define GPIO_GPIO5 5
- #define HSYMTMRH 0x29
- #define HSYMTMRH_HSYMTMR08 0
- #define HSYMTMRH_HSYMTMR09 1
- #define HSYMTMRH_HSYMTMR10 2
- #define HSYMTMRH_HSYMTMR11 3
- #define HSYMTMRH_HSYMTMR12 4
- #define HSYMTMRH_HSYMTMR13 5
- #define HSYMTMRH_HSYMTMR14 6
- #define HSYMTMRH_HSYMTMR15 7
- #define HSYMTMRL 0x28
- #define HSYMTMRL_HSYMTMR0 0
- #define HSYMTMRL_HSYMTMR1 1
- #define HSYMTMRL_HSYMTMR2 2
- #define HSYMTMRL_HSYMTMR3 3
- #define HSYMTMRL_HSYMTMR4 4
- #define HSYMTMRL_HSYMTMR5 5
- #define HSYMTMRL_HSYMTMR6 6
- #define HSYMTMRL_HSYMTMR7 7
- #define INTSTAT 0x31
- #define INTSTAT_HSYMTMRIF 5
- #define INTSTAT_RXIF 3
- #define INTSTAT_SECIF 4
- #define INTSTAT_SLPIF 7
- #define INTSTAT_TXG1IF 1
- #define INTSTAT_TXG2IF 2
- #define INTSTAT_TXNIF 0
- #define INTSTAT_WAKEIF 6
- #define MAINCNT0 0x226
- #define MAINCNT0_MAINCNT0 0
- #define MAINCNT0_MAINCNT1 1
- #define MAINCNT0_MAINCNT2 2
- #define MAINCNT0_MAINCNT3 3
- #define MAINCNT0_MAINCNT4 4
- #define MAINCNT0_MAINCNT5 5
- #define MAINCNT0_MAINCNT6 6
- #define MAINCNT0_MAINCNT7 7
- #define MAINCNT1 0x227
- #define MAINCNT1_MAINCNT10 2
- #define MAINCNT1_MAINCNT11 3
- #define MAINCNT1_MAINCNT12 4
- #define MAINCNT1_MAINCNT13 5
- #define MAINCNT1_MAINCNT14 6
- #define MAINCNT1_MAINCNT15 7
- #define MAINCNT1_MAINCNT8 0
- #define MAINCNT1_MAINCNT9 1
- #define MAINCNT2 0x228
- #define MAINCNT2_MAINCNT16 0
- #define MAINCNT2_MAINCNT17 1
- #define MAINCNT2_MAINCNT18 2
- #define MAINCNT2_MAINCNT19 3

- #define MAINCNT2_MAINCNT20 4
- #define MAINCNT2_MAINCNT21 5
- #define MAINCNT2_MAINCNT22 6
- #define MAINCNT2_MAINCNT23 7
- #define MAINCNT3 0x229
- #define MAINCNT3_MAINCNT24 0
- #define MAINCNT3_MAINCNT25 1
- #define MAINCNT3_STARTCNT 7
- #define MRF_INTCON 0x32
- #define MRF_INTCON_HSYMTMRIE 5
- #define MRF_INTCON_RXIE 3
- #define MRF_INTCON_SECIE 4
- #define MRF_INTCON_SLPIE 7
- #define MRF_INTCON_TXG1IE 1
- #define MRF_INTCON_TXG2IE 2
- #define MRF_INTCON_TXNIE 0
- #define MRF_INTCON_WAKEIE 6
- #define ORDER 0x10
- #define ORDER_BO0 4
- #define ORDER_BO1 5
- #define ORDER_BO2 6
- #define ORDER_BO3 7
- #define ORDER_SO0 0
- #define ORDER_SO1 1
- #define ORDER_SO2 2
- #define ORDER_SO3 3
- #define PACON0 0x16
- #define PACON0_PAONT0 0
- #define PACON0_PAONT1 1
- #define PACON0_PAONT2 2
- #define PACON0_PAONT3 3
- #define PACON0_PAONT4 4
- #define PACON0_PAONT5 5
- #define PACON0_PAONT6 6
- #define PACON0_PAONT7 7
- #define PACON1 0x17
- #define PACON1_PAONT8 0
- #define PACON1_PAONTS0 1
- #define PACON1_PAONTS1 2
- #define PACON1_PAONTS2 3
- #define PACON1_PAONTS3 4
- #define PACON2 0x18
- #define PACON2_FIFOEN 7
- #define PACON2_TXONT7 0
- #define PACON2_TXONT8 1
- #define PACON2_TXONTS0 2
- #define PACON2_TXONTS1 3
- #define PACON2_TXONTS2 4
- #define PACON2_TXONTS3 5
- #define PANIDH 0x02
- #define PANIDL 0x01
- #define REMCNTH 0x225
- #define REMCNTH_REMCNT10 2

- #define REMCNTH_REMCNT11 3
- #define REMCNTH_REMCNT12 4
- #define REMCNTH_REMCNT13 5
- #define REMCNTH_REMCNT14 6
- #define REMCNTH_REMCNT15 7
- #define REMCNTH_REMCNT8 0
- #define REMCNTH_REMCNT9 1
- #define REMCNTL 0x224
- #define REMCNTL_REMCNT0 0
- #define REMCNTL_REMCNT1 1
- #define REMCNTL_REMCNT2 2
- #define REMCNTL_REMCNT3 3
- #define REMCNTL_REMCNT4 4
- #define REMCNTL_REMCNT5 5
- #define REMCNTL_REMCNT6 6
- #define REMCNTL_REMCNT7 7
- #define RFCON0 0x200
- #define RFCON0_CHANNEL0 4
- #define RFCON0_CHANNEL1 5
- #define RFCON0_CHANNEL2 6
- #define RFCON0_CHANNEL3 7
- #define RFCON0_RFOPT0 0
- #define RFCON0_RFOPT1 1
- #define RFCON0_RFOPT2 2
- #define RFCON0_RFOPT3 3
- #define RFCON1 0x201
- #define RFCON1_VCOOPT0 0
- #define RFCON1_VCOOPT1 1
- #define RFCON1_VCOOPT2 2
- #define RFCON1_VCOOPT3 3
- #define RFCON1_VCOOPT4 4
- #define RFCON1_VCOOPT5 5
- #define RFCON1_VCOOPT6 6
- #define RFCON1_VCOOPT7 7
- #define RFCON2 0x202
- #define RFCON2_PLLEN 7
- #define RFCON3 0x203
- #define RFCON3_TXPWRL0 6
- #define RFCON3_TXPWRL1 7
- #define RFCON3_TXPWRS0 3
- #define RFCON3_TXPWRS1 4
- #define RFCON3_TXPWRS2 5
- #define RFCON5 0x205
- #define RFCON5_BATTH0 4
- #define RFCON5_BATTH1 5
- #define RFCON5_BATTH2 6
- #define RFCON5_BATTH3 7
- #define RFCON6 0x206
- #define RFCON6_20MRECVR 4
- #define RFCON6_BATEN 3
- #define RFCON6_TXFIL 7
- #define RFCON7 0x207
- #define RFCON7_CLKOUTMODE0 0

- #define RFCON7_CLKOUTMODE1 1
- #define RFCON7_SLPCLKSEL0 6
- #define RFCON7_SLPCLKSEL1 7
- #define RFCON8 0x208
- #define RFCON8_RFVCO 4
- #define RFCTL 0x36
- #define RFCTL_RFRST 2
- #define RFCTL_WAKECNT7 3
- #define RFCTL_WAKECNT8 4
- #define RFSTATE 0x20F
- #define RFSTATE_RFSTATE0 5
- #define RFSTATE_RFSTATE1 6
- #define RFSTATE_RFSTATE2 7
- #define RSSI 0x210
- #define RSSI_RSSI0 0
- #define RSSI_RSSI1 1
- #define RSSI_RSSI2 2
- #define RSSI_RSSI3 3
- #define RSSI_RSSI4 4
- #define RSSI_RSSI5 5
- #define RSSI_RSSI6 6
- #define RSSI_RSSI7 7
- #define RXFLUSH 0x0D
- #define RXFLUSH_BCNONLY 1
- #define RXFLUSH_CMDONLY 3
- #define RXFLUSH_DATAONLY 2
- #define RXFLUSH_RXFLUSH 0
- #define RXFLUSH_WAKEPAD 5
- #define RXFLUSH_WAKEPOL 6
- #define RXMCR 0x00
- #define RXMCR_COORD 2
- #define RXMCR_ERRPKT 1
- #define RXMCR_NOACKRSP 5
- #define RXMCR_PANCOORD 3
- #define RXMCR_PROMI 0
- #define RXSR 0x30
- #define RXSR_BATIND 5
- #define RXSR_UPSECERR 6
- #define SADRH 0x04
- #define SADRL 0x03
- #define SECCON0 0x2C
- #define SECCON0_RXCIPHER0 3
- #define SECCON0_RXCIPHER1 4
- #define SECCON0_RXCIPHER2 5
- #define SECCON0_SECIGNORE 7
- #define SECCON0_SECSTART 6
- #define SECCON0_TXNCIPHER0 0
- #define SECCON0_TXNCIPHER1 1
- #define SECCON0_TXNCIPHER2 2
- #define SECCON1 0x2D
- #define SECCON1_DISDEC 1
- #define SECCON1_DISENC 0
- #define SECCON1_TXBCIPHER0 4

- #define SECCON1_TXBCIPHER1 5
- #define SECCON1_TXBCIPHER2 6
- #define SECCR2 0x37
- #define SECCR2_TXG1CIPHER0 0
- #define SECCR2_TXG1CIPHER1 1
- #define SECCR2_TXG1CIPHER2 2
- #define SECCR2_TXG2CIPHER0 3
- #define SECCR2_TXG2CIPHER1 4
- #define SECCR2_TXG2CIPHER2 5
- #define SECCR2_UPDEC 7
- #define SECCR2_UPENC 6
- #define SLPACK 0x35
- #define SLPACK_SLPACK 7
- #define SLPACK_WAKECNT0 0
- #define SLPACK_WAKECNT1 1
- #define SLPACK_WAKECNT2 2
- #define SLPACK_WAKECNT3 3
- #define SLPACK_WAKECNT4 4
- #define SLPACK_WAKECNT5 5
- #define SLPACK_WAKECNT6 6
- #define SLPCAL0 0x209
- #define SLPCAL0_SLPCAL0 0
- #define SLPCAL0_SLPCAL1 1
- #define SLPCAL0_SLPCAL2 2
- #define SLPCAL0_SLPCAL3 3
- #define SLPCAL0_SLPCAL4 4
- #define SLPCAL0_SLPCAL5 5
- #define SLPCAL0_SLPCAL6 6
- #define SLPCAL0_SLPCAL7 7
- #define SLPCAL1 0x20A
- #define SLPCAL1_SLPCAL10 2
- #define SLPCAL1_SLPCAL11 3
- #define SLPCAL1_SLPCAL12 4
- #define SLPCAL1_SLPCAL13 5
- #define SLPCAL1_SLPCAL14 6
- #define SLPCAL1_SLPCAL15 7
- #define SLPCAL1_SLPCAL8 0
- #define SLPCAL1_SLPCAL9 1
- #define SLPCAL2 0x20B
- #define SLPCAL2_SLPCAL16 0
- #define SLPCAL2_SLPCAL17 1
- #define SLPCAL2_SLPCAL18 2
- #define SLPCAL2_SLPCAL19 3
- #define SLPCAL2_SLPCALEN 4
- #define SLPCAL2_SLPCALRDY 7
- #define SLPCON0 0x211
- #define SLPCON0_INTEDGE 1
- #define SLPCON0_SLPCLKEN 0
- #define SLPCON1 0x220
- #define SLPCON1_CLKOUTEN 5
- #define SLPCON1_SLPCLKDIV0 0
- #define SLPCON1_SLPCLKDIV1 1
- #define SLPCON1_SLPCLKDIV2 2

- #define SLPCON1_SLPCLKDIV3 3
- #define SLPCON1_SLPCLKDIV4 4
- #define SOFTRST 0x2A
- #define SOFTRST_RSTBB 1
- #define SOFTRST_RSTMAC 0
- #define SOFTRST_RSTPWR 2
- #define SYMTICKH 0x15
- #define SYMTICKH_TICKP8 0
- #define SYMTICKH_TXONT0 1
- #define SYMTICKH_TXONT1 2
- #define SYMTICKH_TXONT2 3
- #define SYMTICKH_TXONT3 4
- #define SYMTICKH_TXONT4 5
- #define SYMTICKH_TXONT5 6
- #define SYMTICKH_TXONT6 7
- #define SYMTICKL 0x14
- #define SYMTICKL_TICKP0 0
- #define SYMTICKL_TICKP1 1
- #define SYMTICKL_TICKP2 2
- #define SYMTICKL_TICKP3 3
- #define SYMTICKL_TICKP4 4
- #define SYMTICKL_TICKP5 5
- #define SYMTICKL_TICKP6 6
- #define SYMTICKL_TICKP7 7
- #define TESTMODE 0x22F
- #define TESTMODE_RSSIWAIT0 3
- #define TESTMODE_RSSIWAIT1 4
- #define TESTMODE_TESTMODE0 0
- #define TESTMODE_TESTMODE1 1
- #define TESTMODE_TESTMODE2 2
- #define TRISGPIO 0x34
- #define TRISGPIO_TRISGP0 0
- #define TRISGPIO_TRISGP1 1
- #define TRISGPIO_TRISGP2 2
- #define TRISGPIO_TRISGP3 3
- #define TRISGPIO_TRISGP4 4
- #define TRISGPIO_TRISGP5 5
- #define TXBCON0 0x1A
- #define TXBCON0_TXBSECEN 1
- #define TXBCON0_TXBTRIG 0
- #define TXBCON1 0x25
- #define TXG1CON 0x1C
- #define TXG1CON_TXG1ACKREQ 2
- #define TXG1CON_TXG1RETRY0 6
- #define TXG1CON_TXG1RETRY1 7
- #define TXG1CON_TXG1SECEN 1
- #define TXG1CON_TXG1SLOT0 3
- #define TXG1CON_TXG1SLOT1 4
- #define TXG1CON_TXG1SLOT2 5
- #define TXG1CON_TXG1TRIG 0
- #define TXG2CON 0x1D
- #define TXG2CON_TXG2ACKREQ 2
- #define TXG2CON_TXG2RETRY0 6

- #define TXG2CON_TXG2RETRY1 7
- #define TXG2CON_TXG2SECEN 1
- #define TXG2CON_TXG2SLOT0 3
- #define TXG2CON_TXG2SLOT1 4
- #define TXG2CON_TXG2SLOT2 5
- #define TXG2CON_TXG2TRIG 0
- #define TXMCR 0x11
- #define TXMCR_BATLIFEXT 6
- #define TXMCR_CSMABF0 0
- #define TXMCR_CSMABF1 1
- #define TXMCR_CSMABF2 2
- #define TXMCR_MACMINBE0 3
- #define TXMCR_MACMINBE1 4
- #define TXMCR_NOCSMA 7
- #define TXMCR_SLOTTED 5
- #define TXNCON 0x1B
- #define TXNCON_FPSTAT 4
- #define TXNCON_INDIRECT 3
- #define TXNCON_TXNACKREQ 2
- #define TXNCON_TXNSECEN 1
- #define TXNCON_TXNTRIG 0
- #define TXPEND 0x21
- #define TXPEND_FPACK 0
- #define TXPEND_GTSSWITCH 1
- #define TXPEND_MLIFS0 2
- #define TXPEND_MLIFS1 3
- #define TXPEND_MLIFS2 4
- #define TXPEND_MLIFS3 5
- #define TXPEND_MLIFS4 6
- #define TXPEND_MLIFS5 7
- #define TXSTAT 0x24
- #define TXSTAT_CCAFAIL 5
- #define TXSTAT_TXG1FNT 3
- #define TXSTAT_TXG1STAT 1
- #define TXSTAT_TXG2FNT 4
- #define TXSTAT_TXG2STAT 2
- #define TXSTAT_TXNRETRY0 6
- #define TXSTAT_TXNRETRY1 7
- #define TXSTAT_TXNSTAT 0
- #define TXSTBL 0x2E
- #define TXSTBL_MSIFS0 0
- #define TXSTBL_MSIFS1 1
- #define TXSTBL_MSIFS2 2
- #define TXSTBL_MSIFS3 3
- #define TXSTBL_RFSTBL0 4
- #define TXSTBL_RFSTBL1 5
- #define TXSTBL_RFSTBL2 6
- #define TXSTBL_RFSTBL3 7
- #define TXTIME 0x27
- #define TXTIME_TURNTIME0 4
- #define TXTIME_TURNTIME1 5
- #define TXTIME_TURNTIME2 6
- #define TXTIME_TURNTIME3 7

- #define UPNONCE0 0x240
- #define UPNONCE1 0x241
- #define UPNONCE10 0x24A
- #define UPNONCE11 0x24B
- #define UPNONCE12 0x24C
- #define UPNONCE2 0x242
- #define UPNONCE3 0x243
- #define UPNONCE4 0x244
- #define UPNONCE5 0x245
- #define UPNONCE6 0x246
- #define UPNONCE7 0x247
- #define UPNONCE8 0x248
- #define UPNONCE9 0x249
- #define WAKECON 0x22
- #define WAKECON_IMMWAKE 7
- #define WAKECON_REGWAKE 6
- #define WAKETIMEH 0x223
- #define WAKETIMEH_WAKETIME10 2
- #define WAKETIMEH_WAKETIME8 0
- #define WAKETIMEH_WAKETIME9 1
- #define WAKETIMEL 0x222
- #define WAKETIMEL_WAKETIME0 0
- #define WAKETIMEL_WAKETIME1 1
- #define WAKETIMEL_WAKETIME2 2
- #define WAKETIMEL_WAKETIME3 3
- #define WAKETIMEL_WAKETIME4 4
- #define WAKETIMEL_WAKETIME5 5
- #define WAKETIMEL_WAKETIME6 6
- #define WAKETIMEL_WAKETIME7 7

Detailed Description

Define Documentation

#define ACKTMOUT 0x12

#define ACKTMOUT_DRPACK 7

#define ACKTMOUT_MAWD0 0

#define ACKTMOUT_MAWD1 1

#define ACKTMOUT_MAWD2 2

#define ACKTMOUT_MAWD3 3

#define ACKTMOUT_MAWD4 4

#define ACKTMOUT_MAWD5 5

#define ACKTMOUT_MAWD6 6

#define ASSOEADR0 0x230

#define ASSOEADR1 0x231

#define ASSOEADR2 0x232

#define ASSOEADR3 0x233

#define ASSOEADR4 0x234

#define ASSOEADR5 0x235

#define ASSOEADR6 0x236

#define ASSOEADR7 0x237

#define ASSOSADR0 0x238

#define ASSOSADR1 0x239

#define BBREG0 0x38

#define BBREG0_TURBO 0

#define BBREG1 0x39

#define BBREG1_RXDECINV 2

#define BBREG2 0x3A

#define BBREG2_CCACSTH0 2

#define BBREG2_CCACSTH1 3

```
#define BBREG2_CCACSTH2  4
#define BBREG2_CCACSTH3  5
#define BBREG2_CCAMODE0  6
#define BBREG2_CCAMODE1  7
#define BBREG3  0x3B
#define BBREG3_PREDETTH0  1
#define BBREG3_PREDETTH1  2
#define BBREG3_PREDETTH2  3
#define BBREG3_PREVALIDTH0  4
#define BBREG3_PREVALIDTH1  5
#define BBREG3_PREVALIDTH2  6
#define BBREG3_PREVALIDTH3  7
#define BBREG4  0x3C
#define BBREG4_CSTH0  5
#define BBREG4_CSTH1  6
#define BBREG4_CSTH2  7
#define BBREG4_PRECNT0  2
#define BBREG4_PRECNT1  3
#define BBREG4_PRECNT2  4
#define BBREG6  0x3E
#define BBREG6_RSSIMODE1  7
#define BBREG6_RSSIMODE2  6
#define BBREG6_RSSIRDY  0
#define CCAEDTH  0x3F
#define CCAEDTH_CCAEDTH0  0
#define CCAEDTH_CCAEDTH1  1
#define CCAEDTH_CCAEDTH2  2
#define CCAEDTH_CCAEDTH3  3
```

```
#define CCAEDTH_CCAEDTH4  4
#define CCAEDTH_CCAEDTH5  5
#define CCAEDTH_CCAEDTH6  6
#define CCAEDTH_CCAEDTH7  7
#define EADR0  0x05
#define EADR1  0x06
#define EADR2  0x07
#define EADR3  0x08
#define EADR4  0x09
#define EADR5  0x0A
#define EADR6  0x0B
#define EADR7  0x0C
#define ESLOTG1  0x13
#define ESLOTG23  0x1E
#define ESLOTG45  0x1F
#define ESLOTG67  0x20
#define FRMOFFSET  0x23
#define FRMOFFSET_OFFSET0  0
#define FRMOFFSET_OFFSET1  1
#define FRMOFFSET_OFFSET2  2
#define FRMOFFSET_OFFSET3  3
#define FRMOFFSET_OFFSET4  4
#define FRMOFFSET_OFFSET5  5
#define FRMOFFSET_OFFSET6  6
#define FRMOFFSET_OFFSET7  7
#define GATECLK  0x26
#define GATECLK_GTSON  3
#define GPIO  0x33
```

```
#define GPIO_GPIO0  0
#define GPIO_GPIO1  1
#define GPIO_GPIO2  2
#define GPIO_GPIO3  3
#define GPIO_GPIO4  4
#define GPIO_GPIO5  5
#define HSYMTMRH  0x29
#define HSYMTMRH_HSYMTMR08  0
#define HSYMTMRH_HSYMTMR09  1
#define HSYMTMRH_HSYMTMR10  2
#define HSYMTMRH_HSYMTMR11  3
#define HSYMTMRH_HSYMTMR12  4
#define HSYMTMRH_HSYMTMR13  5
#define HSYMTMRH_HSYMTMR14  6
#define HSYMTMRH_HSYMTMR15  7
#define HSYMTMRL  0x28
#define HSYMTMRL_HSYMTMR0  0
#define HSYMTMRL_HSYMTMR1  1
#define HSYMTMRL_HSYMTMR2  2
#define HSYMTMRL_HSYMTMR3  3
#define HSYMTMRL_HSYMTMR4  4
#define HSYMTMRL_HSYMTMR5  5
#define HSYMTMRL_HSYMTMR6  6
#define HSYMTMRL_HSYMTMR7  7
#define INTSTAT  0x31
#define INTSTAT_HSYMTMRIF  5
#define INTSTAT_RXIF  3
#define INTSTAT_SECIF  4
```

```
#define INTSTAT_SLPIF   7
#define INTSTAT_TXG1IF   1
#define INTSTAT_TXG2IF   2
#define INTSTAT_TXNIF   0
#define INTSTAT_WAKEIF   6
#define MAINCNT0   0x226
#define MAINCNT0_MAINCNT0   0
#define MAINCNT0_MAINCNT1   1
#define MAINCNT0_MAINCNT2   2
#define MAINCNT0_MAINCNT3   3
#define MAINCNT0_MAINCNT4   4
#define MAINCNT0_MAINCNT5   5
#define MAINCNT0_MAINCNT6   6
#define MAINCNT0_MAINCNT7   7
#define MAINCNT1   0x227
#define MAINCNT1_MAINCNT10   2
#define MAINCNT1_MAINCNT11   3
#define MAINCNT1_MAINCNT12   4
#define MAINCNT1_MAINCNT13   5
#define MAINCNT1_MAINCNT14   6
#define MAINCNT1_MAINCNT15   7
#define MAINCNT1_MAINCNT8   0
#define MAINCNT1_MAINCNT9   1
#define MAINCNT2   0x228
#define MAINCNT2_MAINCNT16   0
#define MAINCNT2_MAINCNT17   1
#define MAINCNT2_MAINCNT18   2
#define MAINCNT2_MAINCNT19   3
```

```
#define MAINCNT2_MAINCNT20  4
#define MAINCNT2_MAINCNT21  5
#define MAINCNT2_MAINCNT22  6
#define MAINCNT2_MAINCNT23  7
#define MAINCNT3  0x229
#define MAINCNT3_MAINCNT24  0
#define MAINCNT3_MAINCNT25  1
#define MAINCNT3_STARTCNT  7
#define MRF_INTCON  0x32
#define MRF_INTCON_HSYMTMRIE  5
#define MRF_INTCON_RXIE  3
#define MRF_INTCON_SECIE  4
#define MRF_INTCON_SLPIE  7
#define MRF_INTCON_TXG1IE  1
#define MRF_INTCON_TXG2IE  2
#define MRF_INTCON_TXNIE  0
#define MRF_INTCON_WAKEIE  6
#define ORDER  0x10
#define ORDER_BO0  4
#define ORDER_BO1  5
#define ORDER_BO2  6
#define ORDER_BO3  7
#define ORDER_SO0  0
#define ORDER_SO1  1
#define ORDER_SO2  2
#define ORDER_SO3  3
#define PACON0  0x16
#define PACON0_PAONT0  0
```

```
#define PACON0_PAONT1  1
#define PACON0_PAONT2  2
#define PACON0_PAONT3  3
#define PACON0_PAONT4  4
#define PACON0_PAONT5  5
#define PACON0_PAONT6  6
#define PACON0_PAONT7  7
#define PACON1  0x17
#define PACON1_PAONT8  0
#define PACON1_PAONTS0  1
#define PACON1_PAONTS1  2
#define PACON1_PAONTS2  3
#define PACON1_PAONTS3  4
#define PACON2  0x18
#define PACON2_FIFOEN  7
#define PACON2_TXONT7  0
#define PACON2_TXONT8  1
#define PACON2_TXONTS0  2
#define PACON2_TXONTS1  3
#define PACON2_TXONTS2  4
#define PACON2_TXONTS3  5
#define PANIDH  0x02
#define PANIDL  0x01
#define REMCNTH  0x225
#define REMCNTH_REMCNT10  2
#define REMCNTH_REMCNT11  3
#define REMCNTH_REMCNT12  4
#define REMCNTH_REMCNT13  5
```

```
#define REMCNTH_REMCNT14   6
#define REMCNTH_REMCNT15   7
#define REMCNTH_REMCNT8   0
#define REMCNTH_REMCNT9   1
#define REMCNTL   0x224
#define REMCNTL_REMCNT0   0
#define REMCNTL_REMCNT1   1
#define REMCNTL_REMCNT2   2
#define REMCNTL_REMCNT3   3
#define REMCNTL_REMCNT4   4
#define REMCNTL_REMCNT5   5
#define REMCNTL_REMCNT6   6
#define REMCNTL_REMCNT7   7
#define RFCON0   0x200
#define RFCON0_CHANNEL0   4
#define RFCON0_CHANNEL1   5
#define RFCON0_CHANNEL2   6
#define RFCON0_CHANNEL3   7
#define RFCON0_RFOPT0   0
#define RFCON0_RFOPT1   1
#define RFCON0_RFOPT2   2
#define RFCON0_RFOPT3   3
#define RFCON1   0x201
#define RFCON1_VCOOPT0   0
#define RFCON1_VCOOPT1   1
#define RFCON1_VCOOPT2   2
#define RFCON1_VCOOPT3   3
#define RFCON1_VCOOPT4   4
```

```
#define RFCON1_VCOOPT5  5
#define RFCON1_VCOOPT6  6
#define RFCON1_VCOOPT7  7
#define RFCON2  0x202
#define RFCON2_PLLEN  7
#define RFCON3  0x203
#define RFCON3_TXPWRL0  6
#define RFCON3_TXPWRL1  7
#define RFCON3_TXPWRS0  3
#define RFCON3_TXPWRS1  4
#define RFCON3_TXPWRS2  5
#define RFCON5  0x205
#define RFCON5_BATTH0  4
#define RFCON5_BATTH1  5
#define RFCON5_BATTH2  6
#define RFCON5_BATTH3  7
#define RFCON6  0x206
#define RFCON6_20MRECVR  4
#define RFCON6_BATEN  3
#define RFCON6_TXFIL  7
#define RFCON7  0x207
#define RFCON7_CLKOUTMODE0  0
#define RFCON7_CLKOUTMODE1  1
#define RFCON7_SLPCLKSEL0  6
#define RFCON7_SLPCLKSEL1  7
#define RFCON8  0x208
#define RFCON8_RFVCO  4
#define RFCTL  0x36
```

```
#define RFCTL_RFRST  2
#define RFCTL_WAKECNT7  3
#define RFCTL_WAKECNT8  4
#define RFSTATE  0x20F
#define RFSTATE_RFSTATE0  5
#define RFSTATE_RFSTATE1  6
#define RFSTATE_RFSTATE2  7
#define RSSI  0x210
#define RSSI_RSSI0  0
#define RSSI_RSSI1  1
#define RSSI_RSSI2  2
#define RSSI_RSSI3  3
#define RSSI_RSSI4  4
#define RSSI_RSSI5  5
#define RSSI_RSSI6  6
#define RSSI_RSSI7  7
#define RXFLUSH  0x0D
#define RXFLUSH_BCNONLY  1
#define RXFLUSH_CMDONLY  3
#define RXFLUSH_DATAONLY  2
#define RXFLUSH_RXFLUSH  0
#define RXFLUSH_WAKEPAD  5
#define RXFLUSH_WAKEPOL  6
#define RXMCR  0x00
#define RXMCR_COORD  2
#define RXMCR_ERRPKT  1
#define RXMCR_NOACKRSP  5
#define RXMCR_PANCOORD  3
```

```
#define RXMCR_PROMI  0
#define RXSR  0x30
#define RXSR_BATIND  5
#define RXSR_UPSECERR  6
#define SADRH  0x04
#define SADRL  0x03
#define SECCON0  0x2C
#define SECCON0_RXCIPHER0  3
#define SECCON0_RXCIPHER1  4
#define SECCON0_RXCIPHER2  5
#define SECCON0_SECIGNORE  7
#define SECCON0_SECSTART  6
#define SECCON0_TXNCIPHER0  0
#define SECCON0_TXNCIPHER1  1
#define SECCON0_TXNCIPHER2  2
#define SECCON1  0x2D
#define SECCON1_DISDEC  1
#define SECCON1_DISENC  0
#define SECCON1_TXBCIPHER0  4
#define SECCON1_TXBCIPHER1  5
#define SECCON1_TXBCIPHER2  6
#define SECCR2  0x37
#define SECCR2_TXG1CIPHER0  0
#define SECCR2_TXG1CIPHER1  1
#define SECCR2_TXG1CIPHER2  2
#define SECCR2_TXG2CIPHER0  3
#define SECCR2_TXG2CIPHER1  4
#define SECCR2_TXG2CIPHER2  5
```

```
#define SECCR2_UPDEC   7
#define SECCR2_UPENC   6
#define SLPACK   0x35
#define SLPACK_SLPACK   7
#define SLPACK_WAKECNT0   0
#define SLPACK_WAKECNT1   1
#define SLPACK_WAKECNT2   2
#define SLPACK_WAKECNT3   3
#define SLPACK_WAKECNT4   4
#define SLPACK_WAKECNT5   5
#define SLPACK_WAKECNT6   6
#define SLPCAL0   0x209
#define SLPCAL0_SLPCAL0   0
#define SLPCAL0_SLPCAL1   1
#define SLPCAL0_SLPCAL2   2
#define SLPCAL0_SLPCAL3   3
#define SLPCAL0_SLPCAL4   4
#define SLPCAL0_SLPCAL5   5
#define SLPCAL0_SLPCAL6   6
#define SLPCAL0_SLPCAL7   7
#define SLPCAL1   0x20A
#define SLPCAL1_SLPCAL10   2
#define SLPCAL1_SLPCAL11   3
#define SLPCAL1_SLPCAL12   4
#define SLPCAL1_SLPCAL13   5
#define SLPCAL1_SLPCAL14   6
#define SLPCAL1_SLPCAL15   7
#define SLPCAL1_SLPCAL8   0
```

```
#define SLPCAL1_SLPCAL9  1
#define SLPCAL2  0x20B
#define SLPCAL2_SLPCAL16  0
#define SLPCAL2_SLPCAL17  1
#define SLPCAL2_SLPCAL18  2
#define SLPCAL2_SLPCAL19  3
#define SLPCAL2_SLPCALEN  4
#define SLPCAL2_SLPCALRDY  7
#define SLPCON0  0x211
#define SLPCON0_INTEDGE  1
#define SLPCON0_SLPCLKEN  0
#define SLPCON1  0x220
#define SLPCON1_CLKOUTEN  5
#define SLPCON1_SLPCLKDIV0  0
#define SLPCON1_SLPCLKDIV1  1
#define SLPCON1_SLPCLKDIV2  2
#define SLPCON1_SLPCLKDIV3  3
#define SLPCON1_SLPCLKDIV4  4
#define SOFTRST  0x2A
#define SOFTRST_RSTBB  1
#define SOFTRST_RSTMAC  0
#define SOFTRST_RSTPWR  2
#define SYMTICKH  0x15
#define SYMTICKH_TICKP8  0
#define SYMTICKH_TXONT0  1
#define SYMTICKH_TXONT1  2
#define SYMTICKH_TXONT2  3
#define SYMTICKH_TXONT3  4
```

```
#define SYMTICKH_TXONT4  5
#define SYMTICKH_TXONT5  6
#define SYMTICKH_TXONT6  7
#define SYMTICKL  0x14
#define SYMTICKL_TICKP0  0
#define SYMTICKL_TICKP1  1
#define SYMTICKL_TICKP2  2
#define SYMTICKL_TICKP3  3
#define SYMTICKL_TICKP4  4
#define SYMTICKL_TICKP5  5
#define SYMTICKL_TICKP6  6
#define SYMTICKL_TICKP7  7
#define TESTMODE  0x22F
#define TESTMODE_RSSIWAIT0  3
#define TESTMODE_RSSIWAIT1  4
#define TESTMODE_TESTMODE0  0
#define TESTMODE_TESTMODE1  1
#define TESTMODE_TESTMODE2  2
#define TRISGPIO  0x34
#define TRISGPIO_TRISGP0  0
#define TRISGPIO_TRISGP1  1
#define TRISGPIO_TRISGP2  2
#define TRISGPIO_TRISGP3  3
#define TRISGPIO_TRISGP4  4
#define TRISGPIO_TRISGP5  5
#define TXBCON0  0x1A
#define TXBCON0_TXBSECEN  1
#define TXBCON0_TXBTRIG  0
```

```
#define TXBCON1  0x25
#define TXG1CON  0x1C
#define TXG1CON_TXG1ACKREQ  2
#define TXG1CON_TXG1RETRY0  6
#define TXG1CON_TXG1RETRY1  7
#define TXG1CON_TXG1SECEN  1
#define TXG1CON_TXG1SLOT0  3
#define TXG1CON_TXG1SLOT1  4
#define TXG1CON_TXG1SLOT2  5
#define TXG1CON_TXG1TRIG  0
#define TXG2CON  0x1D
#define TXG2CON_TXG2ACKREQ  2
#define TXG2CON_TXG2RETRY0  6
#define TXG2CON_TXG2RETRY1  7
#define TXG2CON_TXG2SECEN  1
#define TXG2CON_TXG2SLOT0  3
#define TXG2CON_TXG2SLOT1  4
#define TXG2CON_TXG2SLOT2  5
#define TXG2CON_TXG2TRIG  0
#define TXMCR  0x11
#define TXMCR_BATLIFEXT  6
#define TXMCR_CSMABF0  0
#define TXMCR_CSMABF1  1
#define TXMCR_CSMABF2  2
#define TXMCR_MACMINBE0  3
#define TXMCR_MACMINBE1  4
#define TXMCR_NOCSMA  7
#define TXMCR_SLOTTED  5
```

```
#define TXNCON   0x1B
#define TXNCON_FPSTAT   4
#define TXNCON_INDIRECT   3
#define TXNCON_TXNACKREQ   2
#define TXNCON_TXNSECEN   1
#define TXNCON_TXNTRIG   0
#define TXPEND   0x21
#define TXPEND_FPACK   0
#define TXPEND_GTSSWITCH   1
#define TXPEND_MLIFS0   2
#define TXPEND_MLIFS1   3
#define TXPEND_MLIFS2   4
#define TXPEND_MLIFS3   5
#define TXPEND_MLIFS4   6
#define TXPEND_MLIFS5   7
#define TXSTAT   0x24
#define TXSTAT_CCAFAIL   5
#define TXSTAT_TXG1FNT   3
#define TXSTAT_TXG1STAT   1
#define TXSTAT_TXG2FNT   4
#define TXSTAT_TXG2STAT   2
#define TXSTAT_TXNRETRY0   6
#define TXSTAT_TXNRETRY1   7
#define TXSTAT_TXNSTAT   0
#define TXSTBL   0x2E
#define TXSTBL_MSIFS0   0
#define TXSTBL_MSIFS1   1
#define TXSTBL_MSIFS2   2
```

#define TXSTBL_MSIFS3 3

#define TXSTBL_RFSTBL0 4

#define TXSTBL_RFSTBL1 5

#define TXSTBL_RFSTBL2 6

#define TXSTBL_RFSTBL3 7

#define TXTIME 0x27

#define TXTIME_TURNTIME0 4

#define TXTIME_TURNTIME1 5

#define TXTIME_TURNTIME2 6

#define TXTIME_TURNTIME3 7

#define UPNONCE0 0x240

#define UPNONCE1 0x241

#define UPNONCE10 0x24A

#define UPNONCE11 0x24B

#define UPNONCE12 0x24C

#define UPNONCE2 0x242

#define UPNONCE3 0x243

#define UPNONCE4 0x244

#define UPNONCE5 0x245

#define UPNONCE6 0x246

#define UPNONCE7 0x247

#define UPNONCE8 0x248

#define UPNONCE9 0x249

#define WAKECON 0x22

#define WAKECON_IMMWAKE 7

#define WAKECON_REGWAKE 6

#define WAKETIMEH 0x223

#define WAKETIMEH_WAKETIME10 2

#define WAKETIMEH_WAKETIME8 0

#define WAKETIMEH_WAKETIME9 1

#define WAKETIMEL 0x222

#define WAKETIMEL_WAKETIME0 0

#define WAKETIMEL_WAKETIME1 1

#define WAKETIMEL_WAKETIME2 2

#define WAKETIMEL_WAKETIME3 3

#define WAKETIMEL_WAKETIME4 4

#define WAKETIMEL_WAKETIME5 5

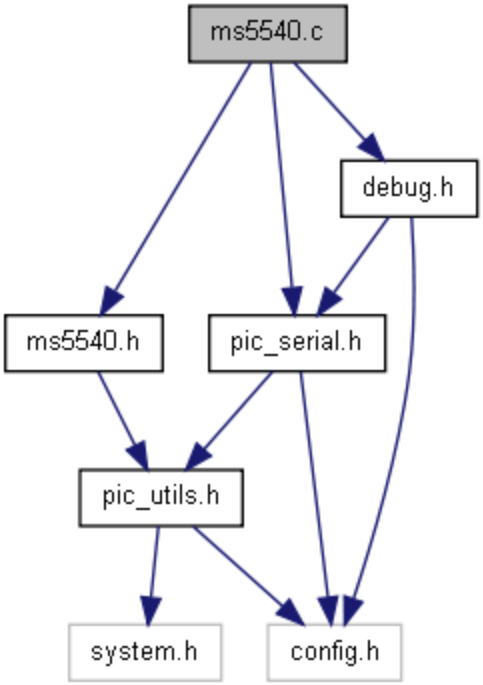
#define WAKETIMEL_WAKETIME6 6

#define WAKETIMEL_WAKETIME7 7

---

ms5540.c File Reference

Include dependency graph for ms5540.c:



Defines

- #define MS5540_DELAY_AMOUNT 2

Functions

- void ms5540_calc_temp_and_pressure ()
- uns16 ms5540_get_config (uns8 config_word)
- uns16 ms5540_get_raw_pressure ()
- uns16 ms5540_get_raw_temp ()
- void ms5540_init ()
- void ms5540_pulse_sclk ()
- void ms5540_reset ()
- void ms5540_send_start ()
- void ms5540_send_stop ()
- void ms5540_setup_io (void)

Variables

- int16 c1
- int16 c2
- int16 c3
- int16 c4
- int16 c5
- int16 c6

Define Documentation

#define MS5540_DELAY_AMOUNT 2

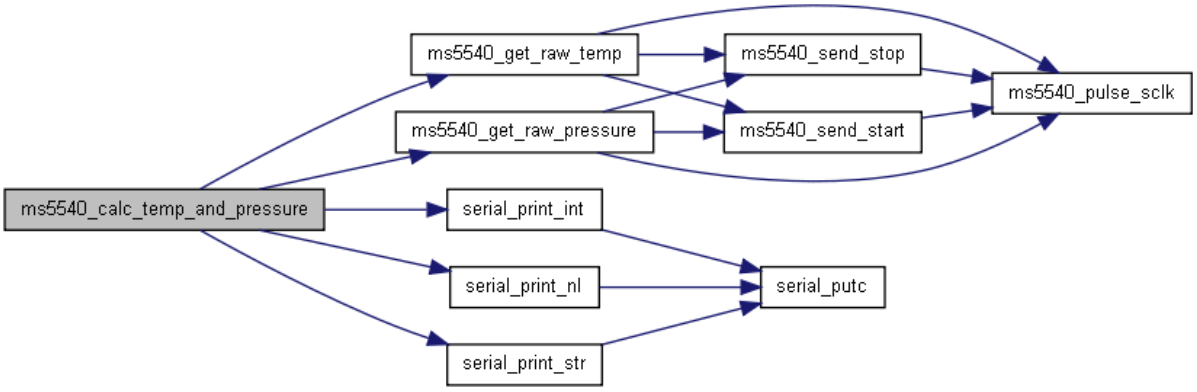
Function Documentation

void ms5540_calc_temp_and_pressure ()

```
322                                         {
323
324 int16 d1, d2, ut1, off;
325 int16 temp, p, dt;
326 int32 sens, x;
327
328 //  c1 = 23470;
329 //  c2 = 1324;
330 //  c3 = 737;
331 //  c4 = 393;
332 //  c5 = 628;
333 //  c6 = 25;
334
335
336     d1 = ms5540_get_raw_pressure();
337 //  d1 = 16460;
338
339     serial_print_str("get pr d1: ");
340     serial_print_int(d1);
341     serial_print_nl();
342
343     d2 = ms5540_get_raw_temp();
344 //  d2 = 27856;
345     serial_print_str("get temp: ");
```

```
346     serial_print_int(d2);
347     serial_print_nl();
348 
349     ut1 = 8*c5+20224;
350 
351     serial_print_str("ut1= ");
352     serial_print_int(ut1);
353     serial_print_nl();
354 
355     dt = d2 - ut1;
356 
357     serial_print_str("dt= ");
358     serial_print_int(dt);
359     serial_print_nl();
360 
361     temp = 200 + (int32)dt * ((int32)c6 + 50) / 1024;
362 
363     serial_print_str("temp= ");
364     serial_print_int(temp);
365     serial_print_nl();
366 
367     off = c2 * 4 + (((int32)c4-512)*dt) / 4096;
368 
369     serial_print_str("off= ");
370     serial_print_int(off);
371     serial_print_nl();
372 
373     sens = c1 + ((int32)c3*dt)/1024 + 24576;
374 
375     serial_print_str("sens= ");
376     serial_print_int(sens);
377     serial_print_nl();
378 
379     x = ((int32)sens * ((int32)d1-7168))/16384 - (int32)off;
380 
381     serial_print_str("x= ");
382     serial_print_int(x);
383     serial_print_nl();
384 
385     p = x * 10/32 + 250*10;
386 
387     serial_print_str("p= ");
388     serial_print_int(p);
389     serial_print_nl();
390 }
```

Here is the call graph for this function:

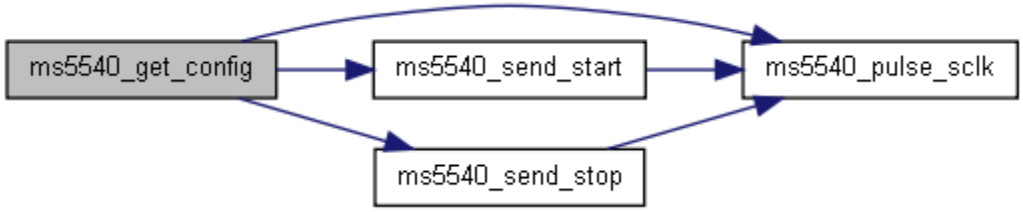


uns16 ms5540_get_config (uns8 *config_word*)

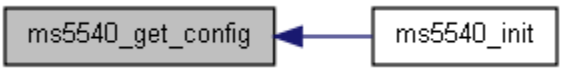
```
102                                             {
```

```
103
104 uns8 request;
105 uns8 count;
106 uns16 result;
107
108     switch (config_word) {
109         case 1:
110             request = 0b01010100;
111             break;
112         case 2:
113             request = 0b01011000;
114             break;
115         case 3:
116             request = 0b01100100;
117             break;
118         case 4:
119             request = 0b01101000;
120             break;
121     }
122     ms5540_send_start();
123
124     for( count = 0 ; count < 6 ; count++ )  // 1- 17 to get correct bit pattern
125     {
126         change_pin(ms5540_din_port, ms5540_din_pin, request.7);
127
128         request = request << 1;
129
130         ms5540_pulse_sclk();
131     }
132
133     ms5540_send_stop();
134
135     // naked clock pulse
136     ms5540_pulse_sclk();
137
138     // okay now clock the word out
139     for(count = 0 ; count < 16 ; count++)
140     {
141         ms5540_pulse_sclk();
142
143         result = result << 1;
144         result.0 = test_pin(ms5540_dout_port, ms5540_dout_pin);
145     }
146
147     // naked clock pulse
148     ms5540_pulse_sclk();
149
150     return result;
151
152 }
```

Here is the call graph for this function:



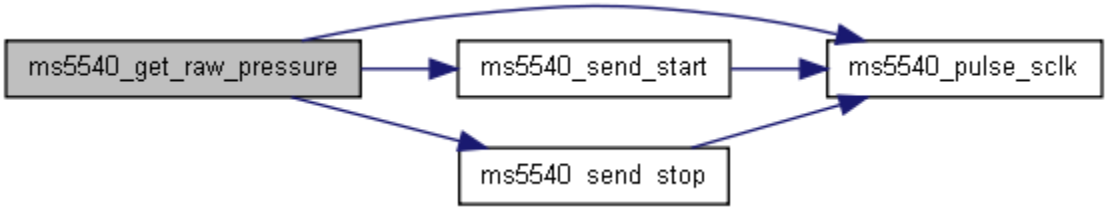
Here is the caller graph for this function:



uns16 ms5540_get_raw_pressure ()

```
154                                 {
155
156 uns8 setup_word = 0b1010;
157 uns8 count;
158 uns16 result;
159
160     ms5540_send_start();
161     // 1
162     set_pin(ms5540_din_port, ms5540_din_pin);
163     ms5540_pulse_sclk();
164     // 0
165     clear_pin(ms5540_din_port, ms5540_din_pin);
166     ms5540_pulse_sclk();
167     // 1
168     set_pin(ms5540_din_port, ms5540_din_pin);
169     ms5540_pulse_sclk();
170     // 0
171     clear_pin(ms5540_din_port, ms5540_din_pin);
172     ms5540_pulse_sclk();
173
174     ms5540_send_stop();
175
176     // two more pulses
177     ms5540_pulse_sclk();
178     ms5540_pulse_sclk();
179
180     // now wait for conversion to finish (din goes low)
181
182     while (test_pin(ms5540_dout_port, ms5540_dout_pin) != 0) {
183     }
184
185     // okay now clock the word out
186     for(count = 0 ; count < 16 ; count++)
187     {
188         ms5540_pulse_sclk();
189
190         result = result << 1;
191         result.0 = test_pin(ms5540_dout_port, ms5540_dout_pin);
192     }
193
194     // naked pulse
195     ms5540_pulse_sclk();
196     return result;
197
198 }
```

Here is the call graph for this function:

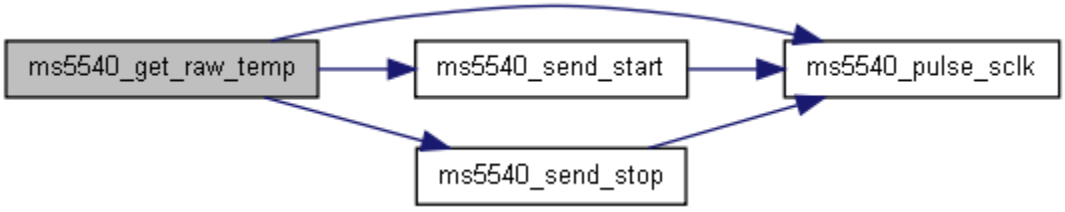


Here is the caller graph for this function:

uns16 ms5540_get_raw_temp ()

```
200                             {
201
202 uns8 setup_word = 0b1010;
203 uns8 count;
204 uns16 result;
205
206
207     ms5540_send_start();
208     // 1
209     set_pin(ms5540_din_port, ms5540_din_pin);
210     ms5540_pulse_sclk();
211     // 0
212     clear_pin(ms5540_din_port, ms5540_din_pin);
213     ms5540_pulse_sclk();
214     // 0
215     ms5540_pulse_sclk();
216     // 1
217     set_pin(ms5540_din_port, ms5540_din_pin);
218     ms5540_pulse_sclk();
219
220     ms5540_send_stop();
221
222     // two more pulses
223     ms5540_pulse_sclk();
224     ms5540_pulse_sclk();
225
226     // now wait for conversion to finish (din goes low)
227
228     while (test_pin(ms5540_dout_port, ms5540_dout_pin) != 0) {
229     }
230
231     // okay now clock the word out
232     for(count = 0 ; count < 16 ; count++)
233     {
234         ms5540_pulse_sclk();
235
236         result = result << 1;
237         result.0 = test_pin(ms5540_dout_port, ms5540_dout_pin);
238     }
239
240     // naked pulse
241     ms5540_pulse_sclk();
242
243     return result;
244
245 }
```

Here is the call graph for this function:



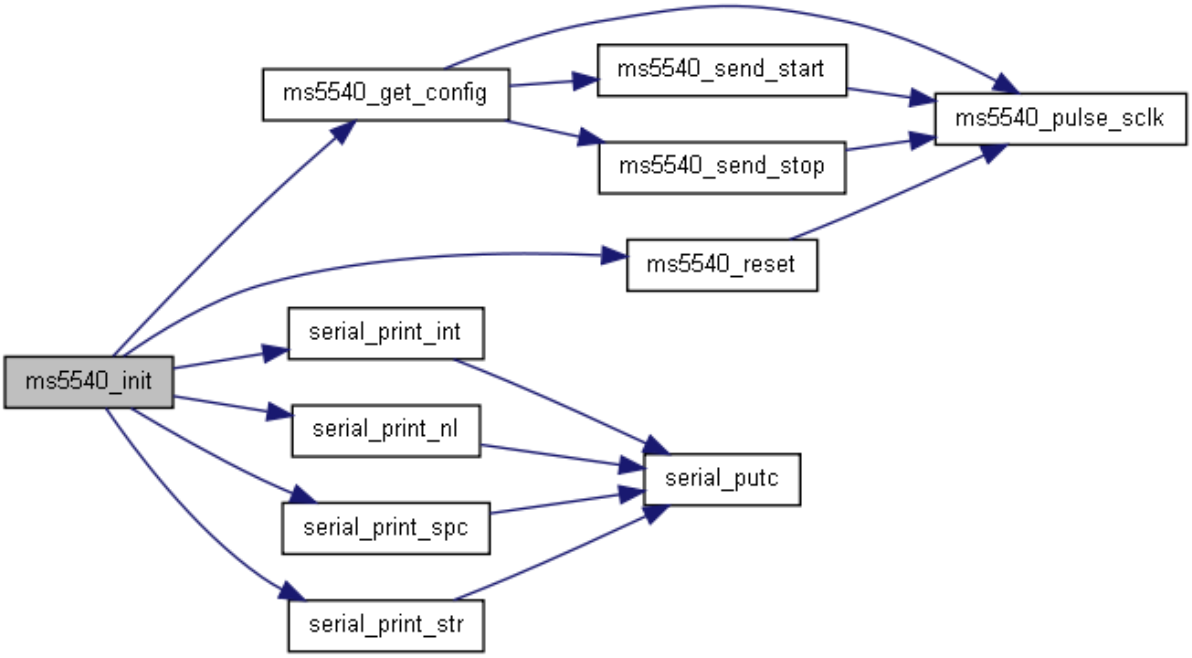
Here is the caller graph for this function:

void ms5540_init ()

```
247                     {
248
249 uns16 word1, word2, word3, word4, x, y;
250 /*c1=22801
251 c2=2256
252 c3=716
253 c4=692
254 c5=872
255 c6=26
256
257 */
258     // reset
259     ms5540_reset();
260
261     // get config
262     word1 = ms5540_get_config(1);
263     serial_print_int(word1);
264     serial_print_spc();
265     word2 = ms5540_get_config(2);
266     serial_print_int(word2);
267     serial_print_spc();
268     word3 = ms5540_get_config(3);
269     serial_print_int(word3);
270     serial_print_spc();
271     word4 = ms5540_get_config(4);
272     serial_print_int(word4);
273     serial_print_nl();
274
275     // parse config words
276
277     c1 = (word1 >> 1) & 0x7fff;
278
279     /*c5 = word2 >> 6;
280     if (test_bit(word1, 0)) {
281         set_bit(c5, 10);
282     }
283     */
284     x = (word1 << 10) & 0x0400;
285     y = (word2 >> 6)  & 0x03ff;
286     c5 = x | y;
287
288     c6 = word2 & 0b0000000000111111; // grab last 6 bits of word2 as c6
289
290     c4 = (word3 >> 6) & 0x03ff;
291
292     x = (word3 << 6) &  0x0fc0;
293     y = (word4 & 0b0000000000111111);
294     c2 = x | y;
295
296     c3 = (word4 >> 6) & 0x03ff;
297
298     serial_print_str("c1=");
299     serial_print_int(c1);
300     serial_print_nl();
301     serial_print_str("c2=");
302     serial_print_int(c2);
303     serial_print_nl();
304     serial_print_str("c3=");
305     serial_print_int(c3);
306     serial_print_nl();
307     serial_print_str("c4=");
308     serial_print_int(c4);
309     serial_print_nl();
310     serial_print_str("c5=");
311     serial_print_int(c5);
312     serial_print_nl();
313     serial_print_str("c6=");
314     serial_print_int(c6);
```

```
315     serial_print_nl();
316
317
318 }
```

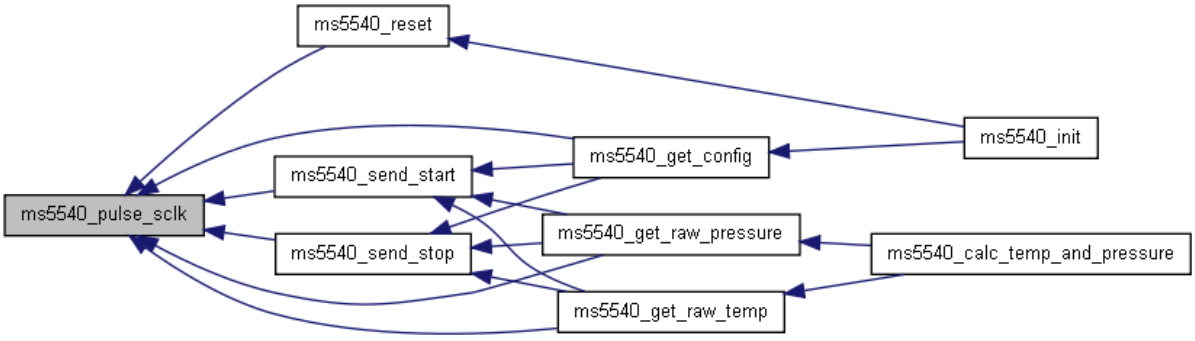
Here is the call graph for this function:



void ms5540_pulse_sclk ()

```
54                          {
55
56     set_pin(ms5540_sclk_port, ms5540_sclk_pin);
57     delay_us(MS5540_DELAY_AMOUNT);
58     clear_pin(ms5540_sclk_port, ms5540_sclk_pin);
59     delay_us(MS5540_DELAY_AMOUNT);
60
61 }
```

Here is the caller graph for this function:



void ms5540_reset ()

```
80                      {
81
82 uns16 reset word = 0b1010101010101010;
83 uns8 count;
84
85     for( count = 1 ; count < 17 ; count++ ) // 1- 17 to get correct bit pattern
86     {
87         change pin(ms5540_din_port, ms5540_din_pin, count.0);
88         ms5540 pulse sclk();
89     }
90     // five more pulses
91     ms5540 pulse sclk();
92     ms5540 pulse sclk();
93     ms5540 pulse sclk();
94     ms5540 pulse sclk();
95     ms5540 pulse sclk();
96
97
98 }
```

Here is the call graph for this function:

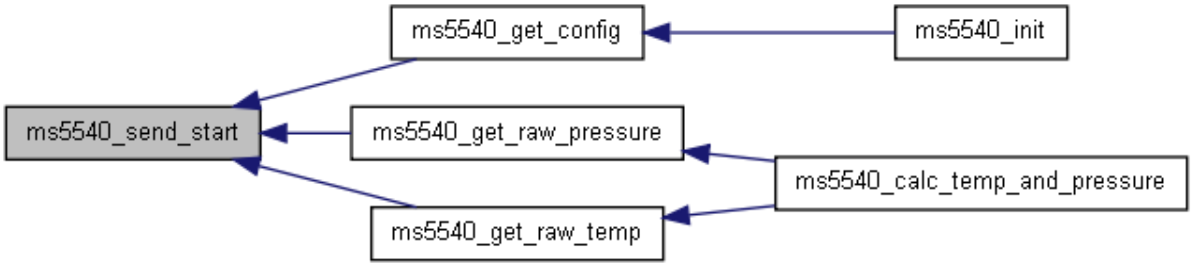
Here is the caller graph for this function:

void ms5540_send_start ()

```
63                          {
64
65     set pin(ms5540_din_port, ms5540_din_pin);
66     ms5540 pulse sclk();
67     ms5540 pulse sclk();
68     ms5540 pulse sclk();
69 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

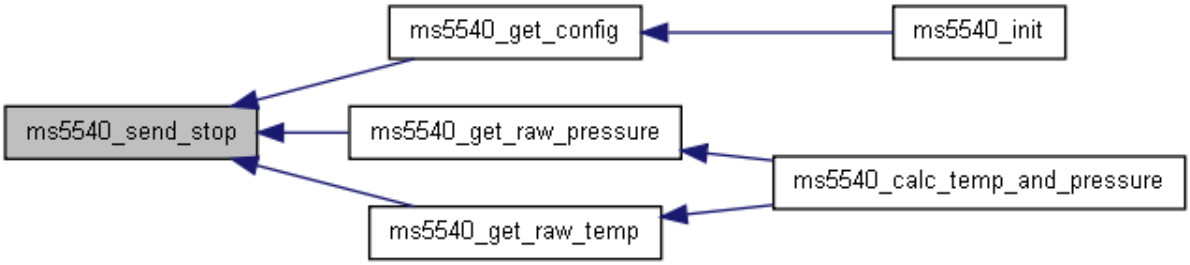


void ms5540_send_stop ()

```
71                          {
72
73      clear_pin(ms5540_din_port, ms5540_din_pin);
74      ms5540_pulse_sclk();
75      ms5540_pulse_sclk();
76      ms5540_pulse_sclk();
77  }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void ms5540_setup_io (void)

```
46                              {
47
48      clear_pin(ms5540_sclk_port, ms5540_sclk_pin);
49      make_output(ms5540_sclk_port, ms5540_sclk_pin);
50      make_output(ms5540_din_port, ms5540_din_pin);
51      make_input(ms5540_dout_port, ms5540_dout_pin);
52  }
```

Variable Documentation

int16 c1

int16 c2

int16 c3

int16 c4

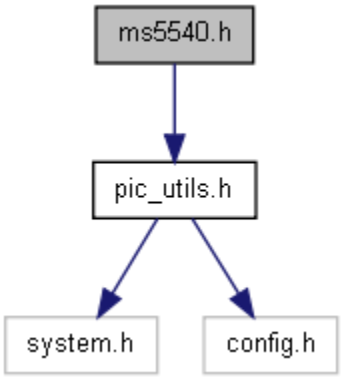
int16 c5

int16 c6

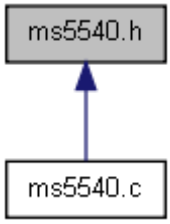
---

ms5540.h File Reference

MS5540 temperature and pressure sensor routines.
Include dependency graph for ms5540.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define ms5540_setup() ms5540_setup_io()

Setup ms5540 ports and pins. Functions

- void ms5540_calc_temp_and_pressure ()
- uns16 ms5540_get_config (uns8 config_word)
- uns16 ms5540_get_raw_pressure ()
- uns16 ms5540_get_raw_temp ()
- void ms5540_init ()
- void ms5540_reset ()
- void ms5540_setup_io (void)

Detailed Description

A library to communicate with the ms5540 sensor

Define Documentation

#define ms5540_setup() ms5540_setup_io()

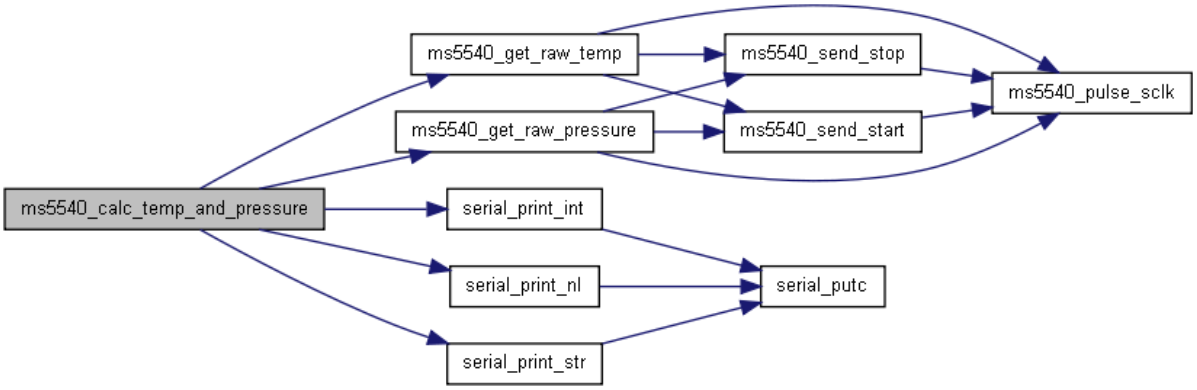
Function Documentation

void ms5540_calc_temp_and_pressure ()

```
322                                     {
323
324 int16 d1, d2, ut1, off;
325 int16 temp, p, dt;
326 int32 sens, x;
327
328 //  c1 = 23470;
329 //  c2 = 1324;
330 //  c3 = 737;
331 //  c4 = 393;
332 //  c5 = 628;
333 //  c6 = 25;
334
335
336     d1 = ms5540_get_raw_pressure();
337 //  d1 = 16460;
338
339     serial_print_str("get pr d1: ");
340     serial_print_int(d1);
341     serial_print_nl();
342
343     d2 = ms5540_get_raw_temp();
344 //  d2 = 27856;
345     serial_print_str("get temp: ");
346     serial_print_int(d2);
347     serial_print_nl();
348
349     ut1 = 8*c5+20224;
350
351     serial_print_str("ut1= ");
352     serial_print_int(ut1);
353     serial_print_nl();
354
355     dt = d2 - ut1;
356
357     serial_print_str("dt= ");
358     serial_print_int(dt);
359     serial_print_nl();
360
361     temp = 200 + (int32)dt * ((int32)c6 + 50) / 1024;
362
363     serial_print_str("temp= ");
364     serial_print_int(temp);
365     serial_print_nl();
366
367     off = c2 * 4 + (((int32)c4-512)*dt) / 4096;
368
369     serial_print_str("off= ");
370     serial_print_int(off);
371     serial_print_nl();
372
373     sens = c1 + ((int32)c3*dt)/1024 + 24576;
374
375     serial_print_str("sens= ");
376     serial_print_int(sens);
377     serial_print_nl();
378
379     x = ((int32)sens * ((int32)d1-7168))/16384 - (int32)off;
380
381     serial_print_str("x= ");
382     serial_print_int(x);
383     serial_print_nl();
384
385     p = x * 10/32 + 250*10;
386
```

```
387     serial_print_str("p= ");
388     serial_print_int(p);
389     serial_print_nl();
390 }
```

Here is the call graph for this function:

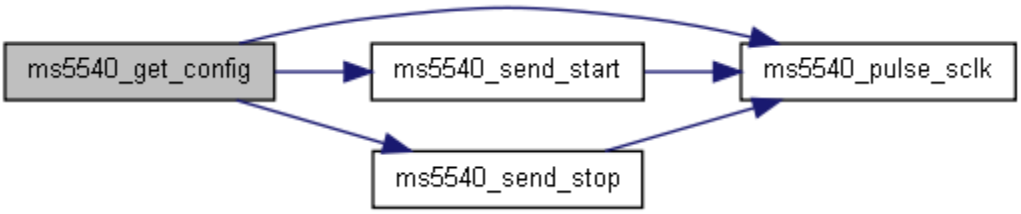


uns16 ms5540_get_config (uns8 *config_word*)

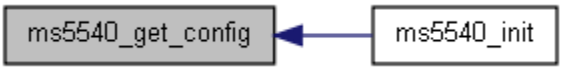
```
102                                             {
103 
104 uns8 request;
105 uns8 count;
106 uns16 result;
107 
108     switch (config_word) {
109         case 1:
110             request = 0b01010100;
111             break;
112         case 2:
113             request = 0b01011000;
114             break;
115         case 3:
116             request = 0b01100100;
117             break;
118         case 4:
119             request = 0b01101000;
120             break;
121     }
122     ms5540_send_start();
123 
124     for( count = 0 ; count < 6 ; count++ )  // 1- 17 to get correct bit pattern
125     {
126         change_pin(ms5540_din_port, ms5540_din_pin, request.7);
127 
128         request = request << 1;
129 
130         ms5540_pulse_sclk();
131     }
132 
133     ms5540_send_stop();
134 
135     // naked clock pulse
136     ms5540_pulse_sclk();
137 
138     // okay now clock the word out
139     for(count = 0 ; count < 16 ; count++)
140     {
141         ms5540_pulse_sclk();
142 
143         result = result << 1;
```

```
144         result.0 = test_pin(ms5540_dout_port, ms5540_dout_pin);
145     }
146 
147     // naked clock pulse
148     ms5540_pulse_sclk();
149 
150     return result;
151 
152 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

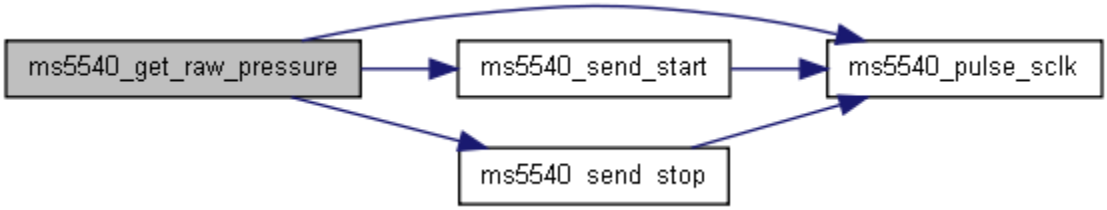


uns16 ms5540_get_raw_pressure ()

```
154                                 {
155 
156 uns8 setup_word = 0b1010;
157 uns8 count;
158 uns16 result;
159 
160     ms5540_send_start();
161     // 1
162     set_pin(ms5540_din_port, ms5540_din_pin);
163     ms5540_pulse_sclk();
164     // 0
165     clear_pin(ms5540_din_port, ms5540_din_pin);
166     ms5540_pulse_sclk();
167     // 1
168     set_pin(ms5540_din_port, ms5540_din_pin);
169     ms5540_pulse_sclk();
170     // 0
171     clear_pin(ms5540_din_port, ms5540_din_pin);
172     ms5540_pulse_sclk();
173 
174     ms5540_send_stop();
175 
176     // two more pulses
177     ms5540_pulse_sclk();
178     ms5540_pulse_sclk();
179 
180     // now wait for conversion to finish (din goes low)
181 
182     while (test_pin(ms5540_dout_port, ms5540_dout_pin) != 0) {
183     }
184 
185     // okay now clock the word out
186     for(count = 0 ; count < 16 ; count++)
187     {
188         ms5540_pulse_sclk();
189 
190         result = result << 1;
191         result.0 = test_pin(ms5540_dout_port, ms5540_dout_pin);
```

```
192     }
193
194     // naked pulse
195     ms5540_pulse_sclk();
196     return result;
197
198 }
```

Here is the call graph for this function:



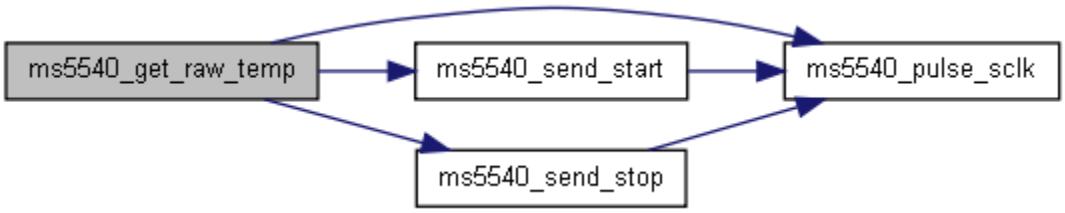
Here is the caller graph for this function:

uns16 ms5540_get_raw_temp ()

```
200                             {
201
202 uns8 setup_word = 0b1010;
203 uns8 count;
204 uns16 result;
205
206
207     ms5540_send_start();
208     // 1
209     set_pin(ms5540_din_port, ms5540_din_pin);
210     ms5540_pulse_sclk();
211     // 0
212     clear_pin(ms5540_din_port, ms5540_din_pin);
213     ms5540_pulse_sclk();
214     // 0
215     ms5540_pulse_sclk();
216     // 1
217     set_pin(ms5540_din_port, ms5540_din_pin);
218     ms5540_pulse_sclk();
219
220     ms5540_send_stop();
221
222     // two more pulses
223     ms5540_pulse_sclk();
224     ms5540_pulse_sclk();
225
226     // now wait for conversion to finish (din goes low)
227
228     while (test_pin(ms5540_dout_port, ms5540_dout_pin) != 0) {
229     }
230
231     // okay now clock the word out
232     for(count = 0 ; count < 16 ; count++)
233     {
234         ms5540_pulse_sclk();
235
236         result = result << 1;
237         result.0 = test_pin(ms5540_dout_port, ms5540_dout_pin);
238     }
239
```

```
240     // naked pulse
241     ms5540_pulse_sclk();
242 
243     return result;
244 
245 }
```

Here is the call graph for this function:



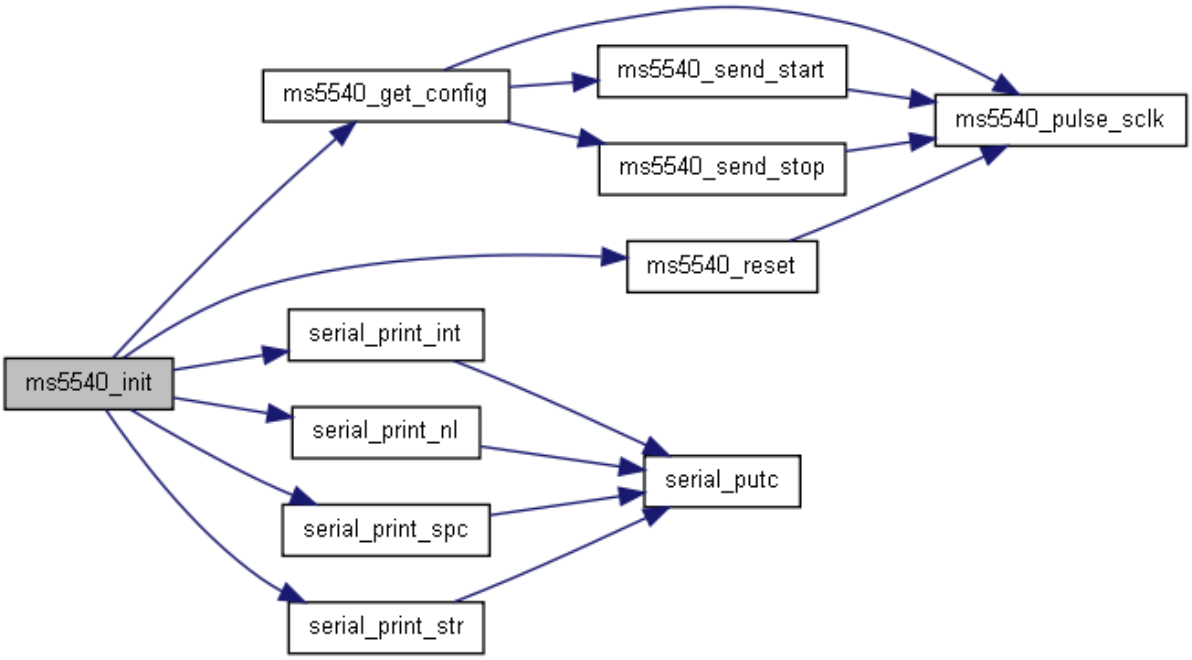
Here is the caller graph for this function:

void ms5540_init ()

```
247                     {
248 
249 uns16 word1, word2, word3, word4, x, y;
250 /*c1=22801
251 c2=2256
252 c3=716
253 c4=692
254 c5=872
255 c6=26
256 
257 */
258     // reset
259     ms5540_reset();
260 
261     // get config
262     word1 = ms5540_get_config(1);
263     serial_print_int(word1);
264     serial_print_spc();
265     word2 = ms5540_get_config(2);
266     serial_print_int(word2);
267     serial_print_spc();
268     word3 = ms5540_get_config(3);
269     serial_print_int(word3);
270     serial_print_spc();
271     word4 = ms5540_get_config(4);
272     serial_print_int(word4);
273     serial_print_nl();
274 
275     // parse config words
276 
277     c1 = (word1 >> 1) & 0x7fff;
278 
279     /*c5 = word2 >> 6;
280     if (test_bit(word1, 0)) {
281         set_bit(c5, 10);
282     }
283     */
284     x = (word1 << 10) & 0x0400;
285     y = (word2 >> 6)  & 0x03ff;
286     c5 = x | y;
287 
```

```
288     c6 = word2 & 0b0000000000111111; // grab last 6 bits of word2 as c6
289
290     c4 = (word3 >> 6) & 0x03ff;
291
292     x = (word3 << 6) &  0x0fc0;
293     y = (word4 & 0b0000000000111111);
294     c2 = x | y;
295
296     c3 = (word4 >> 6) & 0x03ff;
297
298     serial_print_str("c1=");
299     serial_print_int(c1);
300     serial_print_nl();
301     serial_print_str("c2=");
302     serial_print_int(c2);
303     serial_print_nl();
304     serial_print_str("c3=");
305     serial_print_int(c3);
306     serial_print_nl();
307     serial_print_str("c4=");
308     serial_print_int(c4);
309     serial_print_nl();
310     serial_print_str("c5=");
311     serial_print_int(c5);
312     serial_print_nl();
313     serial_print_str("c6=");
314     serial_print_int(c6);
315     serial_print_nl();
316
317
318 }
```

Here is the call graph for this function:

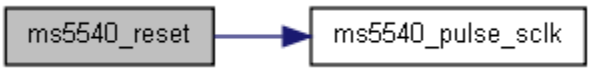


void ms5540_reset ()

```
80                      {
81
82 uns16 reset_word = 0b1010101010101010;
```

```
83 uns8 count;
84
85     for( count = 1 ; count < 17 ; count++ ) // 1- 17 to get correct bit pattern
86     {
87         change_pin(ms5540_din_port, ms5540_din_pin, count.0);
88         ms5540_pulse_sclk();
89     }
90     // five more pulses
91     ms5540_pulse_sclk();
92     ms5540_pulse_sclk();
93     ms5540_pulse_sclk();
94     ms5540_pulse_sclk();
95     ms5540_pulse_sclk();
96
97
98 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

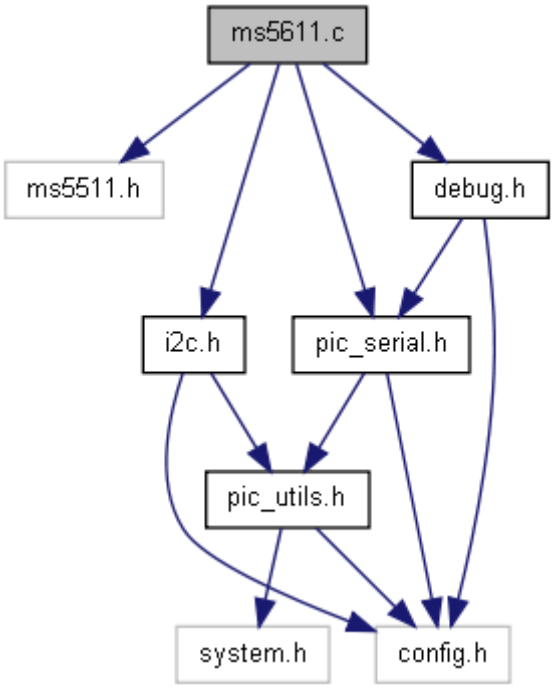
void ms5540_setup_io (void)

```
46                              {
47
48     clear_pin(ms5540_sclk_port, ms5540_sclk_pin);
49     make_output(ms5540_sclk_port, ms5540_sclk_pin);
50     make_output(ms5540_din_port, ms5540_din_pin);
51     make_input(ms5540_dout_port, ms5540_dout_pin);
52 }
```

---

ms5611.c File Reference

Include dependency graph for ms5611.c:



Defines

- #define MS5511_PROM_READ 0xa0
- #define MS5511_RESET 0x1e

Functions

- uns8 ms5511_calc_crc (uns8 n_prom[])
- void ms5511_calc_temp_and_pressure ()
- uns16 ms5511_get_config (uns8 config_word)
- uns32 ms5511_get_raw_pressure ()
- uns32 ms5511_get_raw_temp ()
- void ms5511_init ()
- void ms5511_reset ()
- void ms5511_setup_io (void)

Variables

- int16 c1
- int16 c2
- int16 c3
- int16 c4
- int16 c5
- int16 c6

Define Documentation

#define MS5511_PROM_READ 0xa0

#define MS5511_RESET 0x1e

Function Documentation

uns8 ms5511_calc_crc (uns8 *n_prom*[])

```
164 {
165     int cnt; // simple counter
166     uns16 n_rem; // crc reminder
167     uns16 crc_read; // original value of the crc
168     uns8 n_bit;
169 
170     n_rem = 0x00;
171     crc read = n prom[7]; //save read CRC
172     n_prom[7] = (0xFF00 & (n_prom[7])); //CRC byte is replaced by 0
173 
174     for (cnt = 0; cnt < 16; cnt++) { // operation is performed on bytes
175 
176          // choose LSB or MSB
177         if (cnt%2==1) n_rem ^= (unsigned short) ((n_prom[cnt>>1]) & 0x00FF);
178         else n_rem ^= (unsigned short) (n_prom[cnt>>1]>>8);
179 
180         for (n_bit = 8; n_bit > 0; n_bit--) {
181 
182             if (n_rem & (0x8000)) {
183                 n_rem = (n rem << 1) ^ 0x3000;
184             } else {
185                 n_rem = (n_rem << 1);
186             }
187         }
188     }
189     n_rem = (0x000F & (n_rem >> 12)); // // final 4-bit reminder is CRC code
190     n_prom[7] = crc_read; // restore the crc_read to its original place
191 
192     return (n_rem ^ 0x00);
193 
194 }
```

void ms5511_calc_temp_and_pressure ()

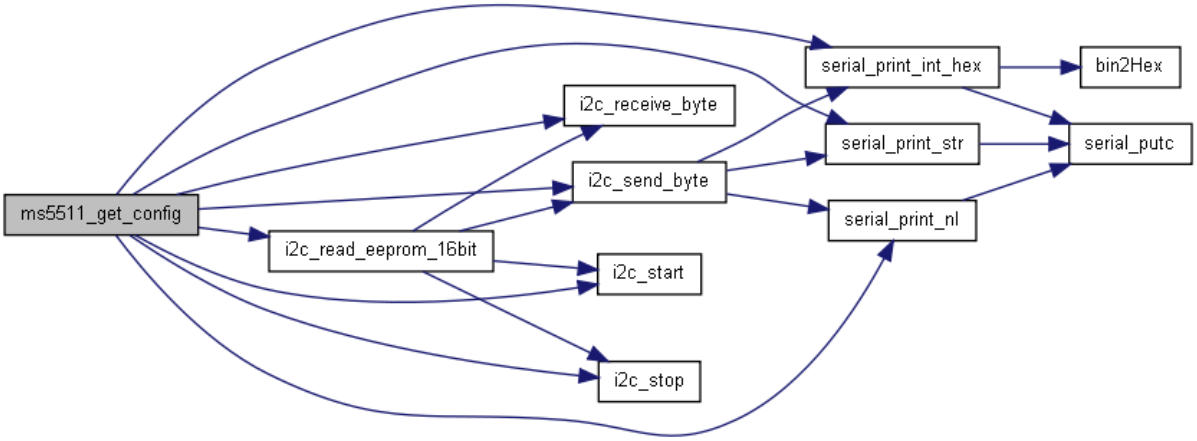
```
160                                     {
161 }
```

uns16 ms5511_get_config (uns8 *config_word*)

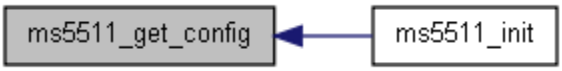
```
67                                          {
68 
69 return i2c_read_eeprom_16bit(MS5511_ADDR << 1, MS5511_PROM_READ + (config_word << 1));
70 
71 uns8 r1, r2;
72 
73     i2c_start();
74     i2c_send_byte(MS5511 ADDR << 1);
75     i2c_send_byte(MS5511_PROM_READ + (config_word << 1));
76     i2c_stop();
```

```
77
78      i2c_start();
79      i2c_send_byte((MS5511_ADDR << 1) + 1);   // read
80
81      r1 = i2c_receive_byte();
82      serial_print_str("r1=");
83      serial_print_int_hex(r1);
84      serial_print_nl();
85
86      r2 = i2c_receive_byte();
87      serial_print_str("r2=");
88      serial_print_int_hex(r2);
89      serial_print_nl();
90
91      return (r1 << 8) + r2;
92
93
94  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



uns32 ms5511_get_raw_pressure ()

```
96                                  {
97
98  uns32 result;
99
100     return result;
101
102 }
```

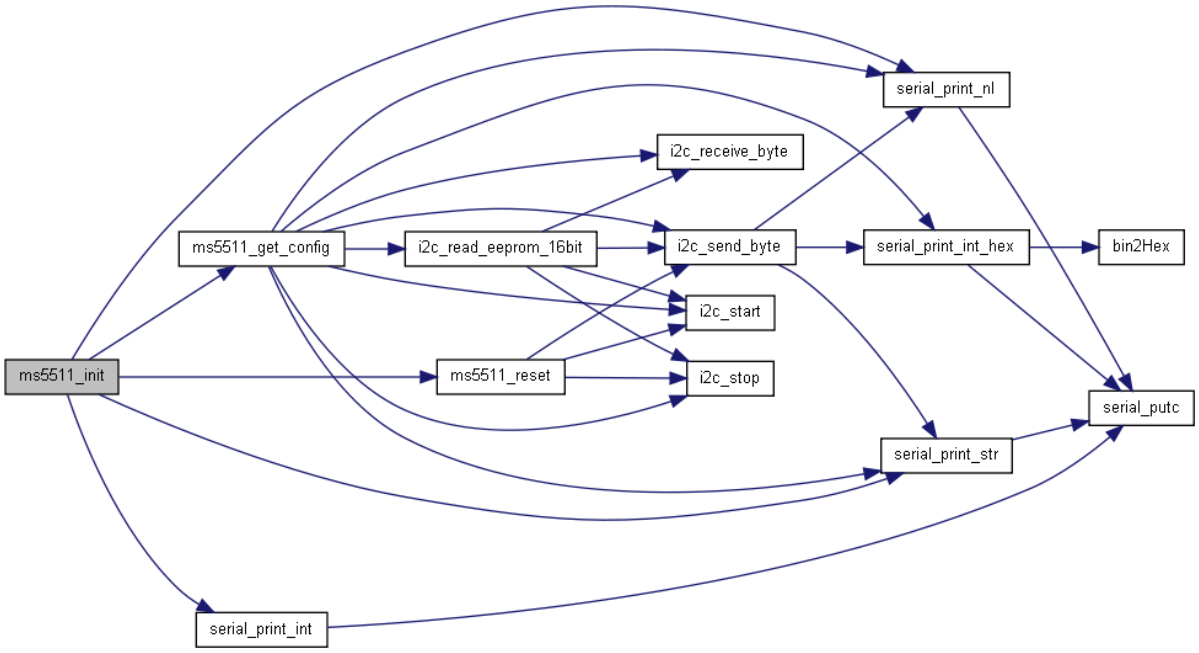
uns32 ms5511_get_raw_temp ()

```
104                                 {
105
106 }
```

void ms5511_init ()

```
108                     {
109 uns16 c0, c7;
110     // reset
111     ms5511_reset();
112 
113     // get config
114     c0 = ms5511_get_config(0);
115 
116     serial_print_str("C1\n");
117     c1 = ms5511_get_config(1);
118     serial_print_str("C2\n");
119     c2 = ms5511_get_config(2);
120     serial_print_str("C3\n");
121     c3 = ms5511_get_config(3);
122     serial_print_str("C4\n");
123     c4 = ms5511_get_config(4);
124     serial_print_str("C5\n");
125     c5 = ms5511_get_config(5);
126     serial_print_str("C6    \n");
127     c6 = ms5511_get_config(6);
128     c7 = ms5511_get_config(7);
129 
130     serial_print_str("c0=");
131     serial_print_int(c0);
132     serial_print_nl();
133     serial_print_str("c1=");
134     serial_print_int(c1);
135     serial_print_nl();
136     serial_print_str("c2=");
137     serial_print_int(c2);
138     serial_print_nl();
139     serial_print_str("c3=");
140     serial_print_int(c3);
141     serial_print_nl();
142     serial_print_str("c4=");
143     serial_print_int(c4);
144     serial_print_nl();
145     serial_print_str("c5=");
146     serial_print_int(c5);
147     serial_print_nl();
148     serial_print_str("c6=");
149     serial_print_int(c6);
150     serial_print_nl();
151     serial_print_str("c7=");
152     serial_print_int(c7);
153     serial_print_nl();
154 
155 
156 }
```

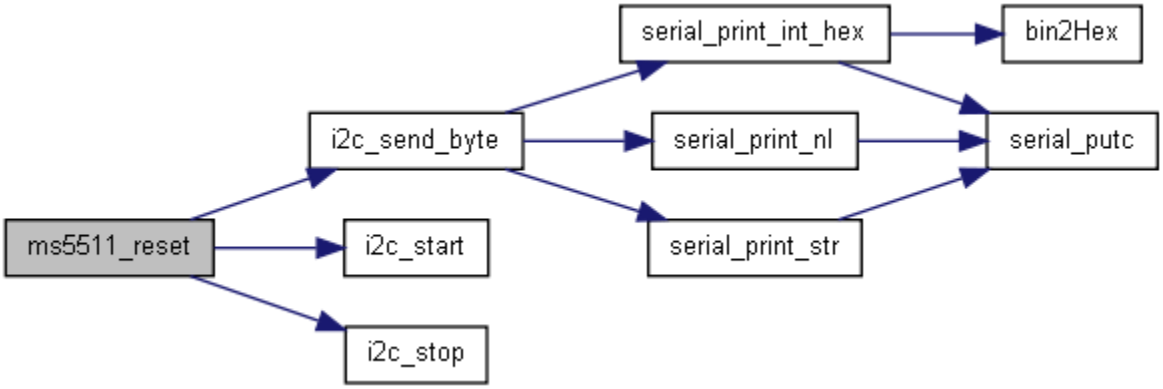
Here is the call graph for this function:



void ms5511_reset ()

```
56                      {
57
58      i2c_start();
59      i2c_send_byte(MS5511_ADDR << 1);
60      i2c_send_byte(MS5511_RESET);
61      i2c_stop();
62
63  }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void ms5511_setup_io (void)

```
48                              {
49
50      i2c_setup_io();
51
52  }
```

Here is the call graph for this function:

Variable Documentation

int16 c1

int16 c2

int16 c3

int16 c4

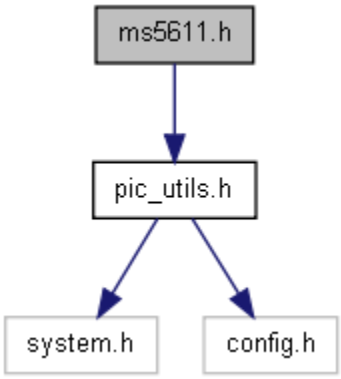
int16 c5

int16 c6

---

ms5611.h File Reference

Include dependency graph for ms5611.h:



Defines

- #define ms5511_setup() ms5511_setup_io()

Setup ms5511 ports and pins. Functions

- void ms5511_calc_temp_and_pressure ()
- uns16 ms5511_get_config (uns8 config_word)
- uns32 ms5511_get_raw_pressure ()
- uns32 ms5511_get_raw_temp ()
- void ms5511_init ()
- void ms5511_reset ()
- void ms5511_setup_io (void)

Define Documentation

#define ms5511_setup() ms5511_setup_io()

Function Documentation

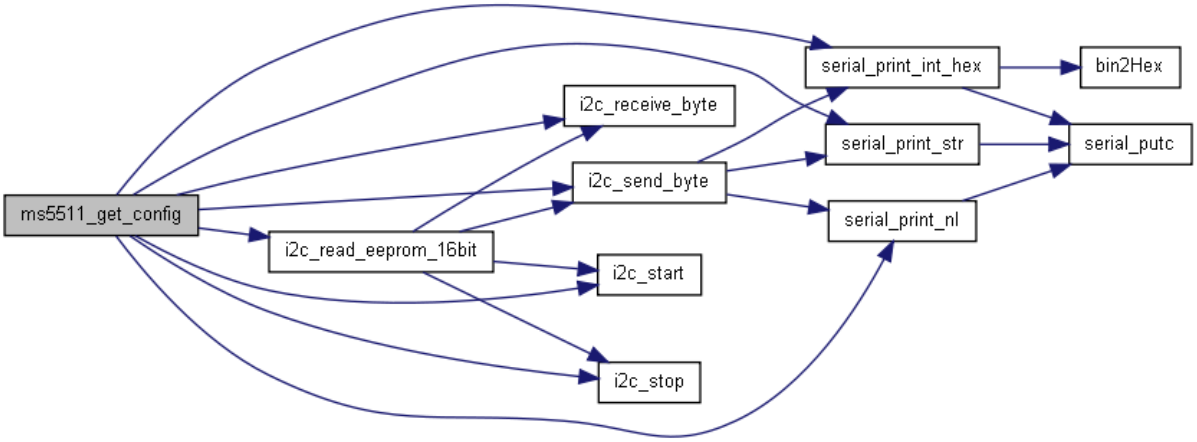
void ms5511_calc_temp_and_pressure ()

```
160                                         {
161 }
```

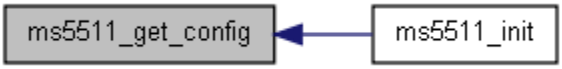
uns16 ms5511_get_config (uns8 *config_word*)

```
67                                              {
68
69 return i2c_read_eeprom_16bit(MS5511_ADDR << 1, MS5511_PROM_READ + (config_word << 1));
70
71 uns8 r1, r2;
72
73     i2c_start();
74     i2c_send_byte(MS5511_ADDR << 1);
75     i2c_send_byte(MS5511_PROM_READ + (config_word << 1));
76     i2c_stop();
77
78     i2c_start();
79     i2c_send_byte((MS5511_ADDR << 1) + 1);   // read
80
81     r1 = i2c_receive_byte();
82     serial_print_str("r1=");
83     serial_print_int_hex(r1);
84     serial_print_nl();
85
86     r2 = i2c_receive_byte();
87     serial_print_str("r2=");
88     serial_print_int_hex(r2);
89     serial_print_nl();
90
91     return (r1 << 8) + r2;
92
93
94 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



uns32 ms5511_get_raw_pressure ()

```
96                                  {
97
98 uns32 result;
99
100     return result;
101
102 }
```

uns32 ms5511_get_raw_temp ()

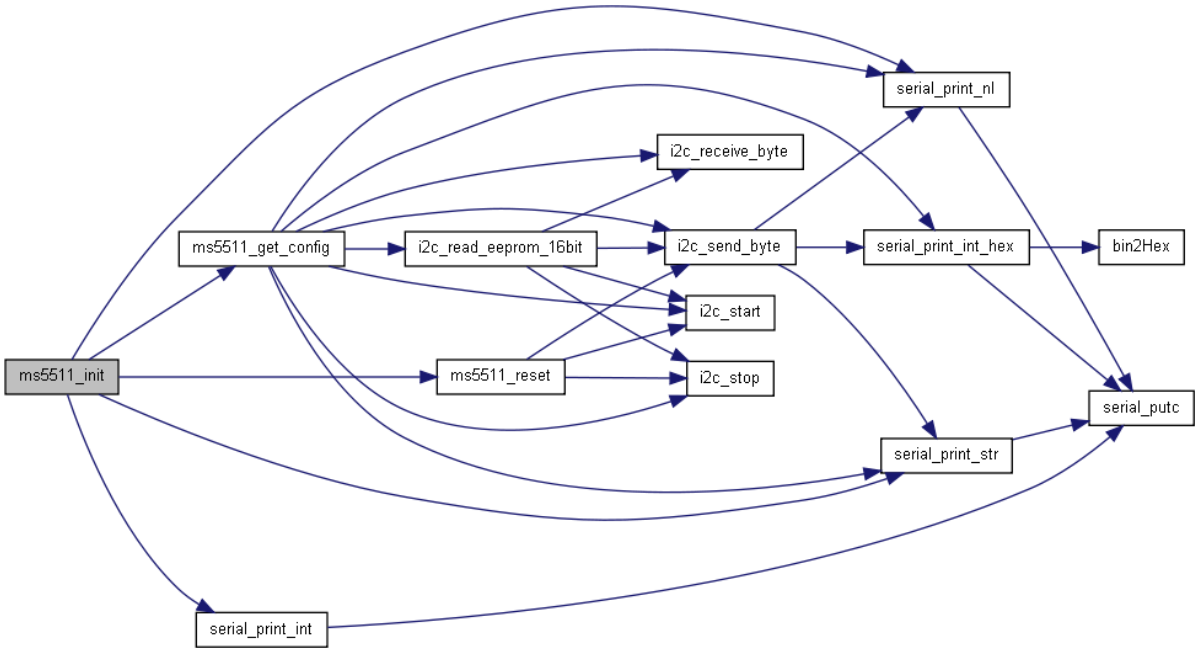
```
104                                 {
105
106 }
```

void ms5511_init ()

```
108                     {
109 uns16 c0, c7;
110     // reset
111     ms5511_reset();
112
113     // get config
114     c0 = ms5511_get_config(0);
115
116     serial_print_str("C1\n");
117     c1 = ms5511_get_config(1);
118     serial_print_str("C2\n");
119     c2 = ms5511_get_config(2);
120     serial_print_str("C3\n");
121     c3 = ms5511_get_config(3);
122     serial_print_str("C4\n");
123     c4 = ms5511_get_config(4);
124     serial_print_str("C5\n");
125     c5 = ms5511_get_config(5);
```

```
126     serial_print_str("C6    \n");
127     c6 = ms5511_get_config(6);
128     c7 = ms5511_get_config(7);
129 
130     serial_print_str("c0=");
131     serial_print_int(c0);
132     serial_print_nl();
133     serial_print_str("c1=");
134     serial_print_int(c1);
135     serial_print_nl();
136     serial_print_str("c2=");
137     serial_print_int(c2);
138     serial_print_nl();
139     serial_print_str("c3=");
140     serial_print_int(c3);
141     serial_print_nl();
142     serial_print_str("c4=");
143     serial_print_int(c4);
144     serial_print_nl();
145     serial_print_str("c5=");
146     serial_print_int(c5);
147     serial_print_nl();
148     serial_print_str("c6=");
149     serial_print_int(c6);
150     serial_print_nl();
151     serial_print_str("c7=");
152     serial_print_int(c7);
153     serial_print_nl();
154 
155 
156 }
```

Here is the call graph for this function:

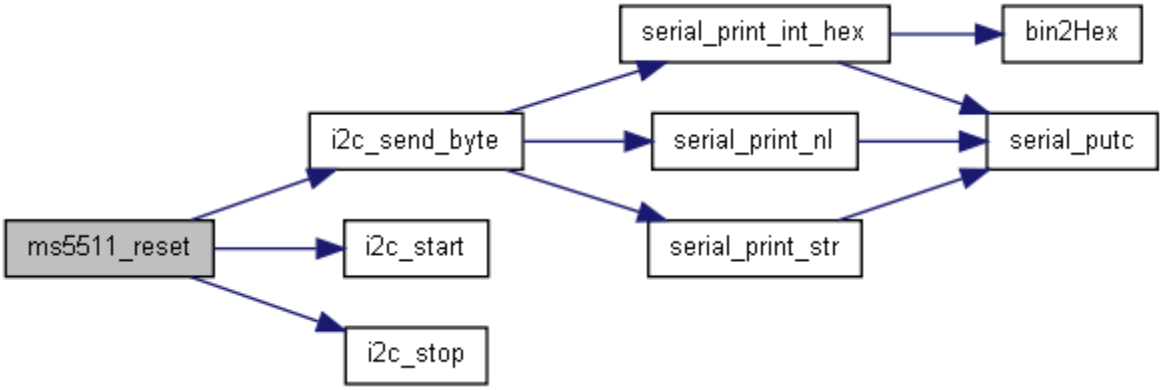


void ms5511_reset ()

```
56                      {
57 
58     i2c_start();
59     i2c_send_byte(MS5511_ADDR << 1);
```

```
60      i2c_send_byte(MS5511_RESET);
61      i2c_stop();
62
63  }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void ms5511_setup_io (void)

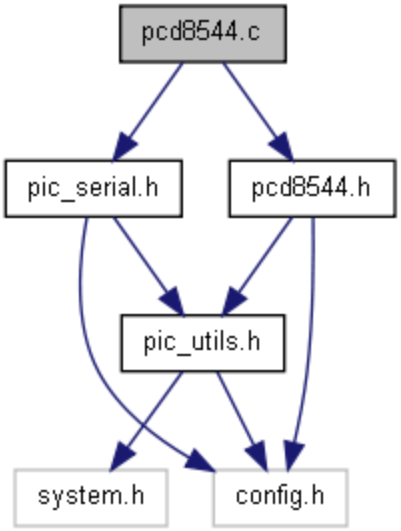
```
48                              {
49
50      i2c_setup_io();
51
52  }
```

Here is the call graph for this function:

---

pcd8544.c File Reference

Include dependency graph for pcd8544.c:



Functions

- void pcd8544_init ()
- void pcd8544_send_byte (uns8 b)
- void pcd8544_send_command (uns8 cmd)
- void pcd8544_send_data (uns8 data)
- void pcd8544_setup_io ()

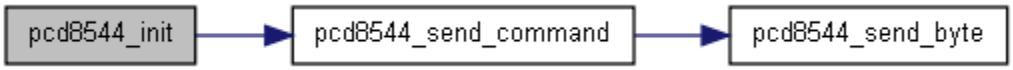
Function Documentation

void pcd8544_init ()

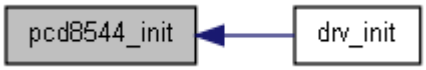
```
55                      {
56
57      /*
58      set_pin   (pcd8544_res_port, pcd8544_res_pin);
59      set_pin(pcd8544_sce_port,  pcd8544_sce_pin);
60      */
61
62      // toggle reset
63      // must be at least 100ns
64      clear_pin(pcd8544_res_port, pcd8544_res_pin);       // pulse low for reset
65      delay_ms(100);
66      set_pin   (pcd8544_res_port, pcd8544_res_pin);
67
68      // Function set: PD=0 (chip active), V=1 (vertical addressing), H=1 (extended instructions)
69      pcd8544_send_command(0b00100011);   // 0x21
70
71      // Set Vop (contrast) to 72 - 7 least significant bits
72      pcd8544_send_command(0b11001000);   // 0xc8
73
74      // Set temperature coefficient to 2 - 2 least significant bits
75      pcd8544_send_command(0b00000110);   // 0x06
76
77      // Set bias mode to 3 (1:48) - 3 least significant bits
78      pcd8544_send_command(0b00010011);   // 0x13
79
80      // Function set: PD=0 (chip active), V=1 (vertical addressing), H=0 (basic instructions)
81      pcd8544_send_command(0b00100010);   // 0x20
82
```

```
83      // Display control: D=0, E=0 (Blank the screen)
84      pcd8544_send_command(0b00001000);   // 0x08
85
86      // Display control: D=1, E=0 (Normal mode)
87      pcd8544_send_command(0b00001100);
88
89  }
```

Here is the call graph for this function:



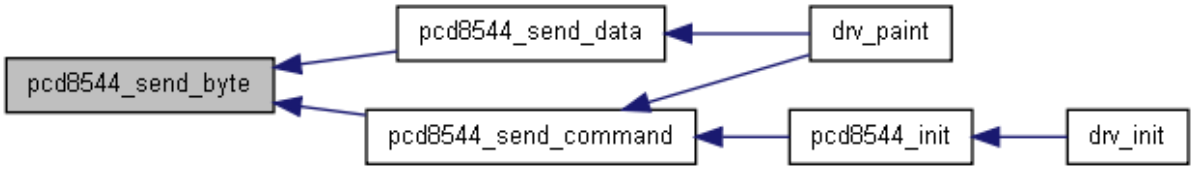
Here is the caller graph for this function:



void pcd8544_send_byte (uns8 *b*)

```
107                                 {
108
109 uns8 count;
110
111     // msb sent first
112     for (count = 0; count < 8; count++) {
113         change_pin(pcd8544_sdin_port, pcd8544_sdin_pin, b.7);
114         b = b << 1;
115         // toggle the clock
116         set_pin(pcd8544_sclk_port, pcd8544_sclk_pin);
117         clear_pin(pcd8544_sclk_port, pcd8544_sclk_pin);
118     }
119 }
```

Here is the caller graph for this function:

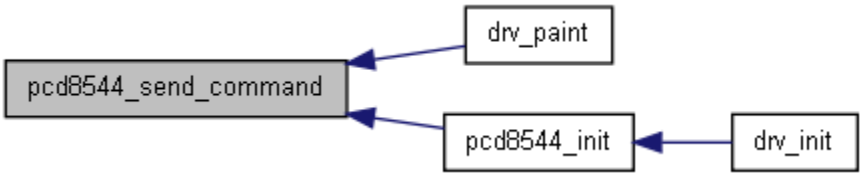


void pcd8544_send_command (uns8 *cmd*)

```
91                                      {
92
93      clear_pin(pcd8544_sce_port, pcd8544_sce_pin);
94      clear_pin(pcd8544_dc_port, pcd8544_dc_pin);
95      pcd8544_send_byte(cmd);
96      set_pin(pcd8544_sce_port, pcd8544_sce_pin);
97  }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void pcd8544_send_data (uns8 *data*)

```
99                                  {
100
101     clear_pin(pcd8544_sce_port, pcd8544_sce_pin);
102     set_pin(pcd8544_dc_port, pcd8544_dc_pin);
103     pcd8544_send_byte(data);
104     set_pin(pcd8544_sce_port, pcd8544_sce_pin);
105 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

void pcd8544_setup_io ()

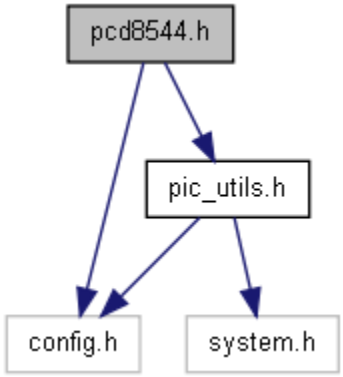
```
40                          {
41
42     // Do this in software for the moment
43
44     make_output(pcd8544_res_port,  pcd8544_res_pin);
45     make_output(pcd8544_sclk_port, pcd8544_sclk_pin);
46     make_output(pcd8544_sdin_port, pcd8544_sdin_pin);
47     make_output(pcd8544_dc_port,   pcd8544_dc_pin);
48     make_output(pcd8544_sce_port,  pcd8544_sce_pin);
49
50     set_pin(pcd8544_res_port, pcd8544_res_pin);     // not in reset
51     set_pin(pcd8544_sce_port,  pcd8544_sce_pin);    // ignore clock for the moment
52     clear_pin(pcd8544_sclk_port, pcd8544_sclk_pin); // clear clock
53 }
```

Here is the caller graph for this function:

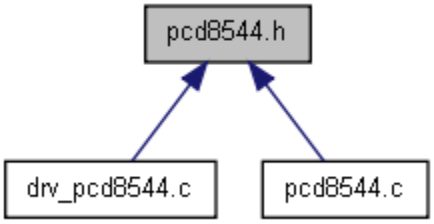
---

pcd8544.h File Reference

PCD8544 Interface routines (used in Nokia 3310 LCD).
Include dependency graph for pcd8544.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define pcd8544_setup() pcd8544_setup_io()

Functions

- void pcd8544_clear ()
- void pcd8544_init ()
- void pcd8544_send_byte (uns8 b)
- void pcd8544_send_command (uns8 command)
- void pcd8544_send_data (uns8 data)
- void pcd8544_set_pixel (uns8 x, uns8 y, uns8 colour)
- void pcd8544_setup_io ()
- void pcd8544_write (uns8 mem_addr, uns8 data)

Detailed Description

Routines to communicate with Nokia 3310 LCD display via the PCD8544 chip

---

Define Documentation

#define pcd8544_setup() pcd8544_setup_io()

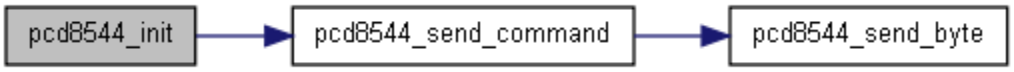
Function Documentation

void pcd8544_clear ()

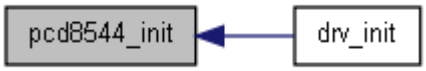
void pcd8544_init ()

```
55                      {
56
57     /*
58     set pin   (pcd8544_res_port, pcd8544_res_pin);
59     set_pin(pcd8544_sce_port,  pcd8544_sce_pin);
60     */
61
62     // toggle reset
63     // must be at least 100ns
64     clear_pin(pcd8544_res_port, pcd8544_res_pin);       // pulse low for reset
65     delay_ms(100);
66     set_pin   (pcd8544_res_port, pcd8544_res_pin);
67
68     // Function set: PD=0 (chip active), V=1 (vertical addressing), H=1 (extended instructions)
69     pcd8544_send_command(0b00100011);   // 0x21
70
71     // Set Vop (contrast) to 72 - 7 least significant bits
72     pcd8544_send_command(0b11001000);   // 0xc8
73
74     // Set temperature coefficient to 2 - 2 least significant bits
75     pcd8544_send_command(0b00000110);   // 0x06
76
77     // Set bias mode to 3 (1:48) - 3 least significant bits
78     pcd8544_send_command(0b00010011);   // 0x13
79
80     // Function set: PD=0 (chip active), V=1 (vertical addressing), H=0 (basic instructions)
81     pcd8544_send_command(0b00100010);   // 0x20
82
83     // Display control: D=0, E=0 (Blank the screen)
84     pcd8544_send_command(0b00001000);   // 0x08
85
86     // Display control: D=1, E=0 (Normal mode)
87     pcd8544_send_command(0b00001100);
88
89 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

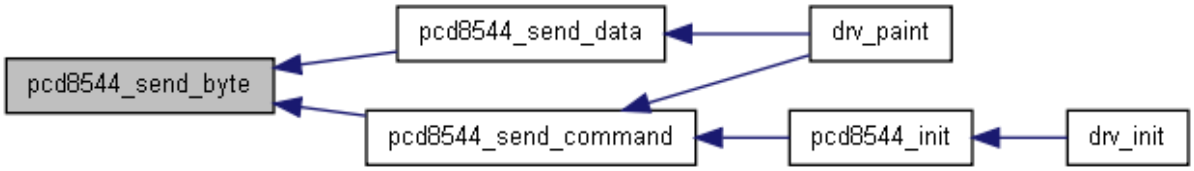


void pcd8544_send_byte (uns8 *b*)

```
107                                 {
108
109 uns8 count;
110
111     // msb sent first
112     for (count = 0; count < 8; count++) {
113         change_pin(pcd8544_sdin_port, pcd8544_sdin_pin, b.7);
114         b = b << 1;
115         // toggle the clock
```

```
116         set_pin(pcd8544_sclk_port, pcd8544_sclk_pin);
117         clear_pin(pcd8544_sclk_port, pcd8544_sclk_pin);
118     }
119 }
```

Here is the caller graph for this function:

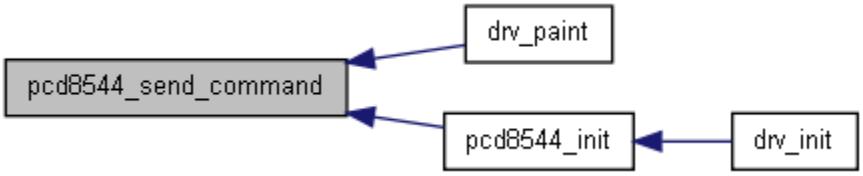


void pcd8544_send_command (uns8 *command*)

```
91                                      {
92 
93     clear_pin(pcd8544_sce_port, pcd8544_sce_pin);
94     clear_pin(pcd8544_dc_port, pcd8544_dc_pin);
95     pcd8544_send_byte(cmd);
96     set_pin(pcd8544_sce_port, pcd8544_sce_pin);
97 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void pcd8544_send_data (uns8 *data*)

```
99                                      {
100 
101     clear_pin(pcd8544_sce_port, pcd8544_sce_pin);
102     set_pin(pcd8544_dc_port, pcd8544_dc_pin);
103     pcd8544_send_byte(data);
104     set_pin(pcd8544_sce_port, pcd8544_sce_pin);
105 }
```

Here is the call graph for this function:

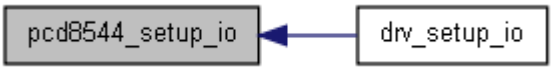
Here is the caller graph for this function:

void pcd8544_set_pixel (uns8 *x*, uns8 *y*, uns8 *colour*)

void pcd8544_setup_io ()

```
40                          {
41
42     // Do this in software for the moment
43
44     make output(pcd8544 res port,  pcd8544 res pin);
45     make output(pcd8544_sclk_port, pcd8544_sclk_pin);
46     make output(pcd8544_sdin_port, pcd8544_sdin_pin);
47     make output(pcd8544_dc_port,   pcd8544_dc_pin);
48     make output(pcd8544_sce_port,  pcd8544_sce_pin);
49
50     set pin(pcd8544_res_port, pcd8544_res_pin);     // not in reset
51     set pin(pcd8544_sce_port,  pcd8544_sce_pin);    // ignore clock for the moment
52     clear pin(pcd8544_sclk_port, pcd8544_sclk_pin); // clear clock
53 }
```

Here is the caller graph for this function:

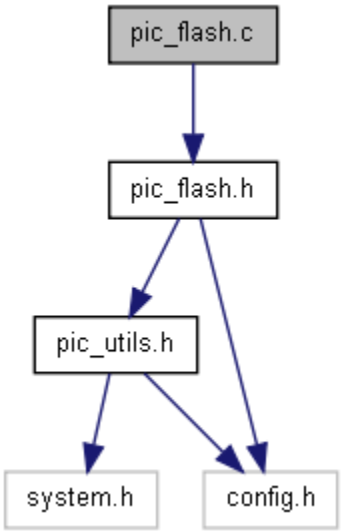


void pcd8544_write (uns8 *mem_addr*, uns8 *data*)

---

pic_flash.c File Reference

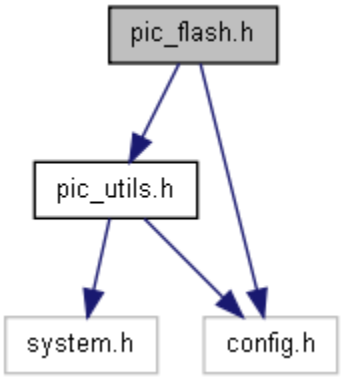
Include dependency graph for pic_flash.c:



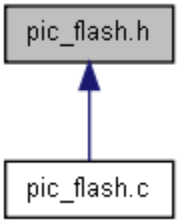
---

pic_flash.h File Reference

Flash write / erase routines.
Include dependency graph for pic_flash.h:



This graph shows which files directly or indirectly include this file:



Functions

- void flash_erase (FLASH_MEM_ADDR_TYPE address)
- void flash_write (FLASH_MEM_ADDR_TYPE address, uns8 count, uns8 *data)

Detailed Description

Function Documentation

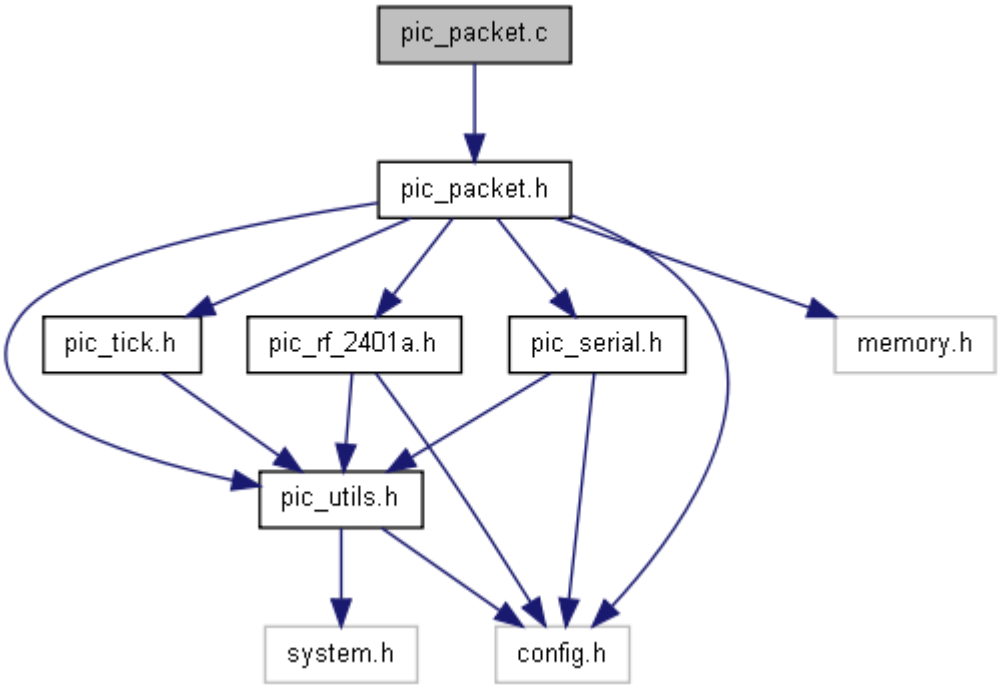
void flash_erase (FLASH_MEM_ADDR_TYPE *address*)

void flash_write (FLASH_MEM_ADDR_TYPE *address*, uns8 *count*, uns8 * *data*)

---

pic_packet.c File Reference

Include dependency graph for pic_packet.c:



Data Structures

- union rf_packet
- struct seen_packet
- struct sending_item

Defines

- #define PKT_FLAG_DELETED 0xff

Functions

- void pkt_calc_check_byte (rf_packet *packet)
- uns8 pkt_check_check_byte (rf_packet *packet)
- void pkt_init (uns16 my_addr, uns16 last_sent_id)
- *Initialise packet delivery system.* uns8 pkt_print_packet (rf_packet *my_packet)
- uns8 pkt_process_rf_data (uns8 *data_in)
- *Process received RF data.* void pkt_process_tx_queue ()
- *Handle queued items.* uns8 pkt_queue_packet (rf_packet *packet, uns8 resend)
- uns8 pkt_seen (uns16 pkt_id, uns16 source_addr)
- void pkt_send_packet (rf_packet *packet)
- uns8 pkt_send_payload (uns16 dest_addr, uns8 *payload, uns8 resend)

Send a payload via the packet delivery system. Variables

- uns16 pkt_my_addr = 0x66
- uns16 pkt_my_next_pkt_id = 0
- static seen_packet pkt_seen_list [PKT_SEEN_LIST_SIZE]
- uns8 pkt_seen_list_last = 0
- static sending_item pkt_tx_queue [PKT_TX_QUEUE_SIZE]

Define Documentation

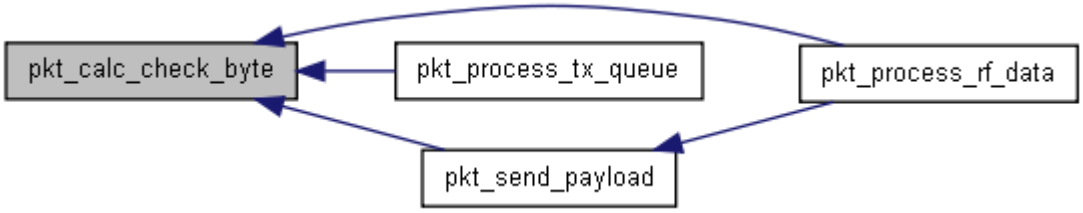
#define PKT_FLAG_DELETED 0xff

Function Documentation

void pkt_calc_check_byte (rf_packet * *packet*)

```
93                                                  {
94
95      uns8 calc, count;
96
97      calc = 0;
98      for (count = 0; count < PKT_PACKET_SIZE -1; count++) { // -1 since we don't want to include
the check digit
99          calc ^= packet->a[count];
100     }
101     packet->d.check_byte = calc;
102 }
```

Here is the caller graph for this function:



uns8 pkt_check_check_byte (rf_packet * *packet*)

```
104                                                 {
105
106     uns8 calc, count;
107
108     calc = 0;
109     for (count = 0; count < PKT_PACKET_SIZE -1; count++) { // -1 since we don't want to include
the check digit
110         calc ^= packet->a[count];
111     }
112
113     if (calc == packet->d.check_byte)
114         return 1;
115      else
116         return 0;
117
118 }
```

Here is the caller graph for this function:

void pkt_init (uns16 *my_addr*, uns16 *last_sent_pkt_id*)

Initialise the packet delivery system ready for use. This routine clears the transmit queue and seen queue. If you don't store the last_sent_pkt_id (eg, in EEPROM) then set this to 0.

Parameters:

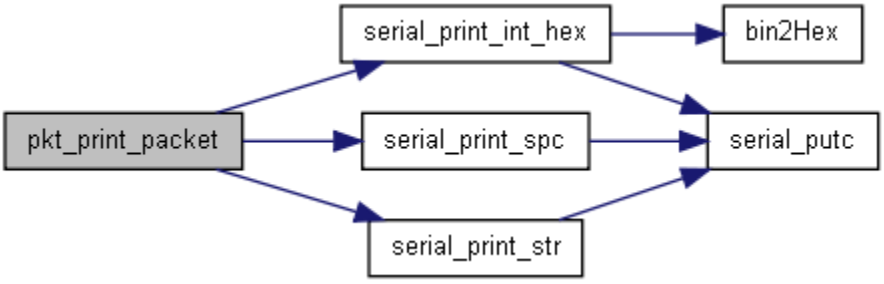
my_addr The address of this system
last_sent_pkt_id The last pkt_id used by this system.

```
423                                                 {
424     // go through tx queue and set all flags to DELETED
425 uns8 count;
426 uns16 current_tick;
427 
428     pkt_my_addr = my_addr;  // store my address
429     pkt_my_next_pkt_id = last_sent_id + 1;  // get next packet id
430     for (count = 0; count < PKT_TX_QUEUE_SIZE; count++) {
431         pkt_tx_queue[count].flag = PKT_FLAG_DELETED;  // deleted
432     }
433 
434     for (count = 0; count < PKT_SEEN_LIST_SIZE; count++ ) {
435         pkt_seen_list[count].source_addr = 0xffff;
436     }
437 }
```

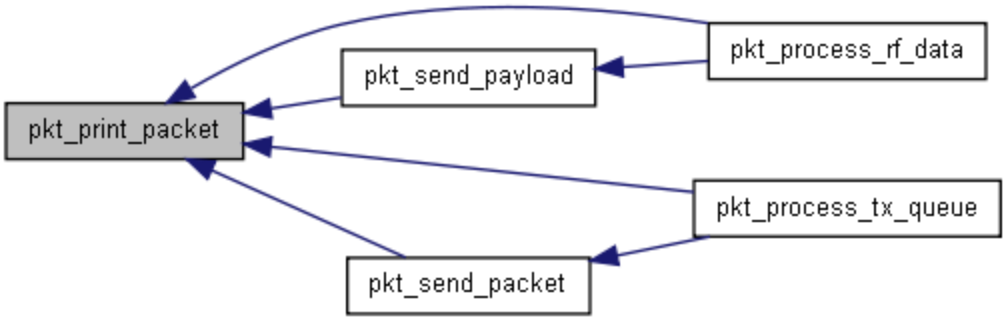
uns8 pkt_print_packet (rf_packet * *my_packet*)

```
120                                             {
121 
122     serial_print_str("[Pkt: s:");
123     serial_print_int_hex(my_packet->d.source_addr);
124     serial_print_str(" i:");
125     serial_print_int_hex(my_packet->d.pkt_id);
126     serial_print_str(" d:");
127     serial_print_int_hex(my_packet->d.dest_addr);
128     serial_print_str(" r1:");
129     serial_print_int_hex(my_packet->d.r1_addr);
130     serial_print_str(" r2:");
131     serial_print_int_hex(my_packet->d.r2_addr);
132     serial_print_str(" r3:");
133     serial_print_int_hex(my_packet->d.r3_addr);
134     serial_print_str(" p:");
135     for (uns8 count = 0; count < PKT_PAYLOAD_SIZE; count++) {
136         serial_print_int_hex(my_packet->d.payload[count]);
137         serial_print_spc();
138     }
139     serial_print_str("] ");
140 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



uns8 pkt_process_rf_data (uns8 * *data_in*)

Call this routine when your RF device has received a chunk of data from somewhere. The packet delivery system will handle everything including acknowledgements and ignoring packets that it has already seen.

Parameters:

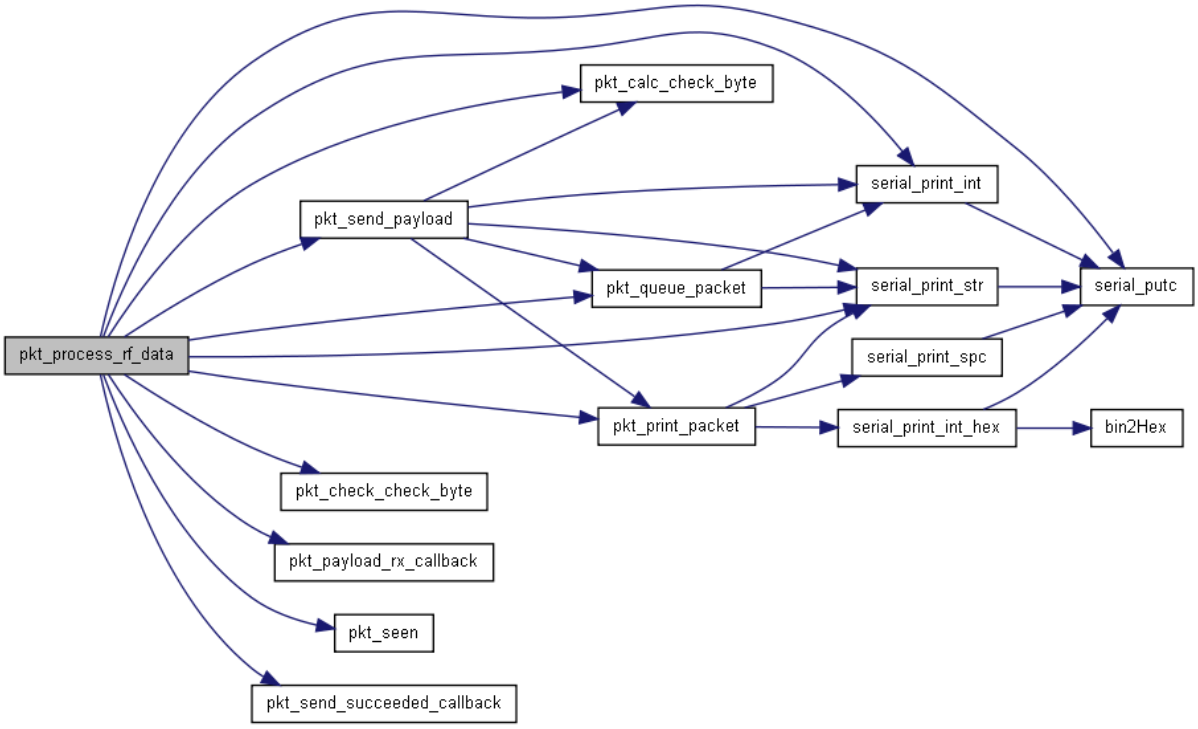
pkt_in A pointer to the received data. It is assumed that this will point to PKT_PACKET_SIZE bytes of data.

```
174                                         {
175
176 uns8 status = 0;
177 uns8 count;
178 uns16 orig_pkt_id;
179 uns8 ack_payload[PKT_PAYLOAD_SIZE] = { 0xff, 0xff };
180 sending item *pitem;
181 rf packet packet;
182     start crit sec();
183
184     memcpy(/*dst*/  (void *)&packet, // copy it in
185            /*src*/  (void *)data_in,
186            /*len*/  PKT PACKET SIZE);
187     end crit sec();
188
189     #ifdef PKT DEBUG
190         serial putc('r');
191         pkt print packet(&packet);
192     #endif
193     if (!pkt check check byte(&packet)) {
194         #ifdef PKT_DEBUG_HIGH
195             serial print str("CF! ");
196         #endif
197         return PKT STATUS CHECK FAIL;
198     }
199
200     // Am I the dest addr?
201     if ((packet.d.dest addr == pkt my addr) ||
202         (packet.d.dest_addr == PKT BROADCAST ADDR)) {
203         status = PKT STATUS PKT IS FOR ME;
204         // It's for me. But is it an ack?
205         if ((packet.d.payload[0] != 0xff) ||
206             (packet.d.payload[1] != 0xff)) { // not an ack, so send one
207             ack_payload[2] = packet.d.pkt_id & 0xff;
208             ack_payload[3] = packet.d.pkt_id >> 8;
209             #ifndef PKT_DEBUG_NO_TRANSMIT
210                 uns8 send status = pkt send payload(packet.d.source addr, ack payload,
PKT FLAG NO RESEND);
211                 #ifdef PKT_DEBUG
212                     serial print str("ACKst=");
213                     serial print int(send_status);
214                 #endif
215             #endif
216         } else { // We got an ack!
217             status = PKT STATUS PKT IS ACK FOR ME;
218             orig_pkt_id = packet.d.payload[3];
```

```
219             orig_pkt_id <<= 8;
220             orig_pkt_id += packet.d.payload[2];
221             for (count = 0; count < PKT TX QUEUE SIZE; count++) { // Search for a packet
222                 pitem = &pkt tx queue[count];
223                 if ((pitem->flag != PKT FLAG DELETED) &&
224                 (pitem->packet.d.pkt id == orig_pkt_id)) {
225                     status = PKT STATUS PKT IS FACK FOR ME;  // Found the ack
226                     #ifdef PKT_CALLBACK_ON_SEND_SUCCEEDED
227                         pkt send succeeded callback(pitem->packet.d.dest addr,
pitem->packet.d.pkt id);
228                     #endif
229 
230                     pitem->flag = PKT FLAG DELETED;
231                 }
232             }
233         } // [we got an ack]
234 
235 
236         // If not seen before, then process
237         if (!pkt seen(packet.d.pkt_id, packet.d.source_addr)) {
238 
239             // process payload
240             if ((status != PKT STATUS PKT IS FACK FOR ME) &&
241                (status != PKT STATUS PKT IS ACK FOR ME)) {
242                 pkt payload rx callback(packet.d.source_addr, packet.d.pkt_id,
packet.d.payload);
243             }
244             // Add to seen list
245             pkt seen list last++;
246             if (pkt seen list last == PKT_SEEN_LIST_SIZE) {
247                 pkt seen list last = 0;
248             }
249             pkt seen list[pkt seen list last].pkt id = packet.d.pkt_id;
250             pkt seen list[pkt seen list last].source addr = packet.d.source_addr;
251         } else {
252             status = PKT STATUS PKT FOR ME BUT SEEN;
253             #ifdef PKT_DEBUG_HIGH
254                 serial print str(" seen ");
255             #endif
256         }
257     }
258     else
259     // Is sender + pkt_id seen?
260     if (pkt seen(packet.d.pkt id, packet.d.source addr)) {
261         // We've seen this packet before
262         status = PKT STATUS SEEN BEFORE;
263     } else
264     // Am I the sender?
265     if (packet.d.source addr == pkt my addr) {
266         status = PKT STATUS I AM SENDER;
267     } else
268     // Is the r1_addr = 0xffff, meaning direct send?
269     if (packet.d.r1_addr == PKT DIRECT SEND ADDR) {
270         status = PKT STATUS DIRECT SEND;
271     } else
272     // Am I one of the routers?
273     if ((packet.d.r1 addr == pkt my addr) ||
274         (packet.d.r2_addr == pkt my addr) ||
275         (packet.d.r3_addr == pkt my addr)) {
276         status = PKT STATUS PREVIOUS ROUTED VIA ME;
277     } else
278     // Is r3 full?
279     if (packet.d.r3_addr != 0) {
280         status = PKT STATUS ROUTING FULL;
281     } else {  // I need to re-broadcast this packet
282         status = PKT STATUS NEED TO REBROADCAST;
283         // Add me as a router
284         if (packet.d.r1_addr == 0) {
285             packet.d.r1 addr = pkt my addr;
286         } else if (packet.d.r2_addr == 0) {
287             packet.d.r2_addr = pkt my addr;
```

```
288         } else {
289             packet.d.r3_addr = pkt_my_addr;
290         }
291         // Update check byte now we've changed packet
292         pkt_calc_check_byte(&packet);
293         // queue packet no resend
294 
295         #ifndef PKT_DEBUG_NO_TRANSMIT
296             pkt_queue_packet(&packet, PKT_FLAG_NO_RESEND);
297         #endif
298         // add to seen list?
299     } // [I need to re-broadcast this packet]
300 
301     return status;
302 }
```

Here is the call graph for this function:



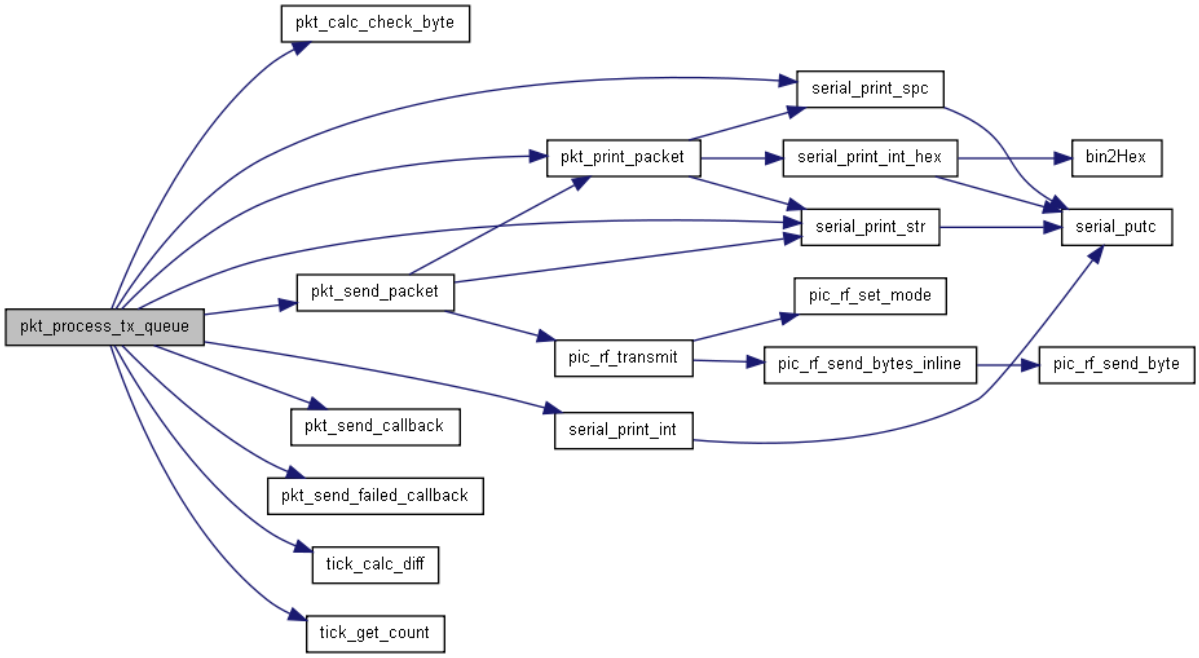
void pkt_process_tx_queue ()

Call this routine regularly (as often as possible) in your main loop in order for the packet delivery system to send any queued packets when it is appropriate to do so. It will also remove any packets from the tx queue that have been there too long.

```
354                             {
355 
356 uns8 count;
357 uns16 current_tick;
358 uns8 sent_count;
359 uns8 flag;
360 sending_item *item;
361 
362     current_tick = tick_get_count();
363 
364     for (count = 0; count < PKT_TX_QUEUE_SIZE; count++) { // search for a packet
365         item = &pkt_tx_queue[count];    // grab item to save code
366         flag = item->flag;
367         if (flag != PKT_FLAG_DELETED) { // got something to send
```

```
368             sent_count = item->sent_count;
369             if (sent_count > PKT_SEND_MAX_TRIES) { // Sent too many times
370                 item->flag = PKT_FLAG_DELETED; // Delete
371                 #ifdef PKT_CALLBACK_ON_SEND_FAILED
372                     pkt_send_failed_callback(item->packet.d.dest_addr,
item->packet.d.pkt_id);
373                 #endif
374                 #ifdef PKT_DEBUG
375                     serial_print_str(" SF!\n ");
376                 #endif
377             } else if ((sent_count == 0) // Never tried to send
378                || (tick_calc_diff(item->tick_sent, current_tick)
379                     > PKT_RESEND_TICK_DELAY*sent_count + count)) { // Or more than delay time
since last try
380                         // Note slight randomness built in by adding array position (count)
381                 item->tick_sent = current_tick; // set tick count to current
382                 //delay_us(pkt_my_addr & 0x07 *30);
383                 #ifdef PKT_DEBUG_PAYLOAD_SENT
384                     item->packet.d.payload[7] = sent_count;
385                     pkt_calc_check_byte((rf_packet*)item);
386                 #endif
387                 #ifdef PKT_DEBUG_HIGH
388                     serial_print_str(" t=");
389                     serial_print_int(current_tick);
390                     serial_print_spc();
391                 #endif
392                 pkt_send_packet((rf_packet*)item);  // send it
393                 #ifdef PKT_CALLBACK_ON_SEND
394                     pkt_send_callback(item->packet.d.dest_addr, item->packet.d.pkt_id);
395                 #endif
396
397                 if (flag != PKT_FLAG_RESEND) {  // no resend?
398                     item->flag = PKT_FLAG_DELETED;  // then delete
399                     #ifdef PKT_DEBUG_HIGH
400                         serial_print_str(" SNR ");
401                         pkt_print_packet((rf_packet*)item);
402                     #endif
403
404                 } else { // Yes resend
405                     item->sent_count++;
406                     #ifdef PKT_DEBUG_HIGH
407                         serial_print_str("SC ");
408                         serial_print_int(pkt_tx_queue[count].sent_count);
409                         serial_print_spc();
410                     #endif
411
412                     // clear direct send, next one goes to everyone!
413                     if (item->packet.d.r1_addr == 0xffff) {
414                         item->packet.d.r1_addr = 0;
415                     }
416                 } // [Yes resend]
417             } // [Okay to send it]
418         }  // [Got something to send]
419     } // [Search for a packet]
420 } // 269 + 12
```

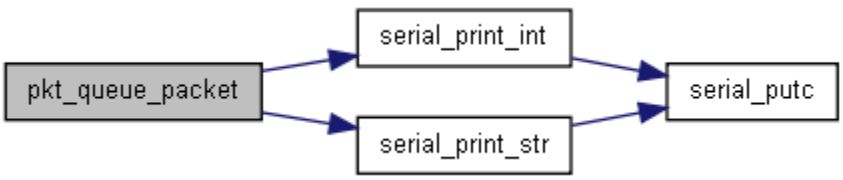
Here is the call graph for this function:



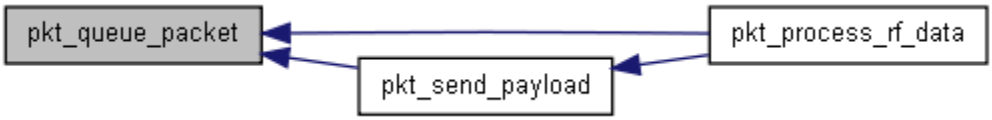
uns8 pkt_queue_packet (rf_packet * *packet*, uns8 *resend*)

```
145 {
146 uns8 count, found;
147     // find slot
148     found = 0;
149     for (count = 0; count < PKT_TX_QUEUE_SIZE; count++) {
150         if (pkt_tx_queue[count].flag == PKT_FLAG_DELETED) {
151             found = 1;
152             break;
153         }
154     }
155     if (found) {
156         #ifdef PKT_DEBUG_HIGH
157             serial_print_str(" Qin ");
158             serial_print_int(count);
159         #endif
160         memcpy(/*dst*/  (void *)&pkt_tx_queue[count], // copy it in
161                /*src*/  (void *)packet,
162                /*len*/  PKT_PACKET_SIZE);
163         pkt_tx_queue[count].sent_count = 0;
164         //  resend = resend
165         pkt_tx_queue[count].flag = resend;
166         return PKT_STATUS_QUEUED;
167     } else {
168         return PKT_STATUS_TX_QUEUE_FULL;
169     }
170
171 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



uns8 pkt_seen (uns16 *pkt_id*, uns16 *source_addr*)

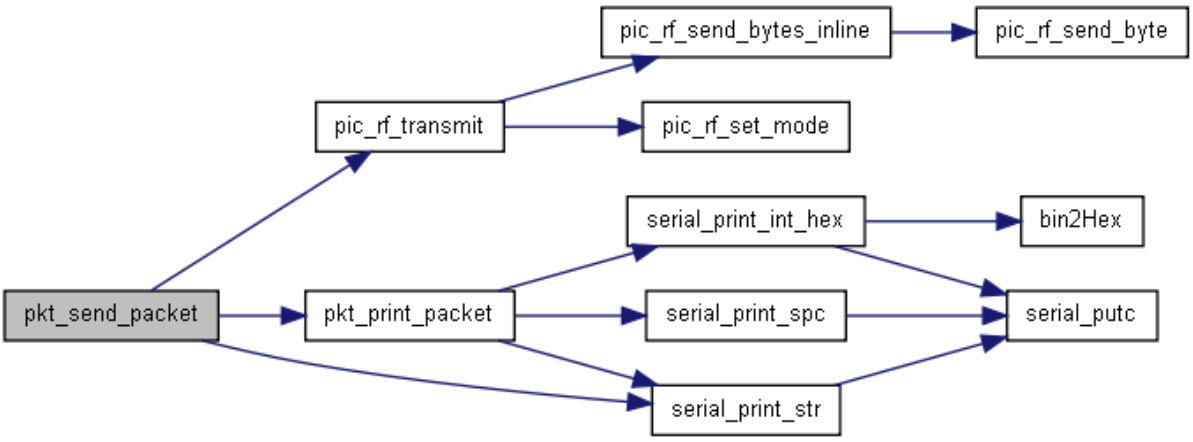
```
80                                                  {
81
82 uns8 count;
83
84     for (count = 0; count < PKT_SEEN_LIST_SIZE; count++ ) {
85         if ((pkt_seen_list[count].pkt_id == pkt_id) &&
86            (pkt_seen_list[count].source_addr == source_addr)) {
87             return 1;
88         } // if
89     } // for
90     return 0; // didn't find it
91 }
```

Here is the caller graph for this function:

void pkt_send_packet (rf_packet * *packet*) [inline]

```
328                                                  {
329
330 uns8 tx_buffer[PKT_PACKET_SIZE + 3]; // +3 for RF address (fixed)
331 uns8 count;
332
333     #ifdef PKT_DEBUG
334         serial_print_str(" <S>");
335     #endif
336     #ifdef PKT_DEBUG_HIGH
337         pkt_print_packet(packet);
338     #endif
339     tx_buffer[0] = 0b11100111; // address
340     tx_buffer[1] = 0b11100111; // address
341     tx_buffer[2] = 0b11100111; // address
342     for (count = 0; count < PKT_PACKET_SIZE; count++) {
343       tx_buffer[count+3] = packet->a[count];
344     }
345     pic_rf_transmit(tx_buffer, PKT_PACKET_SIZE + 3);
346     //pic_rf_transmit((uns8 *)packet, PKT_PACKET_SIZE);
347 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

uns8 pkt_send_payload (uns16 *dest_addr*, uns8 * *payload*, uns8 *resend*)

Use this routine to send a payload of data to the destination address. The payload must point to PKT_PAYLOAD_SIZE bytes of data (as defined in your config.h). The packet will be constructed, setting destination address, sender address (set previously in pkt_init) payload, initial routing directions and the check byte calculated. It will then be placed into the tx queue and will actually be sent next time pkt_process_tx_queue is called.

Parameters:

dest_addr Send payload to this address. Use address of PKT_BROADCAST_ADDR to broadcast to all listening local addresses

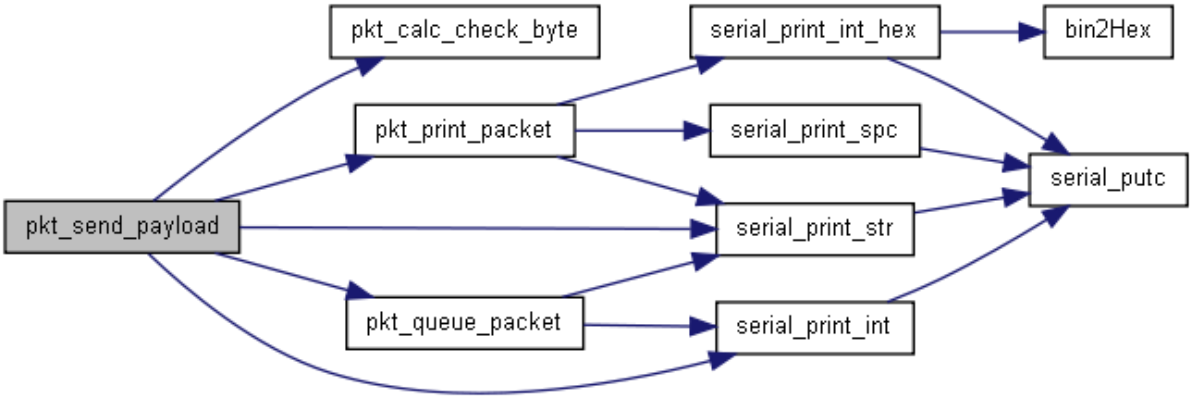
pay_load Pointer to PKT_PAYLOAD_SIZE array of bytes

resend Set to PKT_FLAG_RESEND or PKT_FLAG_NO_RESEND

```
440                                                                 {
441
442 rf_packet my_packet;
443 uns8 count;
444
445     #ifdef PKT_DEBUG_HIGH
446     serial_print_str("\npkt send d:");
447     serial_print_int(dest_addr);
448     serial_print_str("s: ");
449     serial_print_int(pkt_my_addr);
450     serial_print_str("\n");
451     #endif
452
453     // build packet
454
455     my_packet.d.source_addr = pkt_my_addr;
456     my_packet.d.pkt_id = pkt_my_next_pkt_id++;
457     my_packet.d.dest_addr = dest_addr;
458     my_packet.d.r1_addr = 0;
459     if ((dest_addr != PKT_BROADCAST_ADDR) &&
460          ((payload[0] != 0xff) || (payload[1] != 0xff))) {  // if not broadcast send
461         my_packet.d.r1_addr = PKT_DIRECT_SEND_ADDR; // for direct send
462     }
463     my_packet.d.r2_addr = 0;
464     my_packet.d.r3_addr = 0;
465     for (count = 0; count < PKT_PAYLOAD_SIZE; count++) {
466         my_packet.d.payload[count] = payload[count];
467     }
```

```
468     pkt_calc_check_byte(&my_packet);
469
470     #ifdef PKT_DEBUG
471         serial_print_str(" PS");
472         pkt_print_packet(&my_packet);
473     #endif
474
475     // Put it in the queue for sending
476
477     return pkt_queue_packet(&my_packet, resend);
478 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

Variable Documentation

uns16 pkt_my_addr = 0x66

uns16 pkt_my_next_pkt_id = 0

seen_packet pkt_seen_list[PKT_SEEN_LIST_SIZE] `[static]`

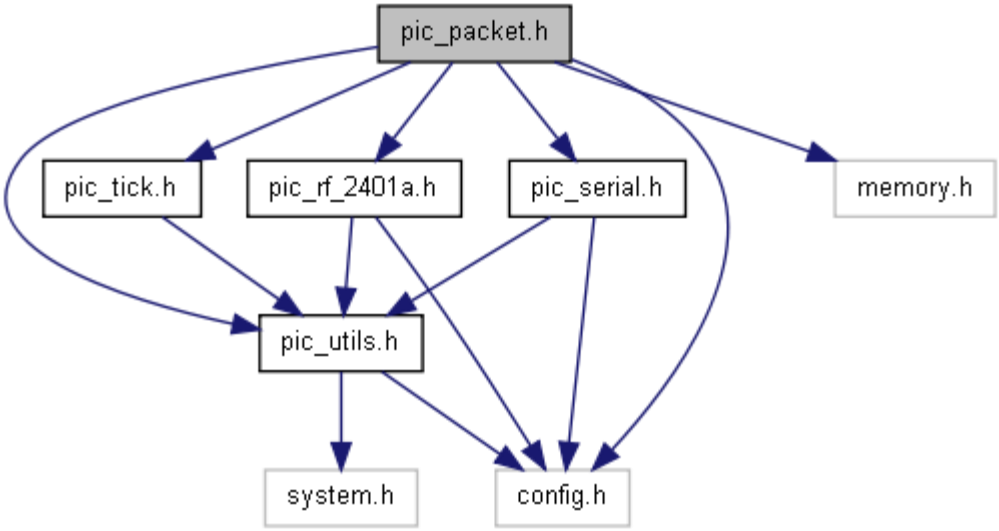
uns8 pkt_seen_list_last = 0

sending_item pkt_tx_queue[PKT_TX_QUEUE_SIZE] `[static]`

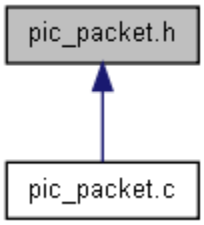
---

pic_packet.h File Reference

Pic meshed packed network library.
Include dependency graph for pic_packet.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct rf_packet_det

Defines

- #define PKT_BROADCAST_ADDR 0xfffe
- #define PKT_CONFIG_ADDR 0xfffd
- #define PKT_DIRECT_SEND_ADDR 0xffff
- #define PKT_FLAG_BROADCAST 2
- #define PKT_FLAG_NO_RESEND 0
- #define PKT_FLAG_RESEND 1
- #define PKT_PACKET_SIZE sizeof(rf_packet_det)
- #define PKT_STATUS_CHECK_FAIL 12
- #define PKT_STATUS_DIRECT_SEND 6
- #define PKT_STATUS_I_AM_SENDER 2
- #define PKT_STATUS_NEED_TO_REBROADCAST 9
- #define PKT_STATUS_PKT_FOR_ME_BUT_SEEN 13
- #define PKT_STATUS_PKT_IS_ACK_FOR_ME 4
- #define PKT_STATUS_PKT_IS_FACK_FOR_ME 5
- #define PKT_STATUS_PKT_IS_FOR_ME 3
- #define PKT_STATUS_PREVIOUS_ROUTED_VIA_ME 7
- #define PKT_STATUS_QUEUED 10
- #define PKT_STATUS_ROUTING_FULL 8
- #define PKT_STATUS_SEEN_BEFORE 1

- #define PKT_STATUS_TX_QUEUE_FULL 11
- #define RF_RX_BUFFER_SIZE PKT_PACKET_SIZE

Functions

- void pkt_init (uns16 my_addr, uns16 last_sent_pkt_id)
- *Initialise packet delivery system.* void pkt_payload_rx_callback (uns16 source_addr, uns16 pkt_id, uns8 *payload)
- *Called when a packet has been received.* uns8 pkt_process_rf_data (uns8 *data_in)
- *Process received RF data.* void pkt_process_tx_queue ()
- *Handle queued items.* void pkt_send_callback (uns16 dest_addr, uns16 pkt_id)
- void pkt_send_failed_callback (uns16 dest_addr, uns16 pkt_id)
- uns8 pkt_send_payload (uns16 dest_addr, uns8 *payload, uns8 resend)
- *Send a payload via the packet delivery system.* void pkt_send_succeeded_callback (uns16 dest_addr, uns16 pkt_id)

Detailed Description

Assumes either nrf2401a or nrf24l01 library

Define Documentation

#define PKT_BROADCAST_ADDR 0xfffe

Send to anyone that will listen

#define PKT_CONFIG_ADDR 0xfffd

Magic config packet address

#define PKT_DIRECT_SEND_ADDR 0xffff

Router address for direct send only (no routing)

#define PKT_FLAG_BROADCAST 2

Packet should be broadcast to anyone on the network

#define PKT_FLAG_NO_RESEND 0

Packet should not be resent if it fails to reach destination

#define PKT_FLAG_RESEND 1

Packet should be resent if it fails to reach destination

#define PKT_PACKET_SIZE sizeof(rf_packet_det)

#define PKT_STATUS_CHECK_FAIL 12

Packet is corrupt, ignoring

#define PKT_STATUS_DIRECT_SEND 6

Packet is direct send, but not for me, ignoring

#define PKT_STATUS_I_AM_SENDER 2

I have sent this packet, so ignoring

#define PKT_STATUS_NEED_TO_REBROADCAST 9

Packet is not for me, but protocol states I need to rebroadcast it

#define PKT_STATUS_PKT_FOR_ME_BUT_SEEN 13

Packet is for me but seen previously

#define PKT_STATUS_PKT_IS_ACK_FOR_ME 4

Packet is ACK for me, but didn't find in my transmit queue

#define PKT_STATUS_PKT_IS_FACK_FOR_ME 5

Packet is ACK for me, found in and removed from transmit queue (successful send)

#define PKT_STATUS_PKT_IS_FOR_ME 3

Packet is for me

#define PKT_STATUS_PREVIOUS_ROUTED_VIA_ME 7

Packet has my address in the router list, ignoring

#define PKT_STATUS_QUEUED 10

Packet queued for transmition

#define PKT_STATUS_ROUTING_FULL 8

Packet not retransmitted since its routing list is full

#define PKT_STATUS_SEEN_BEFORE 1

Packet is already in seen list, ignoring

#define PKT_STATUS_TX_QUEUE_FULL 11

Packet has not been queued for transmition since my transmit queue is full

#define RF_RX_BUFFER_SIZE PKT_PACKET_SIZE

Function Documentation

void pkt_init (uns16 *my_addr*, uns16 *last_sent_pkt_id*)

Initialise the packet delivery system ready for use. This routine clears the transmit queue and seen queue. If you don't store the last_sent_pkt_id (eg, in EEPROM) then set this to 0.

Parameters:

my_addr The address of this system
last_sent_pkt_id The last pkt_id used by this system.

```
423                                                 {
424     // go through tx queue and set all flags to DELETED
425 uns8 count;
426 uns16 current_tick;
427 
428     pkt_my_addr = my_addr;  // store my address
429     pkt_my_next_pkt_id = last_sent_id + 1;  // get next packet id
```

```
430     for (count = 0; count < PKT_TX_QUEUE_SIZE; count++) {
431         pkt_tx_queue[count].flag = PKT_FLAG_DELETED;  // deleted
432     }
433 
434     for (count = 0; count < PKT_SEEN_LIST_SIZE; count++ ) {
435         pkt_seen_list[count].source_addr = 0xffff;
436     }
437 }
```

void pkt_payload_rx_callback (uns16 *source_addr*, uns16 *pkt_id*, uns8 * *payload*)

Once a packet has been received, if it has not been seen before and it is destined to this address (set using pkt_init), then this routine will be called. You must have this routine defined in your own code.

Parameters:

source_addr The address of the system that sent the packet
pkt_id The ID of the packet (the source_addr and pkt_id are used to uniquely identify the packet
payload A pointer to PKT_PACKET_SIZE bytes received

Here is the caller graph for this function:

uns8 pkt_process_rf_data (uns8 * *data_in*)

Call this routine when your RF device has received a chunk of data from somewhere. The packet delivery system will handle everything including acknowledgements and ignoring packets that it has already seen.

Parameters:

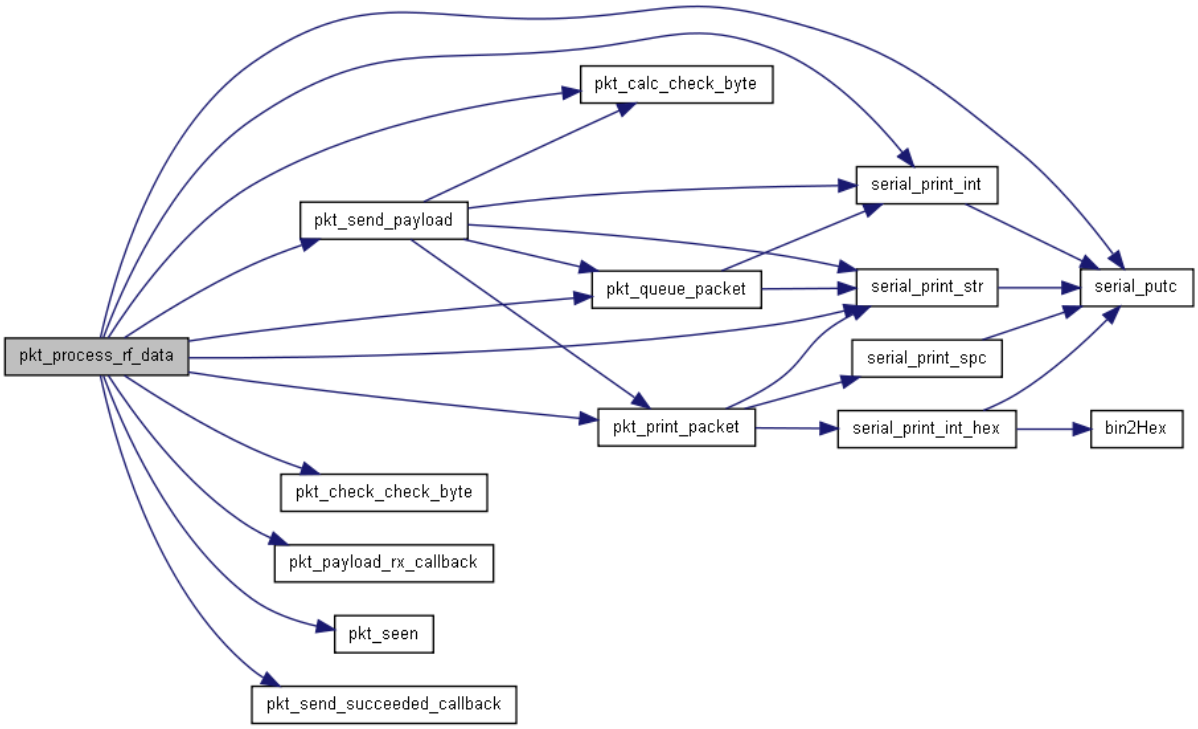
pkt_in A pointer to the received data. It is assumed that this will point to PKT_PACKET_SIZE bytes of data.

```
174                                         {
175 
176 uns8 status = 0;
177 uns8 count;
178 uns16 orig_pkt_id;
179 uns8 ack_payload[PKT_PAYLOAD_SIZE] = { 0xff, 0xff };
180 sending_item *pitem;
181 rf_packet packet;
182     start_crit_sec();
183 
184     memcpy(/*dst*/  (void *)&packet, // copy it in
185            /*src*/  (void *)data_in,
186            /*len*/  PKT_PACKET_SIZE);
187     end_crit_sec();
188 
189     #ifdef PKT_DEBUG
190         serial_putc('r');
191         pkt_print_packet(&packet);
192     #endif
193     if (!pkt_check_check_byte(&packet)) {
194         #ifdef PKT_DEBUG_HIGH
195             serial_print_str("CF! ");
196         #endif
197         return PKT_STATUS_CHECK_FAIL;
198     }
199 
200     // Am I the dest addr?
201     if ((packet.d.dest_addr == pkt_my_addr) ||
202         (packet.d.dest_addr == PKT_BROADCAST_ADDR)) {
203         status = PKT_STATUS_PKT_IS_FOR_ME;
204         // It's for me. But is it an ack?
205         if ((packet.d.payload[0] != 0xff) ||
206             (packet.d.payload[1] != 0xff)) { // not an ack, so send one
207             ack_payload[2] = packet.d.pkt_id & 0xff;
```

```
208             ack_payload[3] = packet.d.pkt_id >> 8;
209             #ifndef PKT_DEBUG_NO_TRANSMIT
210                 uns8 send status = pkt send payload(packet.d.source addr, ack payload,
PKT FLAG NO RESEND);
211                 #ifdef PKT_DEBUG
212                     serial print str("ACKst=");
213                     serial print int(send_status);
214                 #endif
215             #endif
216         } else { // We got an ack!
217             status = PKT STATUS PKT IS ACK FOR ME;
218             orig_pkt_id = packet.d.payload[3];
219             orig_pkt_id <<= 8;
220             orig_pkt_id += packet.d.payload[2];
221             for (count = 0; count < PKT TX QUEUE SIZE; count++) { // Search for a packet
222                 pitem = &pkt tx queue[count];
223                 if ((pitem->flag != PKT FLAG DELETED) &&
224                 (pitem->packet.d.pkt id == orig_pkt_id)) {
225                     status = PKT STATUS PKT IS FACK FOR ME;  // Found the ack
226                     #ifdef PKT_CALLBACK_ON_SEND_SUCCEEDED
227                         pkt send succeeded callback(pitem->packet.d.dest addr,
pitem->packet.d.pkt id);
228                     #endif
229
230                     pitem->flag = PKT FLAG DELETED;
231                 }
232             }
233         } // [we got an ack]
234
235
236         // If not seen before, then process
237         if (!pkt seen(packet.d.pkt id, packet.d.source addr)) {
238
239             // process payload
240             if ((status != PKT STATUS PKT IS FACK FOR ME) &&
241                (status != PKT STATUS PKT IS ACK FOR ME)) {
242                 pkt payload rx callback(packet.d.source_addr, packet.d.pkt_id,
packet.d.payload);
243             }
244             // Add to seen list
245             pkt seen list last++;
246             if (pkt seen list last == PKT_SEEN_LIST_SIZE) {
247                 pkt seen list last = 0;
248             }
249             pkt seen list[pkt seen list last].pkt id = packet.d.pkt_id;
250             pkt seen list[pkt seen list last].source addr = packet.d.source_addr;
251         } else {
252             status = PKT STATUS PKT FOR ME BUT SEEN;
253             #ifdef PKT DEBUG HIGH
254                 serial print str(" seen ");
255             #endif
256         }
257     }
258     else
259     // Is sender + pkt_id seen?
260     if (pkt seen(packet.d.pkt_id, packet.d.source_addr)) {
261         // We've seen this packet before
262         status = PKT STATUS SEEN BEFORE;
263     } else
264     // Am I the sender?
265     if (packet.d.source addr == pkt my addr) {
266         status = PKT STATUS I AM SENDER;
267     } else
268     // Is the r1_addr = 0xffff, meaning direct send?
269     if (packet.d.r1_addr == PKT DIRECT SEND ADDR) {
270         status = PKT STATUS DIRECT SEND;
271     } else
272     // Am I one of the routers?
273     if ((packet.d.r1 addr == pkt my addr) ||
274         (packet.d.r2_addr == pkt my addr) ||
275         (packet.d.r3_addr == pkt my addr)) {
```

```
276         status = PKT_STATUS_PREVIOUS_ROUTED_VIA_ME;
277     } else
278     // Is r3 full?
279     if (packet.d.r3_addr != 0) {
280         status = PKT_STATUS_ROUTING_FULL;
281     } else {  // I need to re-broadcast this packet
282         status = PKT_STATUS_NEED_TO_REBROADCAST;
283         // Add me as a router
284         if (packet.d.r1_addr == 0) {
285             packet.d.r1_addr = pkt_my_addr;
286         } else if (packet.d.r2_addr == 0) {
287             packet.d.r2_addr = pkt_my_addr;
288         } else {
289             packet.d.r3_addr = pkt_my_addr;
290         }
291         // Update check byte now we've changed packet
292         pkt_calc_check_byte(&packet);
293         // queue packet no resend
294 
295         #ifndef PKT_DEBUG_NO_TRANSMIT
296             pkt_queue_packet(&packet, PKT_FLAG_NO_RESEND);
297         #endif
298         // add to seen list?
299     } // [I need to re-broadcast this packet]
300 
301     return status;
302 }
```

Here is the call graph for this function:



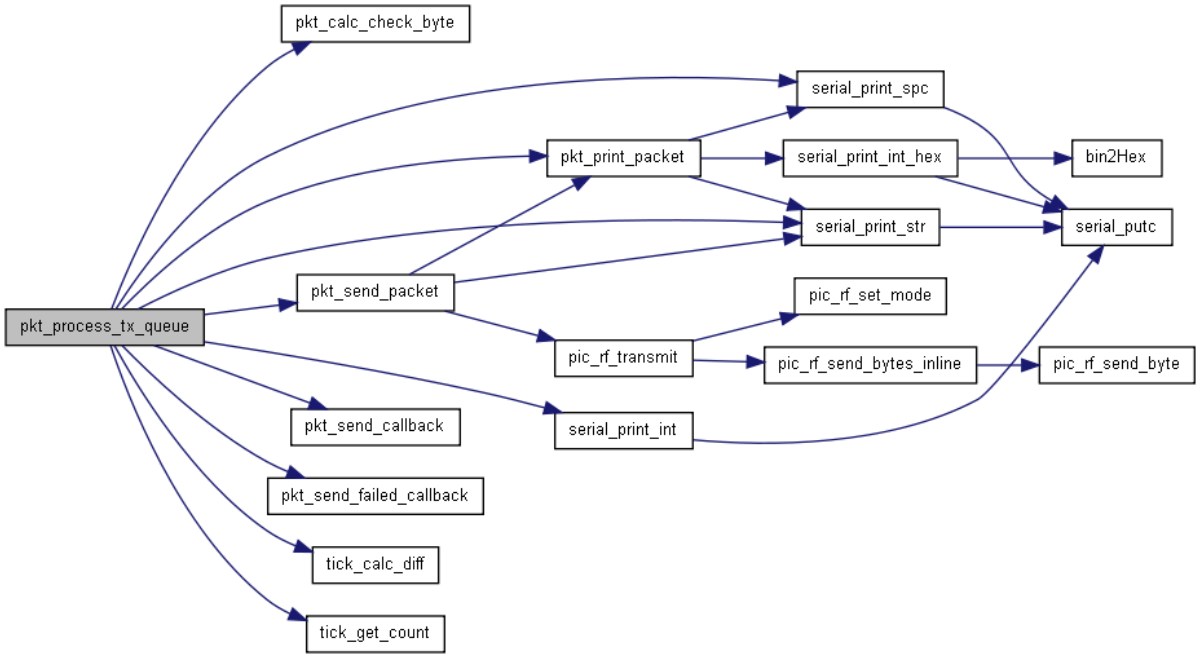
void pkt_process_tx_queue ()

Call this routine regularly (as often as possible) in your main loop in order for the packet delivery system to send any queued packets when it is appropriate to do so. It will also remove any packets from the tx queue that have been there too long.

```
354                             {
355 
```

```
356 uns8 count;
357 uns16 current_tick;
358 uns8 sent count;
359 uns8 flag;
360 sending item *item;
361
362     current_tick = tick get count();
363
364     for (count = 0; count < PKT_TX_QUEUE_SIZE; count++) { // search for a packet
365         item = &pkt tx queue[count];    // grab item to save code
366         flag = item->flag;
367         if (flag != PKT FLAG DELETED) { // got something to send
368             sent_count = item->sent count;
369             if (sent_count > PKT_SEND_MAX_TRIES) { // Sent too many times
370                 item->flag = PKT FLAG DELETED; // Delete
371                 #ifdef PKT_CALLBACK_ON_SEND_FAILED
372                     pkt send failed callback(item->packet.d.dest addr,
item->packet.d.pkt id);
373                 #endif
374                 #ifdef PKT_DEBUG
375                     serial print str(" SF!\n ");
376                 #endif
377             } else if ((sent count == 0) // Never tried to send
378                || (tick calc diff(item->tick sent, current_tick)
379                     > PKT_RESEND_TICK_DELAY*sent_count + count)) { // Or more than delay time
since last try
380                         // Note slight randomness built in by adding array position (count)
381                 item->tick sent = current tick; // set tick count to current
382                 //delay_us(pkt_my_addr & 0x07 *30);
383                 #ifdef PKT_DEBUG_PAYLOAD_SENT
384                     item->packet.d.payload[7] = sent_count;
385                     pkt calc check byte((rf packet*)item);
386                 #endif
387                 #ifdef PKT_DEBUG_HIGH
388                     serial print str(" t=");
389                     serial print int(current_tick);
390                     serial print spc();
391                 #endif
392                 pkt send packet((rf packet*)item);  // send it
393                 #ifdef PKT CALLBACK ON SEND
394                     pkt send callback(item->packet.d.dest addr, item->packet.d.pkt id);
395                 #endif
396
397                 if (flag != PKT FLAG RESEND) {  // no resend?
398                     item->flag = PKT FLAG DELETED;  // then delete
399                     #ifdef PKT_DEBUG_HIGH
400                         serial print str(" SNR ");
401                         pkt print packet((rf packet*)item);
402                     #endif
403
404                 } else { // Yes resend
405                     item->sent count++;
406                     #ifdef PKT_DEBUG_HIGH
407                         serial print str("SC ");
408                         serial print int(pkt tx queue[count].sent_count);
409                         serial print spc();
410                     #endif
411
412                     // clear direct send, next one goes to everyone!
413                     if (item->packet.d.r1 addr == 0xffff) {
414                         item->packet.d.r1 addr = 0;
415                     }
416                 } // [Yes resend]
417             } // [Okay to send it]
418         }  // [Got something to send]
419     } // [Search for a packet]
420 } // 269 + 12
```

Here is the call graph for this function:



void pkt_send_callback (uns16 *dest_addr*, uns16 *pkt_id*)

Here is the caller graph for this function:

void pkt_send_failed_callback (uns16 *dest_addr*, uns16 *pkt_id*)

Here is the caller graph for this function:

uns8 pkt_send_payload (uns16 *dest_addr*, uns8 * *payload*, uns8 *resend*)

Use this routine to send a payload of data to the destination address. The payload must point to PKT_PAYLOAD_SIZE bytes of data (as defined in your config.h). The packet will be constructed, setting destination address, sender address (set previously in pkt_init) payload, initial routing directions and the check byte calculated. It will then be placed into the tx queue and will actually be sent next time pkt_process_tx_queue is called.

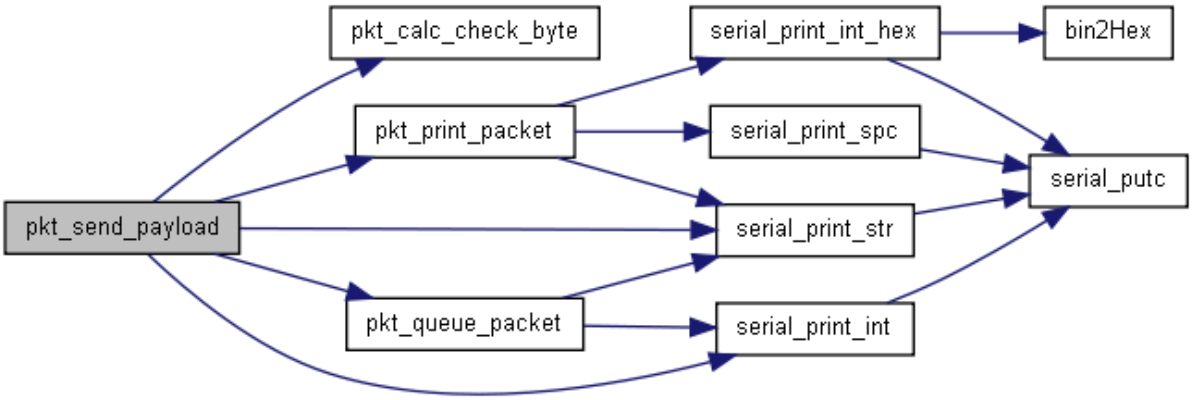
Parameters:

dest_addr Send payload to this address. Use address of PKT_BROADCAST_ADDR to broadcast to all listening local addresses
pay_load Pointer to PKT_PAYLOAD_SIZE array of bytes
resend Set to PKT_FLAG_RESEND or PKT_FLAG_NO_RESEND

```
440                                                                 {
441
442 rf_packet my_packet;
443 uns8 count;
```

```
444
445     #ifdef PKT_DEBUG_HIGH
446     serial_print_str("\npkt send d:");
447     serial_print_int(dest_addr);
448     serial_print_str("s: ");
449     serial_print_int(pkt_my_addr);
450     serial_print_str("\n");
451     #endif
452
453     // build packet
454
455     my_packet.d.source_addr = pkt_my_addr;
456     my_packet.d.pkt_id = pkt_my_next_pkt_id++;
457     my_packet.d.dest_addr = dest_addr;
458     my_packet.d.r1_addr = 0;
459     if ((dest_addr != PKT_BROADCAST_ADDR) &&
460          ((payload[0] != 0xff) || (payload[1] != 0xff))) {  // if not broadcast send
461         my_packet.d.r1_addr = PKT_DIRECT_SEND_ADDR; // for direct send
462     }
463     my_packet.d.r2_addr = 0;
464     my_packet.d.r3_addr = 0;
465     for (count = 0; count < PKT_PAYLOAD_SIZE; count++) {
466         my_packet.d.payload[count] = payload[count];
467     }
468     pkt_calc_check_byte(&my_packet);
469
470     #ifdef PKT_DEBUG
471         serial_print_str(" PS");
472         pkt_print_packet(&my_packet);
473     #endif
474
475     // Put it in the queue for sending
476
477     return pkt_queue_packet(&my_packet, resend);
478 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

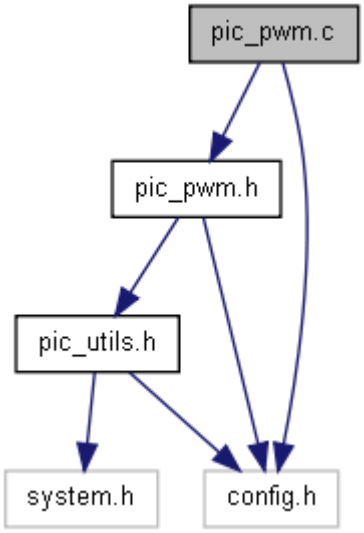
void pkt_send_succeeded_callback (uns16 *dest_addr*, uns16 *pkt_id*)

Here is the caller graph for this function:

---

pic_pwm.c File Reference

Include dependency graph for pic_pwm.c:



Functions

- uns8 pwm_get_level (uns8 pwm_item)
- void pwm_handle ()
- void pwm_set_level (uns8 pwm_item, uns8 level)
- void pwm_set_transition (uns8 pwm_item, uns8 to_level, uns16 steps)
- void pwm_setup_io ()

Variables

- uns8 pwm_count
- uns8 pwm_level [PWM_NUM_PINS]

Function Documentation

uns8 pwm_get_level (uns8 *pwm_item*)

```
50                                      {
51      return pwm_level[pwm_item];
52  }
```

void pwm_handle ()

```
70                      {
71      uns8 count;
72
73      pwm_count++;
74
75      if (pwm_count == 0) {   // Turn them all on
76          set_pin(PWM_PORT_0, PWM_PIN_0);
77          #if PWM_NUM_PINS > 1
78              set_pin(PWM_PORT_1, PWM_PIN_1);
79          #endif
80          #if PWM_NUM_PINS > 2
81              set_pin(PWM_PORT_2, PWM_PIN_2);
82          #endif
83      }
84      if (pwm_level[0] == pwm_count) {
85          clear_pin(PWM_PORT_0, PWM_PIN_0);
86      }
87      #if PWM_NUM_PINS > 1
88          if (pwm_level[1] == pwm_count) {
89              clear_pin(PWM_PORT_1, PWM_PIN_1);
90          }
91      #endif
92      #if PWM_NUM_PINS > 2
93          if (pwm_level[2] == pwm_count) {
94              clear_pin(PWM_PORT_2, PWM_PIN_2);
95          }
96      #endif
97  }
```

void pwm_set_level (uns8 *pwm_item*, uns8 *level*)

```
45                                                  {
46
47      pwm_level[pwm_item] = level;
48  }
```

void pwm_set_transition (uns8 *pwm_item*, uns8 *to_level*, uns16 *steps*)

```
54                                                                      {
55  }
```

void pwm_setup_io ()

```
58                        {
59
60      make_output(PWM_PORT_0, PWM_PIN_0);
61      #if PWM_NUM_PINS > 1
62          make_output(PWM_PORT_1, PWM_PIN_1);
63      #endif
64      #if PWM_NUM_PINS > 2
65          make_output(PWM_PORT_2, PWM_PIN_2);
66      #endif
67
68  }
```

Variable Documentation

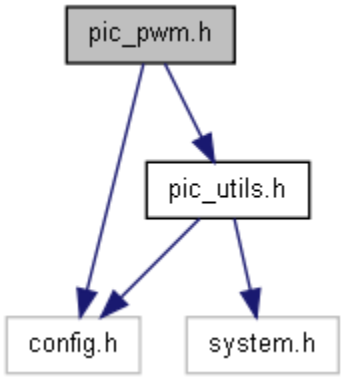
uns8 pwm_count

uns8 pwm_level[PWM_NUM_PINS]

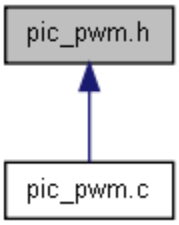
---

pic_pwm.h File Reference

Software (timer-based) PWM.
Include dependency graph for pic_pwm.h:



This graph shows which files directly or indirectly include this file:



Functions

- uns8 pwm_get_level (uns8 pwm_item)
- void pwm_handle ()
- void pwm_set_level (uns8 pwm_item, uns8 level)
- void pwm_set_transition (uns8 to_level, uns16 steps)
- void pwm_setup_io ()

Variables

- uns8 pwm_count
- uns8 pwm_level [PWM_NUM_PINS]

Detailed Description

Function Documentation

uns8 pwm_get_level (uns8 *pwm_item*)

```
50                                      {
51     return pwm_level[pwm_item];
52 }
```

void pwm_handle ()

```
70                      {
71     uns8 count;
72
73     pwm_count++;
74
75     if (pwm_count == 0) {   // Turn them all on
76         set_pin(PWM_PORT_0, PWM_PIN_0);
77         #if PWM_NUM_PINS > 1
78             set_pin(PWM_PORT_1, PWM_PIN_1);
79         #endif
80         #if PWM_NUM_PINS > 2
81             set_pin(PWM_PORT_2, PWM_PIN_2);
82         #endif
83     }
84     if (pwm_level[0] == pwm_count) {
85         clear_pin(PWM_PORT_0, PWM_PIN_0);
86     }
87     #if PWM_NUM_PINS > 1
88         if (pwm_level[1] == pwm_count) {
89             clear_pin(PWM_PORT_1, PWM_PIN_1);
90         }
91     #endif
92     #if PWM_NUM_PINS > 2
93         if (pwm_level[2] == pwm_count) {
94             clear_pin(PWM_PORT_2, PWM_PIN_2);
95         }
96     #endif
97 }
```

void pwm_set_level (uns8 *pwm_item*, uns8 *level*)

```
45                                                  {
46
47     pwm_level[pwm_item] = level;
48 }
```

void pwm_set_transition (uns8 *to_level*, uns16 *steps*)

void pwm_setup_io ()

```
58                      {
59
60     make_output(PWM_PORT_0, PWM_PIN_0);
```

```
61      #if PWM_NUM_PINS > 1
62          make_output(PWM_PORT_1, PWM_PIN_1);
63      #endif
64      #if PWM_NUM_PINS > 2
65          make_output(PWM_PORT_2, PWM_PIN_2);
66      #endif
67
68  }
```

Variable Documentation

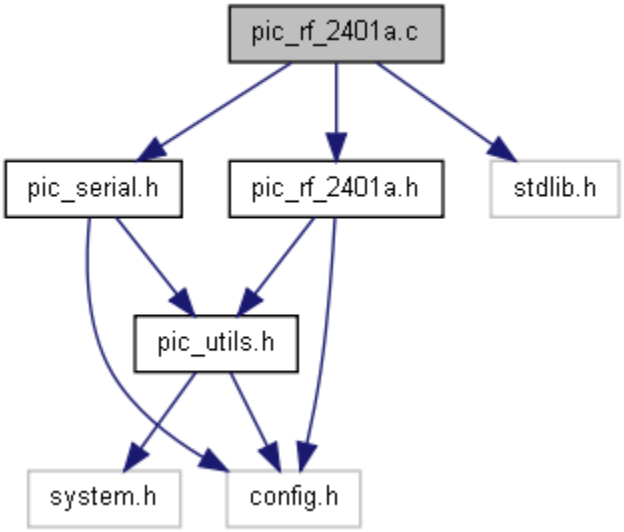
uns8 pwm_count

uns8 pwm_level[PWM_NUM_PINS]

---

pic_rf_2401a.c File Reference

Include dependency graph for pic_rf_2401a.c:



Functions

- void pic_rf_init (rf_config *my_config)
- *Initialise nrf2401a chip with config.* void pic_rf_quick_init (char *my_config, uns8 my_channel, bit my_receive_on)
- *Initialise nrf2401a chip with quick config.* void pic_rf_receive (uns8 *data, uns8 bytes_to_receive)
- *Receive data from nrf2401a.* void pic_rf_send_byte (uns8 b)
- *Internal routine to send a byte to nrf2401a.* void pic_rf_send_bytes (char *bytes, uns8 num_bytes)
- *Internal routine to send bytes to nrf2401a.* void pic_rf_set_channel (uns8 channel)
- *Change channel on the nrf2401a.* void pic_rf_set_mode (uns8 mode)
- *Set rf mode to transmit or receive.* void pic_rf_setup ()
- *Setup ports and pins for communication with nrf2401a.* void pic_rf_transmit (char *data, uns8 bytes_to_transmit)

Transmit data from nrf2401a.

Function Documentation

void pic_rf_init (rf_config * *my_config*)

Initialise nRF24L01 chip with config.

Sends the configuration to the Nordic nrf2401a chip ready to begin communication. This routine assumes you have already set my_config to the correct values.

```
84 {
85 uns8 temp;
86 uns8 options;
87
88     make output(rf_data_port, rf_data_pin); // make data pin output
89     clear pin(rf_clk1_port, rf_clk1_pin);       // make sure it's 0
90
91     pic rf chip enable(0);
92     pic rf chip select(1);  // config mode
93
94     pic rf send byte(my config->payload width ch2);
95     pic rf send byte(my_config->payload width ch1);
96     pic rf send bytes(my_config->address ch2, 5);
97     pic rf send bytes(my_config->address ch1, 5);
98
99     options = my_config->options;
100
101     temp = my_config->address width << 2;
102     temp.1 = options.OP LONG CRC; // |= my config->long crc << 1;
103     temp.0 = options.OP_ENABLE_CRC; // |= my_config->enable_crc;
104
105     pic rf send byte(temp);
106
107     temp = options & 0b11100000;  //pull off top three bits
108     //temp.7 = options.ENABLE_CH2; // |= my_config->enable_ch2 << 7;
109     //temp.6 = options.ENABLE_SHOCKBURST; // |= my_config->enable_shockburst << 6;
110     //temp.5 = options.ENABLE_1_MBPS; //my_config->enable_1_mbps << 5;
111     temp |= (my config->crystal & 0b00000111) << 2; // bits 4,3,2 - mask 3 bit range
112     temp |= (my_config->output power & 0b00000011); // bits 1,0 - mask 2 bit range
113
114     pic rf send byte(temp);
115
116     temp = my_config->channel << 1;
117     rf current channel = my_config->channel;
118
119     temp |= options.OP ENABLE RECEIVE; // my config->enable receive;
120     rf current mode receive = options.OP_ENABLE_RECEIVE;
121
122     pic rf send byte(temp);
123
124     pic rf chip select(0);  // config mode ended
125     pic rf chip enable(1);  // on the air!
126
127   }
```

void pic_rf_quick_init (char * *my_config*, uns8 *my_channel*, bit *my_receive_on*)

While the usual pic_rf_init() routine is excellent when you want to programatically change the 2401a config, if you're only doing this once (at the start) then it's likely you're burning a lot of instructions (154 words on a PIC16 device) just to send some bytes of config out to the 2401a. If you know your config in advance, then you can just send the byte-stream config using this routine. Use the nrf2401a_config.pl script in the tools directory to generate this string.

```
62                                                                                {
63
64 uns8 byte counter;
65     make output(rf_data_port, rf_data_pin); // make data pin output
66     clear pin(rf_clk1_port, rf_clk1_pin);       // make sure it's 0
67
```

```
68      pic_rf_chip_enable(0);
69      pic_rf_chip_select(1);  // config mode
70
71      for(byte_counter = 0 ; byte_counter < 15 ; byte_counter++) {
72          pic_rf_send_byte(my_config[byte_counter]);
73      }
74      rf_current_channel = my_channel;
75      rf_current_mode_receive = my_receive_on;
76
77      pic_rf_chip_select(0);  // config mode ended
78      pic_rf_chip_enable(1);  // on the air!
79
80  }
```

void pic_rf_receive (uns8 * *data*, uns8 *bytes_to_receive*)

Receive data from nRF24L01.

Having been notified that there is data available, call this routine to clock the data in from the nrf2401a.

!pic_rf_chip_enable(0); // save power

pic_rf_chip_enable(1); // turn chip back on

```
130                                                     {
131 uns8 byte_count, bit_count, temp;
132
134     bit  my_store_gie = intcon.GIE;
135     kill_interrupts();
136
137     make_input(rf_data_port, rf_data_pin); // make data pin input
138
139     for (byte_count = 0; byte_count < bytes_to_receive; byte_count++) {
140
141         for (bit_count = 0; bit_count < 8; bit_count++) {
142             temp <<= 1;
143             temp.0 = test_pin(rf_data_port, rf_data_pin);
144             set_pin(rf_clk1_port, rf_clk1_pin);         // clock it out
145             clear_pin(rf_clk1_port, rf_clk1_pin);       // ready for next bit
146         }
147         data[byte_count] =  temp;
148     }
149
151     intcon.GIE = my_store_gie;
152 }
```

void pic_rf_send_byte (uns8 *b*)

Clock a byte into the nRF24L01.

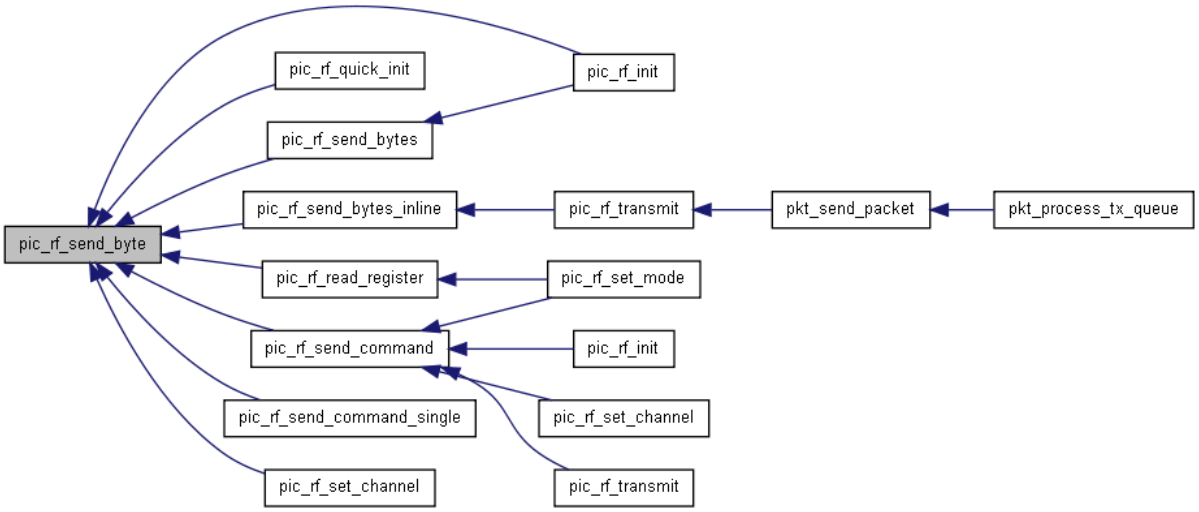
Internal routine to send a byte to the nrf2401a. Generally you shouldn't need to use this, see pic_rf_transmit instead

See also:

pic_rf_transmit

```
42 {
43 uns8 bit_counter;
44      for(bit_counter = 0 ; bit_counter < 8 ; bit_counter++) {
45          change_pin(rf_data_port, rf_data_pin, b.7); // Put data on data pin
46          set_pin(rf_clk1_port, rf_clk1_pin);         // clock it in (positive edge)
47          clear_pin(rf_clk1_port, rf_clk1_pin);       // ready for next bit
48
49          b <<= 1;    // Move all the bits left
50      }   // repeat until finished
51
52  }   // pic_rf_send_byte
```

Here is the caller graph for this function:



void pic_rf_send_bytes (char * *bytes*, uns8 *num_bytes*)

Internal routine to send bytes to the nrf2401a. Generally you shouldn't need to use this, see pic_rf_transmit instead

See also:

pic_rf_transmit

```
54                                                  {
55
56 uns8 byte_counter;
57     for(byte_counter = 0 ; byte_counter < num_bytes ; byte_counter++) {
58         pic_rf_send_byte(bytes[byte_counter]);
59     }
60 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

void pic_rf_set_channel (uns8 *channel*)

Receive data from nRF24L01 (inline).

Reclocks the essential config to change the current channel used by the nrf2401a.

```
200 {
201     bit  my_store_gie = intcon.GIE;
202     kill_interrupts();
203
204     clear_bit(tris_array[rf_data_port - PORTA], rf_data_pin); // make data pin output
205
206     pic_rf_chip_enable(0);
207     pic_rf_chip_select(1);  // config mode
208
209     rf_current_channel = channel;
210     channel <<= 1;
```

```
211     channel |= rf_current_mode_receive;
212
213     pic_rf_send_byte(channel);
214
215     pic_rf_chip_select(0);  // config mode ended
216     pic_rf_chip_enable(1);
217
218     intcon.GIE = my_store_gie;
219 }
```

void pic_rf_set_mode (uns8 *mode*)

Pass RECEIVE_MODE or TRANSMIT_MODE to change current mode. Generally, you shouldn't need to call this routine. The library assumes you want to receive until you transmit, in which case it switches automatically to transmit mode and back to receive afterwards.

```
179 {
180     bit  my_store_gie = intcon.GIE;
181     kill_interrupts();
182
183     make_output(rf_data_port, rf_data_pin); // make data pin output
184     pic_rf_chip_enable(0);
185     pic_rf_chip_select(1);  // config mode
186
187     change_pin(rf_data_port, rf_data_pin, mode);    // send a zero
188     set_pin(rf_clk1_port, rf_clk1_pin);         // clock it in (positive edge)
189     clear_pin(rf_clk1_port, rf_clk1_pin);       // ready for next bit
190
191     pic_rf_chip_select(0);  // config mode ended
192     pic_rf_chip_enable(1);
193
194     rf_current_mode_receive = mode;
195
196     intcon.GIE = my_store_gie;
197 }
```

Here is the caller graph for this function:

void pic_rf_setup ()

Setup ports and pins for communication with nRF24L01.

Set up ports and pins to correct input/output for communication with Nordif nrf2401a

```
221                     {
222
223     make_output(rf_data_port, rf_data_pin); // make data pin output
224     make_output(rf_cs_port, rf_cs_pin); // make cs pin output
225     make_output(rf_ce_port, rf_ce_pin); // make ce pin output
226     make_input (rf_dr1_port, rf_dr1_pin); // make dr1 pin input
227     make_output(rf_clk1_port, rf_clk1_pin); // make clk1 pin output
228
229 }
```

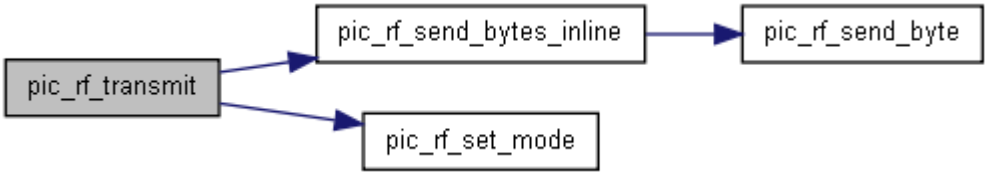
void pic_rf_transmit (char * *data*, uns8 *bytes_to_transmit*)

Changes to transmit mode, clocks data into the nRF2401A and hits the shockburst button. Returns to receive mode when finished.

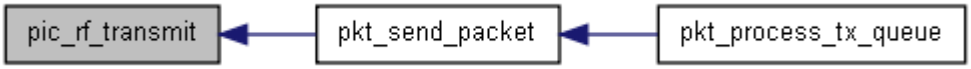
```
154                                                     {
155
156 uns8 byte_count, bit_count, temp;
157
```

```
158     bit  my_store_gie = intcon.GIE;
159     kill_interrupts();
160 
161     if (rf_current_mode_receive) {
162         pic_rf_set_mode(TRANSMIT_MODE);
163     }
164 
165     make_output(rf_data_port, rf_data_pin); // make data pin output
166 
167     pic_rf_send_bytes_inline(data, bytes_to_transmit);
168 
169     pic_rf_chip_enable(0);  // On low, enable shockburst
170 
171     delay_ms(1);
172 
173     pic_rf_set_mode(RECEIVE_MODE);  // Go back to receive mode
174 
175     intcon.GIE = my_store_gie;
176 }
```

Here is the call graph for this function:



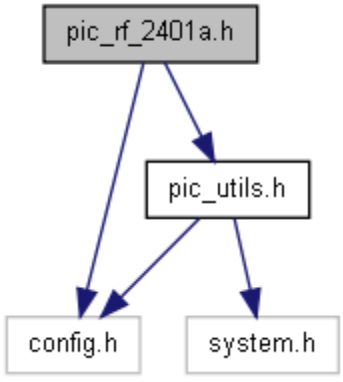
Here is the caller graph for this function:



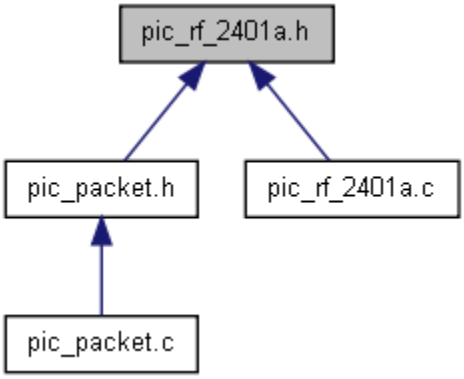
---

pic_rf_2401a.h File Reference

Pic Nordic nrf2401a routines.
Include dependency graph for pic_rf_2401a.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct rf_config

Defines

- #define OP_ENABLE_1_MBPS 5
- #define OP_ENABLE_CH2 7
- #define OP_ENABLE_CRC 2
- #define OP_ENABLE_RECEIVE 0
- #define OP_ENABLE_SHOCKBURST 6
- #define OP_LONG_CRC 1
- #define pic_rf_chip_enable(value) change_pin(rf_ce_port, rf_ce_pin, value);
- #define pic_rf_chip_select(value) change_pin(rf_cs_port, rf_cs_pin, value);
- #define pic_rf_init_inline(my_config)
- *Inline version of the pic_rf_init routine.* #define pic_rf_receive_mode() pic_rf_set_mode(RECEIVE_MODE)
- #define pic_rf_transmit_mode() pic_rf_set_mode(TRANSMIT_MODE)
- #define RECEIVE_MODE 1
- #define TRANSMIT_MODE 0

Functions

- void pic_rf_init (rf_config *my_config)
- *Initialise nrf2401a chip with config.* void pic_rf_quick_init (char *my_config, uns8 my_channel, bit my_receive_on)
- *Initialise nrf2401a chip with quick config.* void pic_rf_receive (uns8 *data, uns8 bytes_to_receive)
- *Receive data from nrf2401a.* void pic_rf_send_byte (uns8 b)
- *Internal routine to send a byte to nrf2401a.* void pic_rf_send_bytes (char *bytes, uns8 num_bytes)
- *Internal routine to send bytes to nrf2401a.* void pic_rf_send_bytes_inline (char *bytes, uns8 num_bytes)
- *Inline version of send_bytes.* void pic_rf_set_channel (uns8 channel)
- *Change channel on the nrf2401a.* void pic_rf_set_mode (uns8 mode)
- *Set rf mode to transmit or receive.* void pic_rf_setup ()
- *Setup ports and pins for communication with nrf2401a.* void pic_rf_transmit (char *data, uns8 bytes_to_transmit)

Transmit data from nrf2401a. Variables

- static uns8 rf_current_channel = 2
- static bit rf_current_mode_receive = 0

Detailed Description

Nordic nrf2401a routines

Define Documentation

#define OP_ENABLE_1_MBPS 5

rf_config options - enable 1 MPS transmition (otherwise 250Kbps)

#define OP_ENABLE_CH2 7

rf_config options - enable channel 2 reception

#define OP_ENABLE_CRC 2

rf_config options - enable CRC (recommended!)

#define OP_ENABLE_RECEIVE 0

rf_config options - enable receive

#define OP_ENABLE_SHOCKBURST 6

rf_config options - enable shockburst transmition

#define OP_LONG_CRC 1

rf_config options - enable 16 bit CRC (otherwise 8 bit CRC if 0)

#define pic_rf_chip_enable(value) change_pin(rf_ce_port, rf_ce_pin, value);

Set chip enable line (CE) to the given value

#define pic_rf_chip_select(value) change_pin(rf_cs_port, rf_cs_pin, value);

Set chip select line (CS) to the given value

#define pic_rf_init_inline(my_config)

Depending upon your chip, if you're using the programmatic method of configuring the nrf2401a you may find that this routine helps you use less flash and/or stack space

#define pic_rf_receive_mode() pic_rf_set_mode(RECEIVE_MODE)

Change to receive mode

#define pic_rf_transmit_mode() pic_rf_set_mode(TRANSMIT_MODE)

Change to transmit mode

#define RECEIVE_MODE 1

Mode selection - receive

#define TRANSMIT_MODE 0

Mode selection - transmit

Function Documentation

void pic_rf_init (rf_config * *my_config*)

Sends the configuration to the Nordic nrf2401a chip ready to begin communication. This routine assumes you have already set my_config to the correct values.

```
84 {
85 uns8 temp;
86 uns8 options;
87
88     make_output(rf_data_port, rf_data_pin); // make data pin output
89     clear_pin(rf_clk1_port, rf_clk1_pin);       // make sure it's 0
90
91     pic_rf_chip_enable(0);
92     pic_rf_chip_select(1);  // config mode
93
94     pic_rf_send_byte(my_config->payload_width_ch2);
95     pic_rf_send_byte(my_config->payload_width_ch1);
96     pic_rf_send_bytes(my_config->address_ch2, 5);
97     pic_rf_send_bytes(my_config->address_ch1, 5);
98
99     options = my_config->options;
100
101     temp = my_config->address_width << 2;
102     temp.1 = options.OP_LONG_CRC; // |= my_config->long_crc << 1;
103     temp.0 = options.OP_ENABLE_CRC; // |= my_config->enable_crc;
104
105     pic_rf_send_byte(temp);
106
107     temp = options & 0b11100000;  //pull off top three bits
108     //temp.7 = options.ENABLE_CH2; // |= my_config->enable_ch2 << 7;
109     //temp.6 = options.ENABLE_SHOCKBURST; // |= my_config->enable_shockburst << 6;
110     //temp.5 = options.ENABLE_1_MBPS; //my_config->enable_1_mbps << 5;
111     temp |= (my_config->crystal & 0b00000111) << 2; // bits 4,3,2 - mask 3 bit range
112     temp |= (my_config->output_power & 0b00000011); // bits 1,0 - mask 2 bit range
113
114     pic_rf_send_byte(temp);
115
116     temp = my_config->channel << 1;
117     rf_current_channel = my_config->channel;
118
119     temp |= options.OP_ENABLE_RECEIVE; // my_config->enable_receive;
120     rf_current_mode_receive = options.OP_ENABLE_RECEIVE;
121
122     pic_rf_send_byte(temp);
123
124     pic_rf_chip_select(0);  // config mode ended
125     pic_rf_chip_enable(1);  // on the air!
126
127   }
```

void pic_rf_quick_init (char * *my_config*, uns8 *my_channel*, bit *my_receive_on*)

While the usual pic_rf_init() routine is excellent when you want to programatically change the 2401a config, if you're only doing this once (at the start) then it's likely you're burning a lot of instructions (154 words on a PIC16 device) just to send some bytes of config out to the 2401a. If you know your config in advance, then you can just send the byte-stream config using this routine. Use the nrf2401a_config.pl script in the tools directory to generate this string.

```
62                                                                          {
63
64 uns8 byte_counter;
65     make_output(rf_data_port, rf_data_pin); // make data pin output
66     clear_pin(rf_clk1_port, rf_clk1_pin);       // make sure it's 0
67
68     pic_rf_chip_enable(0);
69     pic_rf_chip_select(1);  // config mode
```

```
70
71      for(byte_counter = 0 ; byte_counter < 15 ; byte_counter++) {
72          pic rf send byte(my config[byte counter]);
73      }
74      rf current channel = my_channel;
75      rf current mode receive = my_receive_on;
76
77      pic rf chip select(0);  // config mode ended
78      pic rf chip enable(1);  // on the air!
79
80  }
```

void pic_rf_receive (uns8 * *data*, uns8 *bytes_to_receive*)

Having been notified that there is data available, call this routine to clock the data in from the nrf2401a.
!pic_rf_chip_enable(0); // save power
pic_rf_chip_enable(1); // turn chip back on

```
130                                                     {
131 uns8 byte_count, bit_count, temp;
132
134     bit  my_store_gie = intcon.GIE;
135     kill interrupts();
136
137     make input(rf data port, rf data pin); // make data pin input
138
139     for (byte_count = 0; byte_count < bytes_to_receive; byte_count++) {
140
141         for (bit_count = 0; bit_count < 8; bit_count++) {
142             temp <<= 1;
143             temp.0 = test pin(rf_data_port, rf_data_pin);
144             set pin(rf_clk1_port, rf_clk1_pin);         // clock it out
145             clear pin(rf clk1 port, rf clk1 pin);       // ready for next bit
146         }
147         data[byte_count] =  temp;
148     }
149
151     intcon.GIE = my_store_gie;
152 }
```

void pic_rf_send_byte (uns8 *b*)

Internal routine to send a byte to the nrf2401a. Generally you shouldn't need to use this, see pic_rf_transmit instead

See also:

pic_rf_transmit

```
42 {
43 uns8 bit_counter;
44      for(bit_counter = 0 ; bit_counter < 8 ; bit_counter++) {
45          change pin(rf_data_port, rf_data_pin, b.7); // Put data on data pin
46          set pin(rf_clk1_port, rf_clk1_pin);         // clock it in (positive edge)
47          clear pin(rf_clk1_port, rf_clk1_pin);       // ready for next bit
48
49          b <<= 1;    // Move all the bits left
50      }   // repeat until finished
51
52 }   // pic_rf_send_byte
```

void pic_rf_send_bytes (char * *bytes*, uns8 *num_bytes*)

Internal routine to send bytes to the nrf2401a. Generally you shouldn't need to use this, see pic_rf_transmit instead

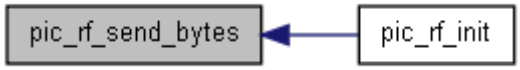
See also:

pic_rf_transmit

```
54                                                  {
55
56 uns8 byte_counter;
57     for(byte_counter = 0 ; byte_counter < num_bytes ; byte_counter++) {
58         pic_rf_send_byte(bytes[byte_counter]);
59     }
60 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void pic_rf_send_bytes_inline (char * *bytes*, uns8 *num_bytes*) [inline]

Inline version of pc_rf_send_bytes. This is an internal routine, and generally you shouldn't need to call it. Use pic_rf_transmit instead.

See also:

pic_rf_transmit

```
273                                                         {
274
275 uns8 byte_counter;
276     for(byte_counter = 0 ; byte_counter < num_bytes ; byte_counter++) {
277         pic_rf_send_byte(bytes[byte_counter]);
278     }
279 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

void pic_rf_set_channel (uns8 *channel*)

Reclocks the essential config to change the current channel used by the nrf2401a.

```
200 {
201     bit  my_store_gie = intcon.GIE;
202     kill_interrupts();
203
204     clear_bit(tris_array[rf_data_port - PORTA], rf_data_pin); // make data pin output
205
206     pic_rf_chip_enable(0);
207     pic_rf_chip_select(1);  // config mode
```

```
208
209     rf_current_channel = channel;
210     channel <<= 1;
211     channel |= rf_current_mode_receive;
212
213     pic_rf_send_byte(channel);
214
215     pic_rf_chip_select(0);  // config mode ended
216     pic_rf_chip_enable(1);
217
218     intcon.GIE = my_store_gie;
219 }
```

void pic_rf_set_mode (uns8 *mode*)

Pass RECEIVE_MODE or TRANSMIT_MODE to change current mode. Generally, you shouldn't need to call this routine. The library assumes you want to receive until you transmit, in which case it switches automatically to transmit mode and back to receive afterwards.

```
179 {
180     bit  my_store_gie = intcon.GIE;
181     kill_interrupts();
182
183     make_output(rf_data_port, rf_data_pin); // make data pin output
184     pic_rf_chip_enable(0);
185     pic_rf_chip_select(1);  // config mode
186
187     change_pin(rf_data_port, rf_data_pin, mode);    // send a zero
188     set_pin(rf_clk1_port, rf_clk1_pin);         // clock it in (positive edge)
189     clear_pin(rf_clk1_port, rf_clk1_pin);       // ready for next bit
190
191     pic_rf_chip_select(0);  // config mode ended
192     pic_rf_chip_enable(1);
193
194     rf_current_mode_receive = mode;
195
196     intcon.GIE = my_store_gie;
197 }
```

void pic_rf_setup ()

Set up ports and pins to correct input/output for communication with Nordif nrf2401a

```
221                     {
222
223     make_output(rf_data_port, rf_data_pin); // make data pin output
224     make_output(rf_cs_port, rf_cs_pin); // make cs pin output
225     make_output(rf_ce_port, rf_ce_pin); // make ce pin output
226     make_input (rf_dr1_port, rf_dr1_pin); // make dr1 pin input
227     make_output(rf_clk1_port, rf_clk1_pin); // make clk1 pin output
228
229 }
```

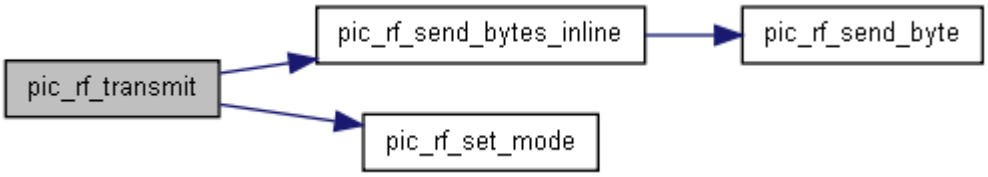
void pic_rf_transmit (char * *data*, uns8 *bytes_to_transmit*)

Changes to transmit mode, clocks data into the nRF2401A and hits the shockburst button. Returns to receive mode when finished.

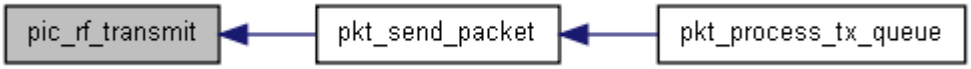
```
154                                                         {
155
156 uns8 byte_count, bit_count, temp;
157
158     bit  my_store_gie = intcon.GIE;
159     kill_interrupts();
160
161     if (rf_current_mode_receive) {
162         pic_rf_set_mode(TRANSMIT_MODE);
```

```
163     }
164
165     make_output(rf_data_port, rf_data_pin); // make data pin output
166
167     pic_rf_send_bytes_inline(data, bytes_to_transmit);
168
169     pic_rf_chip_enable(0);  // On low, enable shockburst
170
171     delay_ms(1);
172
173     pic_rf_set_mode(RECEIVE_MODE);  // Go back to receive mode
174
175     intcon.GIE = my_store_gie;
176 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



Variable Documentation

uns8 rf_current_channel = 2 [static]

Maintain state of current channel

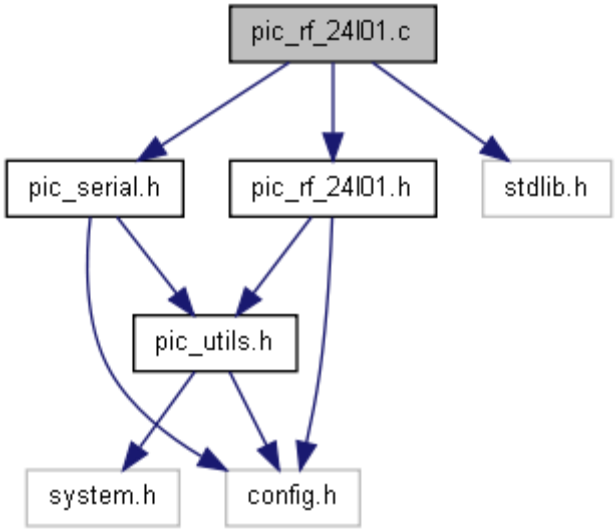
bit rf_current_mode_receive = 0 [static]

Maintain state of current mode (1 = receive mode)

---

pic_rf_24l01.c File Reference

Include dependency graph for pic_rf_24l01.c:



Functions

- void pic_rf_init (rf_config *my_config)
- *Initialise nrf2401a chip with config.* void pic_rf_quick_init (char *my_config, uns8 my_channel, bit my_receive_on)
- *Initialise nrf2401a chip with quick config.* uns8 pic_rf_read_register (uns8 cmd, uns8 *data, uns8 data_len)
- *Read nRF24L01 register.* uns8 pic_rf_read_register_int (uns8 cmd, uns8 *data, uns8 data_len)
- uns8 pic_rf_receive (uns8 *data, uns8 bytes_to_receive)
- *Receive data from nrf2401a.* void pic_rf_receive2 (uns8 *data, uns8 bytes_to_receive)
- void pic_rf_receive_inline (uns8 *data, uns8 bytes_to_receive)
- *Receive data from nRF24L01.* uns8 pic_rf_send_byte (uns8 b)
- *Internal routine to send a byte to nrf2401a.* uns8 pic_rf_send_byte_int (uns8 b)
- *Clock a byte into the nRF24L01.* uns8 pic_rf_send_command (uns8 cmd, uns8 *data, uns8 data_len)
- *Send command to the nrf24l01.* uns8 pic_rf_send_command_single (uns8 cmd, uns8 data)
- *Send a single data byte command to the nrf24l01.* void pic_rf_set_channel (uns8 channel)
- *Change channel on the nrf2401a.* void pic_rf_set_mode (uns8 requested_mode)
- *Set rf mode to transmit or receive.* void pic_rf_setup ()
- *Setup ports and pins for communication with nrf2401a.* void pic_rf_transmit (uns8 *data, uns8 bytes_to_transmit)

Transmit data from nRF24L01.

Function Documentation

void pic_rf_init (rf_config * *my_config*)

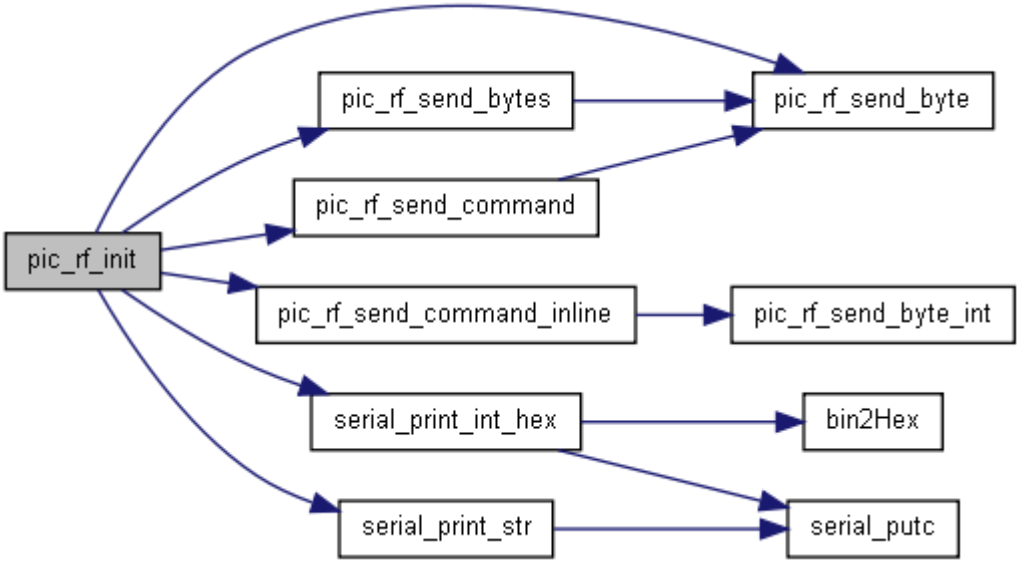
Initialise nRF24L01 chip with config.

Sends the configuration to the Nordic nrf2401a chip ready to begin communication. This routine assumes you have already set my_config to the correct values.

```
254 {
255 uns8 temp;
256 uns8 options;
257 uns8 data[5];
258
259     start_crit_sec();
260
261     clear_pin(rf_ce_port, rf_ce_pin); // go into standby, so we can configure
262     delay_ms(100);
263     //data[0] = 0b00111100;
```

```
264     temp = pic_rf_send_command(RF_WR_REG_CONFIG_REG, // write register 0x00 = CONFIG_REG
265                           "\x3c", 1); //
7-reserved=0,6-MASK RX DR=0,5-MASK TX DS=1,4-MASK MAX RT=1,
266                                   // 3-EN_CRC=1,2-CRC0=1 (1=2 byte)
267                                   // 1-PWR_UP=0,0-PRIM_RX=0
268     //data[0] = 0b00000000;
269     pic_rf_send_command(RF_WR_REG_SETUP_RETR, // write register 0x04 = SETUP_RETR
270                           "\x00", 1);     // 7-4 retry delay=0,
271                                           // 3-0 Auto retransmit count=0
272     //data[0] =  0b00000001;
273     pic_rf_send_command(RF_WR_REG_SETUP_AW, // write register 0x03 = SETUP AW Address widths
274                           "\x01", 1);     // 7-2 reserved=0, 1-0 *01*-3 bytes 10-4 bytes 11-5
bytes
275
276     //data[0] = 0b00000111;
277     pic_rf_send_command(RF_WR_REG_RF_SETUP, // write register 6 = RF_SETUP
278                           "\x07", 1);     // 7-5 reserved=0, 4 PLL_LOCK=0, 3 RF_DR=0, 2-1
RF_PWR=11, 0 LNA_HCURR=1
279     //data[0] = 0b00000010;
280     pic_rf_send_command (RF_WR_REG_RF_CH,   // write register 0x05 = RF_CH
281                           "\x02", 1);    // 7 reserved=0, 6:0 RF_CH = 2
282
283     data[0] = 0b11100111;   // 0xe7 LSB
284     data[1] = 0b11100111;   // 0xe7
285     data[2] = 0b11100111;   // 0xe7 MSB
286     pic_rf_send_command(RF_WR_REG_TX_ADDR,  // write register 0x10 TX_ADDR
287                           &data, 3);
288
289
290     pic_rf_send_command(RF_WR_REG_RX_ADDR_P0, // write register 0x0a = RX_ADDR_P0
291                           &data, 3);
292
293     pic_rf_send_command(RF_WR_REG_EN_AA, // write register 0x01 = EN_AA
294                       "\x00", 1);  // 7:6 reserved=0, 5 ENAA_P5=0, 4 ENAA_P4=0, ... 0 ENAA_P0=0
295
296     // Set payload to 21 bytes
297     // 0b00010101
298     //PKT_PACKET_SIZE
299     pic_rf_send_command(RF_WR_REG_RX_PW_P0,  // write register 0x11 = RX_PW_P0 receive
payload width
300                           "\x15", 1);     // 7:6 reserved=0 5:0 bytes in payload
301     // 0b00111111
302     pic_rf_send_command(RF_WR_REG_CONFIG_REG,  // write register 0x00 = CONFIG_REG
303                           "\x3f", 1); //
7-reserved=0,6-MASK_RX_DR=0,5-MASK_TX_DS=1,4-MASK_MAX_RT=1,
304                                   // 3-EN_CRC=1,2-CRC0=1 (1=2 byte)
305                                   // 1-PWR_UP=1,0-PRIM_RX=1
306     delay_ms(2);    // 1.5ms settling after power up
307     pic_rf_send_command (RF_FLUSH_TX,
308                           0, 0   );
309     pic_rf_send_command (RF_FLUSH_RX,
310                           0, 0   );
311     //clear interrupts
312     pic_rf_send_command_inline(RF_WR_REG_STATUS, "\x40", 1);
313     delay_ms(2);    // 1.5ms settling after power up
314
315     set_pin(rf_ce_port, rf_ce_pin); // We're on the air (receive)
316
317     end_crit_sec();
318     serial_print_str("got=");
319     serial_print_int_hex(temp);
320     serial_print_str(" ");
321
322     rf_current_mode_receive = 1;
323  }
```

Here is the call graph for this function:



void pic_rf_quick_init (char * *my_config*, uns8 *my_channel*, bit *my_receive_on*)

While the usual pic_rf_init() routine is excellent when you want to programatically change the 2401a config, if you're only doing this once (at the start) then it's likely you're burning a lot of instructions (154 words on a PIC16 device) just to send some bytes of config out to the 2401a. If you know your config in advance, then you can just send the byte-stream config using this routine. Use the nrf2401a_config.pl script in the tools directory to generate this string.

```
157                                                                     {
158
159
160 }
```

Here is the call graph for this function:

uns8 pic_rf_read_register (uns8 *cmd*, uns8 * *data*, uns8 *data_len*)

Internal routine to read a particular nRF24L01 register. Clocks out data_len bytes from the chip. Internal routine.

Parameters:

cmd Read register command, eg RF_RD_REG_STATUS
data Pointer to array of bytes where data will be put
data_len Number of bytes to clock out

```
92                                                              { // returns status
93
94 uns8 byte_counter, status;
95
96     clear_pin(rf_csn_port, rf_csn_pin);
97     status = pic_rf_send_byte(cmd);
98     for(byte_counter = 0 ; byte_counter < data_len ; byte_counter++) {
99         data[byte_counter] = pic_rf_send_byte(0);
100     }
101
102     set_pin(rf_csn_port, rf_csn_pin);
103     return status;
```

```
104 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

uns8 pic_rf_read_register_int (uns8 *cmd*, uns8 * *data*, uns8 *data_len*)

```
106                                                                 { // returns status
107
108 uns8 byte_counter, status;
109
110     clear_pin(rf_csn_port, rf_csn_pin);
111     status = pic_rf_send_byte_int(cmd);
112     for(byte_counter = 0 ; byte_counter < data_len ; byte_counter++) {
113         data[byte_counter] = pic_rf_send_byte_int(0);
114     }
115
116     set_pin(rf_csn_port, rf_csn_pin);
117     return status;
118 }
```

Here is the call graph for this function:

uns8 pic_rf_receive (uns8 * *data*, uns8 *bytes_to_receive*)

Receive data from nRF24L01.

Having been notified that there is data available, call this routine to clock the data in from the nrf2401a.

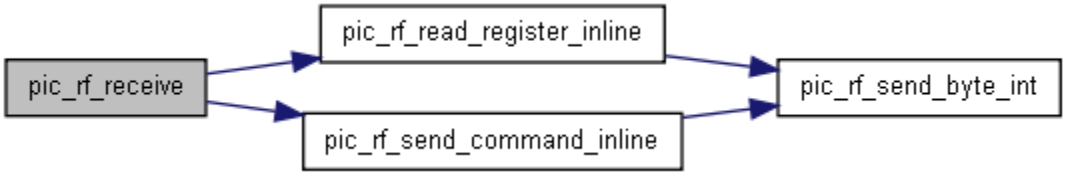
!pic_rf_chip_enable(0); // save power

pic_rf_chip_enable(1); // turn chip back on

```
41                                                      {
42
43 uns8 fifo_status;
44 uns8 res;
45
46     res = 0;
47
48     // Could check status here, but we know why it is...
49     //clear_pin(rf_ce_port, rf_ce_pin); // Go into standby
50
51     pic_rf_read_register_inline(RF_RD_REG_FIFO_STATUS, &fifo_status, 1);
52     while (!test_bit(fifo_status, 0)) { // Fifo has something in it
53         //serial_putc('|');
54         pic_rf_read_register_inline(RF_R_RX_PAYLOAD, data, bytes_to_receive);   // receive
55         pic_rf_send_command_inline(RF_WR_REG_STATUS, "\x40", 1);
56         res++;
57         pic_rf_read_register_inline(RF_RD_REG_FIFO_STATUS, &fifo_status, 1);
58     }
59     // clear rf interrupt
```

```
60      //pic_rf_send_command_inline (RF_FLUSH_RX,  0, 0 );
61      //set_pin(rf_ce_port, rf_ce_pin); // Back on the air
62      return res;
63  }
```

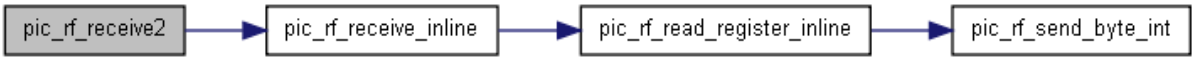
Here is the call graph for this function:



void pic_rf_receive2 (uns8 * *data*, uns8 *bytes_to_receive*)

```
352                                                     {
353
354     pic_rf_receive_inline(data, bytes_to_receive);
355 }
```

Here is the call graph for this function:

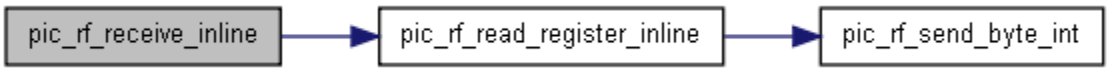


void pic_rf_receive_inline (uns8 * *data*, uns8 *bytes_to_receive*) `[inline]`

Having been notified that there is data available, call this routine to clock the data in from the nRF24L01.

```
348                                                                 {
349         pic_rf_read_register_inline(RF_R_RX_PAYLOAD, data, bytes_to_receive);
350 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

uns8 pic_rf_send_byte (uns8 *b*)

Clock a byte into the nRF24L01.

Internal routine to send a byte to the nrf2401a. Generally you shouldn't need to use this, see pic_rf_transmit instead

See also:

pic_rf_transmit

```
124 {
125 uns8 bit_counter, status;
126     // - For debug: print_int_hex(b); putc(' ');
127     for(bit_counter = 0 ; bit_counter < 8 ; bit_counter++) {
128         change_pin(rf_mosi_port, rf_mosi_pin, b.7); // Put data on data pin
```

```
129         set_pin(rf_sck_port, rf_sck_pin);           // clock it in (positive edge)
130         status <<= 1;
131         status.0 = test_pin(rf_miso_port, rf_miso_pin);
132         clear_pin(rf_sck_port, rf_sck_pin);     // ready for next bit
133
134         b <<= 1;    // Move all the bits left
135     }   // repeat until finished
136     return status;
137 }   // pic_rf_send_byte
```

uns8 pic_rf_send_byte_int (uns8 *b*)

Clock one byte into the nRF24L01. Internal routine.

Parameters:

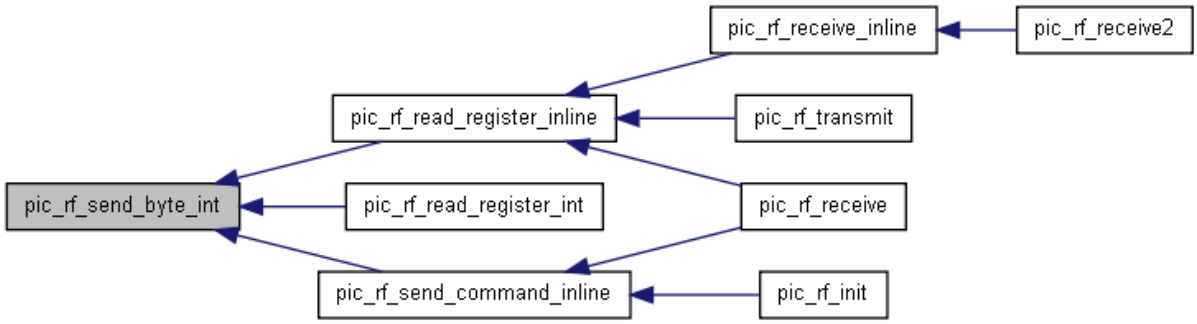
b The byte to send

Returns:

nRF24L01 status

```
141 {
142 uns8 bit_counter, status;
143     // - For debug: print_int_hex(b); putc(' ');
144     for(bit_counter = 0 ; bit_counter < 8 ; bit_counter++) {
145         change_pin(rf_mosi_port, rf_mosi_pin, b.7); // Put data on data pin
146         set_pin(rf_sck_port, rf_sck_pin);           // clock it in (positive edge)
147         status <<= 1;
148         status.0 = test_pin(rf_miso_port, rf_miso_pin);
149         clear_pin(rf_sck_port, rf_sck_pin);     // ready for next bit
150
151         b <<= 1;    // Move all the bits left
152     }   // repeat until finished
153     return status;
154 }   // pic_rf_send_byte
```

Here is the caller graph for this function:



uns8 pic_rf_send_command (uns8 *cmd*, uns8 * *data*, uns8 *data_len*)

Send a command and associated data to the nRF24L01

Parameters:

cmd Command to send, eg, RF_WR_REG_SETUP_RETR
data Pointer to an array of bytes to send as data for the command
data_len Number of bytes in the array

Returns:

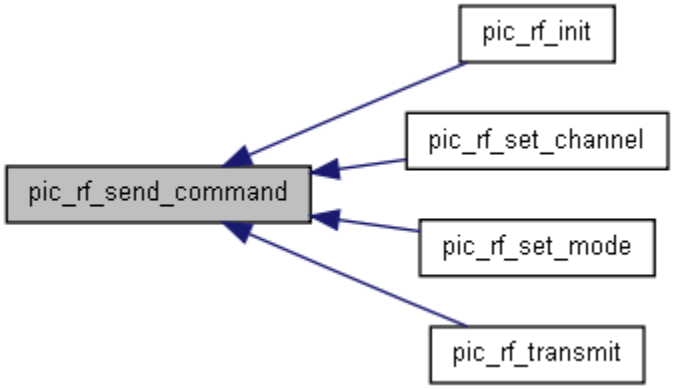
nRF24L01 status

```
66                                                          {
67
68 uns8 byte_counter, status;
69
70     clear_pin(rf_csn_port, rf_csn_pin);
```

```
71      status = pic_rf_send_byte(cmd);
72      for(byte_counter = 0 ; byte_counter < data_len ; byte_counter++) {
73          pic_rf_send_byte(data[byte_counter]);
74      }
75
76      set_pin(rf_csn_port, rf_csn_pin);
77      return status;
78  }
```

Here is the call graph for this function:

Here is the caller graph for this function:



uns8 pic_rf_send_command_single (uns8 *cmd*, uns8 *data*)

Send a command and 1 byte of data to the nRF24L01

Parameters:

cmd Command to send, eg, RF_WR_REG_SETUP_RETR
data One byte of data for the command

Returns:

nRF24L01 status

```
80                                                      {
81
82  uns8 byte_counter, status;
83
84      clear_pin(rf_csn_port, rf_csn_pin);
85      status = pic_rf_send_byte(cmd);
86      pic_rf_send_byte(data);
87      set_pin(rf_csn_port, rf_csn_pin);
88      return status;
89  }
```

Here is the call graph for this function:

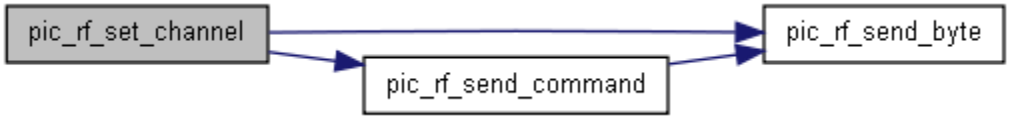
void pic_rf_set_channel (uns8 *channel*)

Receive data from nRF24L01 (inline).

Reclocks the essential config to change the current channel used by the nrf2401a.

```
386 {
387     start_crit_sec();
388 
389     clear_pin(rf_ce_port, rf_ce_pin);
390     pic_rf_send_command (RF_WR_REG_RF_CH,    // write register 0x05 = RF_CH
391                          &channel, 1);       // 7 reserved=0, 6:0 RF_CH = channel
392     if (rf_current_mode_receive) {
393         set_pin(rf_ce_port, rf_ce_pin); // Go back on the air!
394     }
395     rf_current_channel = channel;
396 
397     end_crit_sec();
398 }
```

Here is the call graph for this function:

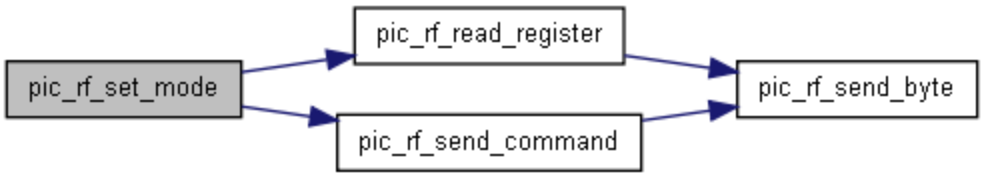


void pic_rf_set_mode (uns8 *mode*)

Pass RECEIVE_MODE or TRANSMIT_MODE to change current mode. Generally, you shouldn't need to call this routine. The library assumes you want to receive until you transmit, in which case it switches automatically to transmit mode and back to receive afterwards.

```
359 {
360 uns8 config_reg;
361 
362     start_crit_sec();
363 
364     if ((requested_mode == TRANSMIT_MODE) && (rf_current_mode_receive)) {
365         // RX -> TX
366         // need to lower CE to go into stand by mode
367         clear_pin(rf_ce_port, rf_ce_pin); // Go into standby
368         pic_rf_read_register(RF_RD_REG_CONFIG_REG, &config_reg, 1);
369         clear_bit(config_reg, CONFIG_PRIM_RX);
370         pic_rf_send_command(RF_WR_REG_CONFIG_REG, &config_reg, 1);
371         rf_current_mode_receive = 0;
372     } else if ((requested_mode == RECEIVE_MODE) && (!rf_current_mode_receive)) {
373         // TX -> RX
374         // CE should already be low, so we're in standby mode
375         pic_rf_read_register(RF_RD_REG_CONFIG_REG, &config_reg, 1);
376         set_bit(config_reg, CONFIG_PRIM_RX);
377         pic_rf_send_command(RF_WR_REG_CONFIG_REG, &config_reg, 1);
378         set_pin(rf_ce_port, rf_ce_pin); // we're on the air! (rx)
379         rf_current_mode_receive = 1;
380     }
381 
382     end_crit_sec();
383 }
```

Here is the call graph for this function:



void pic_rf_setup ()

Setup ports and pins for communication with nRF24L01.

Set up ports and pins to correct input/output for communication with Nordif nrf2401a

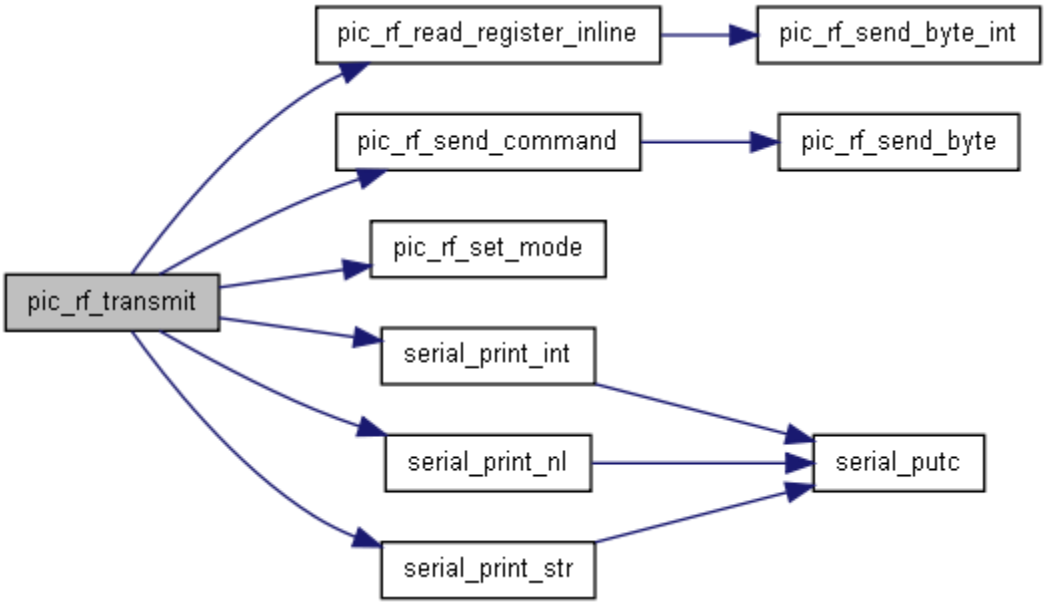
```
400                     {
401
402     make_output(rf_ce_port,   rf_ce_pin);
403     make_output(rf_csn_port,  rf_csn_pin);
404     make_output(rf_sck_port,  rf_sck_pin);
405     make_output(rf_mosi_port, rf_mosi_pin);
406     make_input (rf_miso_port, rf_miso_pin);
407     make_input (rf_irq_port,  rf_irq_pin);
408
409     set_pin(rf_csn_port, rf_csn_pin);
410     clear_pin(rf_ce_port, rf_ce_pin);
411 }
```

void pic_rf_transmit (uns8 * *data*, uns8 *bytes_to_transmit*)

Changes to transmit mode, clocks data into the nrf24L01 and hits the shockburst button. Returns to receive mode when finished.

```
326                                                             {
327
328 uns8 byte_count, bit_count, temp, cd;
329     start_crit_sec();
330
331     pic_rf_set_mode(TRANSMIT_MODE);
332
333     //pic_rf_send_command (RF_FLUSH_TX, 0, 0    );
334     pic_rf_read_register_inline(RF_RD_REG_CD, &cd, 1);
335     serial_print_str("\n cd=");
336     serial_print_int(cd);
337     serial_print_nl();
338     pic_rf_send_command(RF_W_TX_PAYLOAD, data, bytes_to_transmit);
339
340     set_pin(rf_ce_port, rf_ce_pin);
341     delay_us(10);   // 10us pulse - send packet out on airways
342     clear_pin(rf_ce_port, rf_ce_pin);
343     delay_us(130);  // TX settling
344     pic_rf_set_mode(RECEIVE_MODE);  // go back to receive mode
345
346     end_crit_sec();
347 }
```

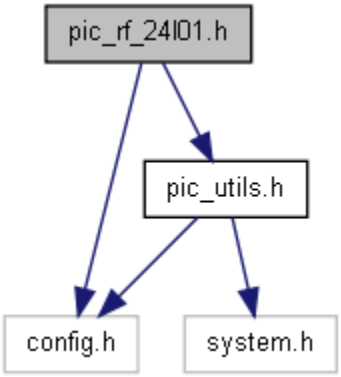
Here is the call graph for this function:



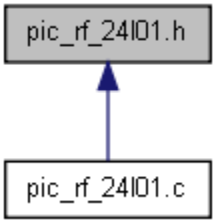
---

pic_rf_24l01.h File Reference

RF routines for the Nordic nRF24L01 chip.
Include dependency graph for pic_rf_24l01.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct rf_config

Defines

- #define CONFIG_CRCO 2
- #define CONFIG_EN_CRC 3
- #define CONFIG_MASK_MAX_RT 4
- #define CONFIG_MASK_RX_DR 6
- #define CONFIG_MASK_TX_DS 5
- #define CONFIG_PRIM_RX 0
- #define CONFIG_PWR_UP 1
- #define FIFO_STATUS_RX_EMPTY 0
- #define FIFO_STATUS_RX_FULL 1
- #define FIFO_STATUS_TX_EMPTY 4
- #define FIFO_STATUS_TX_FULL 5
- #define FIFO_STATUS_TX_REUSE 6
- #define OP_ENABLE_1_MBPS 5
- #define OP_ENABLE_CH2 7
- #define OP_ENABLE_CRC 2
- #define OP_ENABLE_RECEIVE 0
- #define OP_ENABLE_SHOCKBURST 6
- #define OP_LONG_CRC 1
- #define pic_rf_get_status() pic_rf_read_register(RF_NOP, 0, 0)
- #define pic_rf_receive_mode() pic_rf_set_mode(RECEIVE_MODE)
- #define pic_rf_set_status(status) pic_rf_send_command(RF_WR_REG_STATUS, status, 1)
- #define pic_rf_transmit_mode() pic_rf_set_mode(TRANSMIT_MODE)
- #define RECEIVE_MODE 1
- #define RF_FLUSH_RX 0b11100010
- #define RF_FLUSH_TX 0b11100001
- #define RF_NOP 0b11111111
- #define RF_R_RX_PAYLOAD 0b01100001
- #define RF_RD_REG_CD 0b00001001
- #define RF_RD_REG_CONFIG_REG 0b00000000
- #define RF_RD_REG_FIFO_STATUS 0b00010111
- #define RF_RD_REG_RX_PW_P0 0b00010001
- #define RF_RD_REG_STATUS 0b00000111
- #define RF_W_TX_PAYLOAD 0b10100000
- #define RF_WR_REG_CONFIG_REG 0b00100000
- #define RF_WR_REG_EN_AA 0b00100001
- #define RF_WR_REG_RF_CH 0b00100101
- #define RF_WR_REG_RF_SETUP 0b00100110
- #define RF_WR_REG_RX_ADDR_P0 0b00101010
- #define RF_WR_REG_RX_PW_P0 0b00110001
- #define RF_WR_REG_SETUP_AW 0b00100011
- #define RF_WR_REG_SETUP_RETR 0b00100100
- #define RF_WR_REG_STATUS 0b00100111
- #define RF_WR_REG_TX_ADDR 0b00110000
- #define STATUS_MAX_RT 4
- #define STATUS_RX_DR 6
- #define STATUS_TX_DS 5
- #define STATUS_TX_FULL 0
- #define TRANSMIT_MODE 0

Functions

- void pic_rf_init (rf_config *my_config)
- *Initialise nRF24L01 chip with config.* void pic_rf_quick_init (char *my_config, uns8 my_channel, bit my_receive_on)
- *Initialise nrf2401a chip with quick config.* uns8 pic_rf_read_register (uns8 cmd, uns8 *data, uns8 data_len)
- *Read nRF24L01 register.* uns8 pic_rf_read_register_inline (uns8 cmd, uns8 *data, uns8 data_len)
- *Read nRF24L01 register (inline).* uns8 pic_rf_receive (uns8 *data, uns8 bytes_to_receive)
- *Receive data from nRF24L01.* void pic_rf_receive_inline (uns8 *data, uns8 bytes_to_receive)
- *Receive data from nRF24L01.* uns8 pic_rf_send_byte (uns8 b)
- *Clock a byte into the nRF24L01.* uns8 pic_rf_send_byte_int (uns8 b)
- *Clock a byte into the nRF24L01.* uns8 pic_rf_send_command (uns8 cmd, uns8 *data, uns8 data_len)
- *Send command to the nrf24l01.* uns8 pic_rf_send_command_inline (uns8 cmd, uns8 *data, uns8 data_len)
- *Send command to the nrf24l01 (inline).* uns8 pic_rf_send_command_single (uns8 cmd, uns8 data)
- *Send a single data byte command to the nrf24l01.* void pic_rf_set_channel (uns8 channel)
- *Receive data from nRF24L01 (inline).* void pic_rf_set_mode (uns8 mode)
- *Set rf mode to transmit or receive.* void pic_rf_setup ()
- *Setup ports and pins for communication with nRF24L01.* void pic_rf_transmit (uns8 *data, uns8 bytes_to_transmit)

Transmit data from nRF24L01. Variables

- static uns8 rf_current_channel = 2
- static bit rf_current_mode_receive = 0

Detailed Description

RF routines for the Nordic nRF24L01 chip

Define Documentation

#define CONFIG_CRCO 2

#define CONFIG_EN_CRC 3

#define CONFIG_MASK_MAX_RT 4

#define CONFIG_MASK_RX_DR 6

#define CONFIG_MASK_TX_DS 5

#define CONFIG_PRIM_RX 0

#define CONFIG_PWR_UP 1

#define FIFO_STATUS_RX_EMPTY 0

#define FIFO_STATUS_RX_FULL 1

#define FIFO_STATUS_TX_EMPTY 4

#define FIFO_STATUS_TX_FULL 5

#define FIFO_STATUS_TX_REUSE 6

#define OP_ENABLE_1_MBPS 5

rf_config options - enable 1 MPS transmition (otherwise 250Kbps)

#define OP_ENABLE_CH2 7

rf_config options - enable channel 2 reception

#define OP_ENABLE_CRC 2

rf_config options - enable CRC (recommended!)

#define OP_ENABLE_RECEIVE 0

rf_config options - enable receive

#define OP_ENABLE_SHOCKBURST 6

rf_config options - enable shockburst transmition

#define OP_LONG_CRC 1

rf_config options - enable 16 bit CRC (otherwise 8 bit CRC if 0)

#define pic_rf_get_status() pic_rf_read_register(RF_NOP, 0, 0)

#define pic_rf_receive_mode() pic_rf_set_mode(RECEIVE_MODE)

#define pic_rf_set_status(status) pic_rf_send_command(RF_WR_REG_STATUS, status, 1)

#define pic_rf_transmit_mode() pic_rf_set_mode(TRANSMIT_MODE)

#define RECEIVE_MODE 1

Mode selection - receive

```
#define RF_FLUSH_RX  0b11100010
#define RF_FLUSH_TX  0b11100001
#define RF_NOP  0b11111111
#define RF_R_RX_PAYLOAD  0b01100001
#define RF_RD_REG_CD  0b00001001
#define RF_RD_REG_CONFIG_REG  0b00000000
#define RF_RD_REG_FIFO_STATUS  0b00010111
#define RF_RD_REG_RX_PW_P0  0b00010001
#define RF_RD_REG_STATUS  0b00000111
#define RF_W_TX_PAYLOAD  0b10100000
#define RF_WR_REG_CONFIG_REG  0b00100000
#define RF_WR_REG_EN_AA  0b00100001
#define RF_WR_REG_RF_CH  0b00100101
#define RF_WR_REG_RF_SETUP  0b00100110
#define RF_WR_REG_RX_ADDR_P0  0b00101010
#define RF_WR_REG_RX_PW_P0  0b00110001
#define RF_WR_REG_SETUP_AW  0b00100011
#define RF_WR_REG_SETUP_RETR  0b00100100
#define RF_WR_REG_STATUS  0b00100111
#define RF_WR_REG_TX_ADDR  0b00110000
#define STATUS_MAX_RT  4
#define STATUS_RX_DR  6
#define STATUS_TX_DS  5
#define STATUS_TX_FULL  0
#define TRANSMIT_MODE  0
```

Mode selection - transmit

Function Documentation

void pic_rf_init (rf_config * *my_config*)

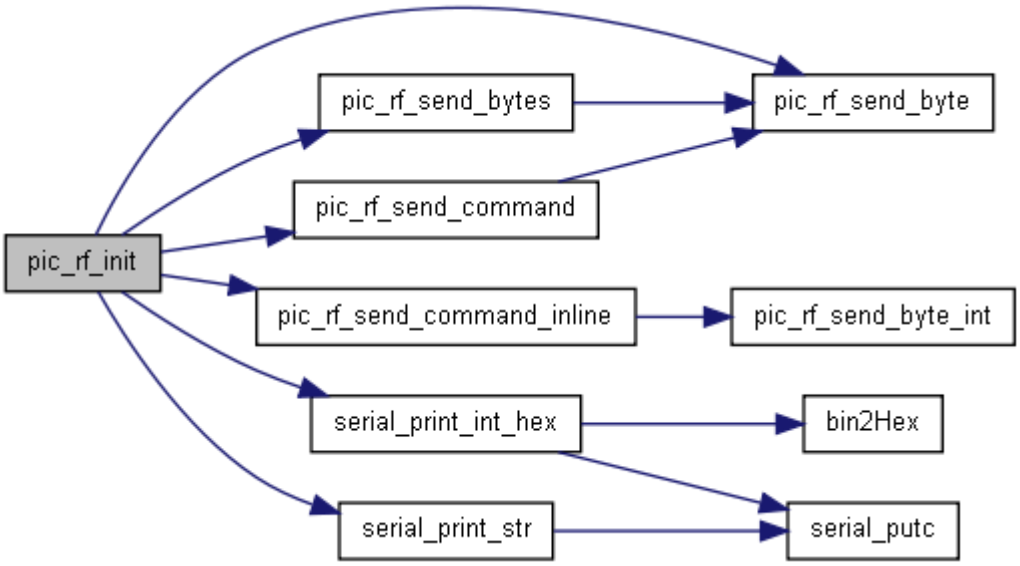
Sends the configuration to the Nordic nRF24L01 chip ready to begin communication. This routine assumes you have already set my_config to the correct values.

Initialise nRF24L01 chip with config.

Sends the configuration to the Nordic nrf2401a chip ready to begin communication. This routine assumes you have already set my_config to the correct values.

```
84 {
85 uns8 temp;
86 uns8 options;
87 
88     make_output(rf_data_port, rf_data_pin); // make data pin output
89     clear_pin(rf_clk1_port, rf_clk1_pin);       // make sure it's 0
90 
91     pic_rf_chip_enable(0);
92     pic_rf_chip_select(1);  // config mode
93 
94     pic_rf_send_byte(my_config->payload_width_ch2);
95     pic_rf_send_byte(my_config->payload_width_ch1);
96     pic_rf_send_bytes(my_config->address_ch2, 5);
97     pic_rf_send_bytes(my_config->address_ch1, 5);
98 
99     options = my_config->options;
100 
101     temp = my_config->address_width << 2;
102     temp.1 = options.OP_LONG_CRC; // |= my_config->long_crc << 1;
103     temp.0 = options.OP_ENABLE_CRC; // |= my_config->enable_crc;
104 
105     pic_rf_send_byte(temp);
106 
107     temp = options & 0b11100000;  //pull off top three bits
108     //temp.7 = options.ENABLE_CH2; // |= my_config->enable_ch2 << 7;
109     //temp.6 = options.ENABLE_SHOCKBURST; // |= my_config->enable_shockburst << 6;
110     //temp.5 = options.ENABLE_1_MBPS; //my_config->enable_1_mbps << 5;
111     temp |= (my_config->crystal & 0b00000111) << 2; // bits 4,3,2 - mask 3 bit range
112     temp |= (my_config->output_power & 0b00000011); // bits 1,0 - mask 2 bit range
113 
114     pic_rf_send_byte(temp);
115 
116     temp = my_config->channel << 1;
117     rf_current_channel = my_config->channel;
118 
119     temp |= options.OP_ENABLE_RECEIVE; // my_config->enable_receive;
120     rf_current_mode_receive = options.OP_ENABLE_RECEIVE;
121 
122     pic_rf_send_byte(temp);
123 
124     pic_rf_chip_select(0);  // config mode ended
125     pic_rf_chip_enable(1);  // on the air!
126 
127   }
```

Here is the call graph for this function:



void pic_rf_quick_init (char * *my_config*, uns8 *my_channel*, bit *my_receive_on*)

While the usual pic_rf_init() routine is excellent when you want to programatically change the 2401a config, if you're only doing this once (at the start) then it's likely you're burning a lot of instructions (154 words on a PIC16 device) just to send some bytes of config out to the 2401a. If you know your config in advance, then you can just send the byte-stream config using this routine. Use the nrf2401a_config.pl script in the tools directory to generate this string.

```
62                                                                              {
63
64 uns8 byte_counter;
65     make_output(rf_data_port, rf_data_pin); // make data pin output
66     clear_pin(rf_clk1_port, rf_clk1_pin);       // make sure it's 0
67
68     pic_rf_chip_enable(0);
69     pic_rf_chip_select(1);  // config mode
70
71     for(byte_counter = 0 ; byte_counter < 15 ; byte_counter++) {
72         pic_rf_send_byte(my_config[byte_counter]);
73     }
74     rf_current_channel = my_channel;
75     rf_current_mode_receive = my_receive_on;
76
77     pic_rf_chip_select(0);  // config mode ended
78     pic_rf_chip_enable(1);  // on the air!
79
80 }
```

Here is the call graph for this function:

uns8 pic_rf_read_register (uns8 *cmd*, uns8 * *data*, uns8 *data_len*)

Internal routine to read a particular nRF24L01 register. Clocks out data_len bytes from the chip. Internal routine.

Parameters:

cmd Read register command, eg RF_RD_REG_STATUS

data Pointer to array of bytes where data will be put
data_len Number of bytes to clock out

```
92                                                                  { // returns status
93
94 uns8 byte_counter, status;
95
96     clear_pin(rf_csn_port, rf_csn_pin);
97     status = pic_rf_send_byte(cmd);
98     for(byte_counter = 0 ; byte_counter < data_len ; byte_counter++) {
99         data[byte_counter] = pic_rf_send_byte(0);
100     }
101
102     set_pin(rf_csn_port, rf_csn_pin);
103     return status;
104 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

uns8 pic_rf_read_register_inline (uns8 *cmd*, uns8 * *data*, uns8 *data_len*) `[inline]`

Internal routine to read a particular nRF24L01 register. Clocks out data_len bytes from the chip. Internal routine. Inline version.

Parameters:

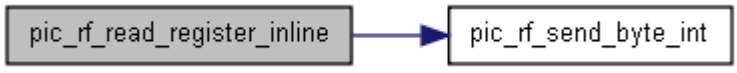
cmd Read register command, eg RF_RD_REG_STATUS
data Pointer to array of bytes where data will be put
data_len Number of bytes to clock out

Returns:

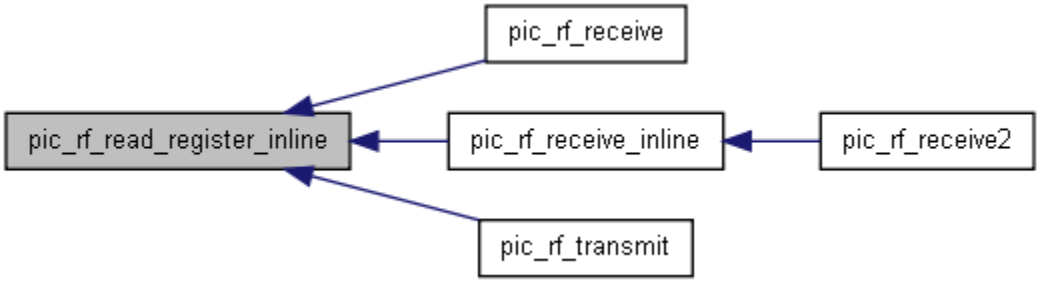
nRF24L01 status

```
326                                                                                 {
327
328 uns8 byte_counter, status;
329
330     clear_pin(rf_csn_port, rf_csn_pin);
331
332     status = pic_rf_send_byte_int(cmd);
333
334     for(byte_counter = 0 ; byte_counter < data_len ; byte_counter++) {
335         data[byte_counter] = pic_rf_send_byte_int(0); // dummy send to get byte back
336     }
337
338     set_pin(rf_csn_port, rf_csn_pin);
339
340     return status;
341 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



uns8 pic_rf_receive (uns8 * *data*, uns8 *bytes_to_receive*)

Having been notified that there is data available, call this routine to clock the data in from the nRF24L01.

Receive data from nRF24L01.

Having been notified that there is data available, call this routine to clock the data in from the nrf2401a.

!pic_rf_chip_enable(0); // save power

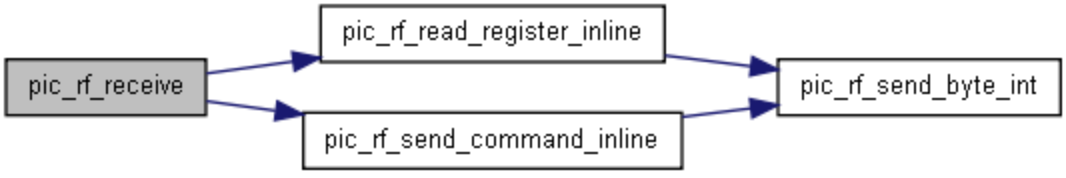
pic_rf_chip_enable(1); // turn chip back on

!pic_rf_chip_enable(0); // save power

pic_rf_chip_enable(1); // turn chip back on

```
130                                                     {
131 uns8 byte_count, bit_count, temp;
132
134     bit  my_store_gie = intcon.GIE;
135     kill interrupts();
136
137     make input(rf data port, rf data pin); // make data pin input
138
139     for (byte_count = 0; byte_count < bytes_to_receive; byte_count++) {
140
141         for (bit_count = 0; bit_count < 8; bit_count++) {
142             temp <<= 1;
143             temp.0 = test pin(rf_data_port, rf_data_pin);
144             set pin(rf_clk1_port, rf_clk1_pin);         // clock it out
145             clear pin(rf_clk1_port, rf_clk1_pin);       // ready for next bit
146         }
147         data[byte_count] =  temp;
148     }
149
151     intcon.GIE = my store gie;
152 }
```

Here is the call graph for this function:

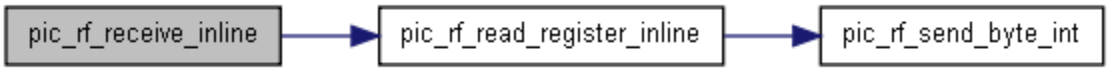


void pic_rf_receive_inline (uns8 * *data*, uns8 *bytes_to_receive*) `[inline]`

Having been notified that there is data available, call this routine to clock the data in from the nRF24L01.

```
348                                                                     {
349         pic_rf_read_register_inline(RF_R_RX_PAYLOAD, data, bytes_to_receive);
350 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

uns8 pic_rf_send_byte (uns8 *b*)

Clock one byte into the nRF24L01. Internal routine.

Parameters:

b The byte to send

Returns:

nRF24L01 status

Clock a byte into the nRF24L01.

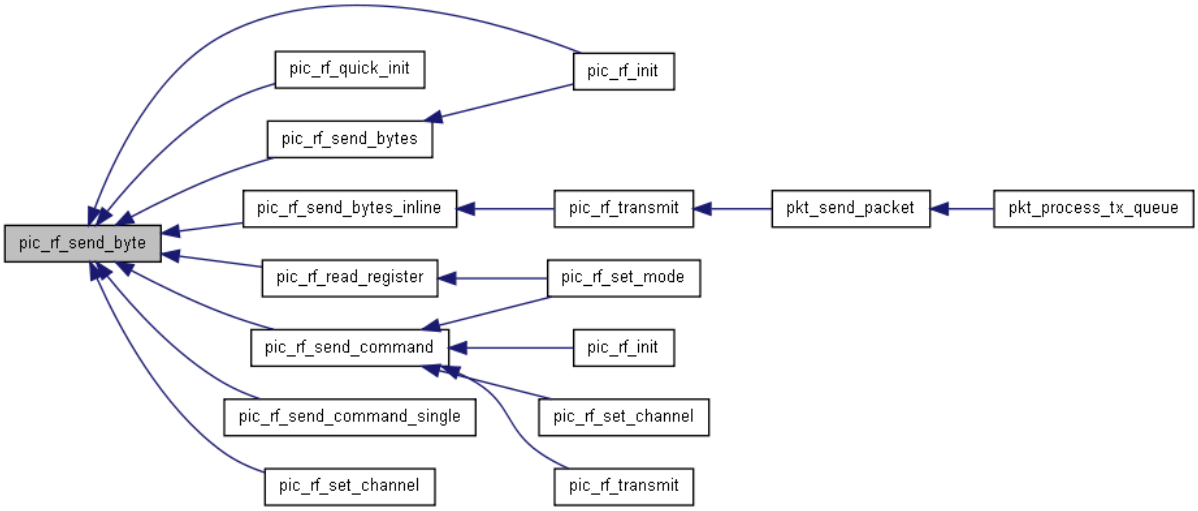
Internal routine to send a byte to the nrf2401a. Generally you shouldn't need to use this, see pic_rf_transmit instead

See also:

pic_rf_transmit

```
42 {
43 uns8 bit_counter;
44     for(bit_counter = 0 ; bit_counter < 8 ; bit_counter++) {
45         change_pin(rf_data_port, rf_data_pin, b.7); // Put data on data pin
46         set_pin(rf_clk1_port, rf_clk1_pin);         // clock it in (positive edge)
47         clear_pin(rf_clk1_port, rf_clk1_pin);       // ready for next bit
48 
49         b <<= 1;    // Move all the bits left
50     }   // repeat until finished
51 
52 }   // pic_rf_send_byte
```

Here is the caller graph for this function:



uns8 pic_rf_send_byte_int (uns8 *b*)

Clock one byte into the nRF24L01. Internal routine.

Parameters:

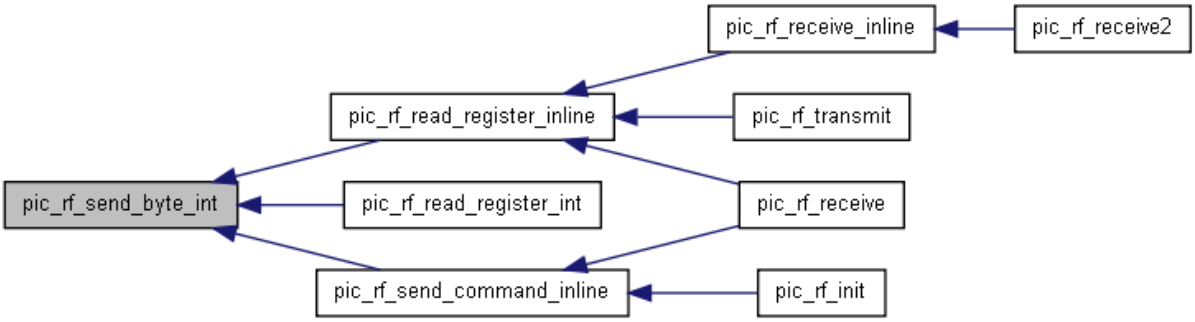
b The byte to send

Returns:

nRF24L01 status

```
141 {
142 uns8 bit_counter, status;
143     // - For debug: print_int_hex(b); putc(' ');
144     for(bit_counter = 0 ; bit_counter < 8 ; bit_counter++) {
145         change pin(rf mosi port, rf mosi pin, b.7); // Put data on data pin
146         set pin(rf_sck_port, rf_sck_pin);           // clock it in (positive edge)
147         status <<= 1;
148         status.0 = test pin(rf_miso_port, rf_miso_pin);
149         clear pin(rf_sck_port, rf_sck_pin);     // ready for next bit
150 
151         b <<= 1;    // Move all the bits left
152     }   // repeat until finished
153     return status;
154 }   // pic_rf_send_byte
```

Here is the caller graph for this function:



uns8 pic_rf_send_command (uns8 *cmd*, uns8 * *data*, uns8 *data_len*)

Send a command and associated data to the nRF24L01

Parameters:

cmd Command to send, eg, RF_WR_REG_SETUP_RETR
data Pointer to an array of bytes to send as data for the command
data_len Number of bytes in the array

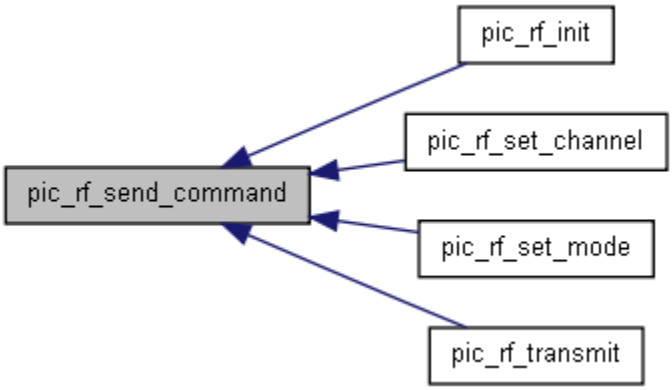
Returns:

nRF24L01 status

```
66                                                          {
67 
68 uns8 byte counter, status;
69 
70     clear pin(rf_csn_port, rf_csn_pin);
71     status = pic rf send byte(cmd);
72     for(byte_counter = 0 ; byte_counter < data_len ; byte_counter++) {
73         pic rf send byte(data[byte_counter]);
74     }
75 
76     set pin(rf csn port, rf csn pin);
77     return status;
78 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



uns8 pic_rf_send_command_inline (uns8 *cmd*, uns8 * *data*, uns8 *data_len*) `[inline]`

Send a command and associated data to the nRF24L01. Internal routine. Inline version.

Parameters:

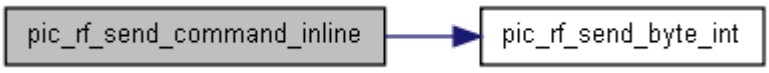
cmd Command to send, eg, RF_WR_REG_SETUP_RETR
data Pointer to an array of bytes to send as data for the command
data_len Number of bytes in the array

Returns:

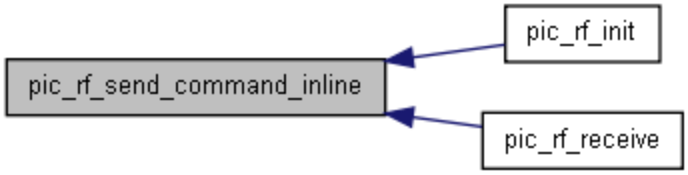
nRF24L01 status

```
354                                                                     {
355
356 uns8 byte_counter, status;
357
358     clear_pin(rf_csn_port, rf_csn_pin);
359     status = pic_rf_send_byte_int(cmd);
360     for(byte_counter = 0 ; byte_counter < data_len ; byte_counter++) {
361         pic_rf_send_byte_int(data[byte_counter]);
362     }
363
364     set_pin(rf_csn_port, rf_csn_pin);
365     return status;
366 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



uns8 pic_rf_send_command_single (uns8 *cmd*, uns8 *data*)

Send a command and 1 byte of data to the nRF24L01

Parameters:

cmd Command to send, eg, RF_WR_REG_SETUP_RETR
data One byte of data for the command

Returns:

nRF24L01 status

```
80                                                      {
81
82 uns8 byte_counter, status;
83
84     clear_pin(rf_csn_port, rf_csn_pin);
85     status = pic_rf_send_byte(cmd);
86     pic_rf_send_byte(data);
87     set_pin(rf_csn_port, rf_csn_pin);
88     return status;
89 }
```

Here is the call graph for this function:

void pic_rf_set_channel (uns8 *channel*)

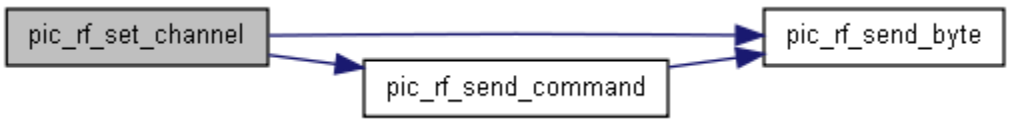
Having been notified that there is data available, call this routine to clock the data in from the nRF24L01. Change channel on the nRF24L01 Changes the current channel used by the nRF24L01.

Receive data from nRF24L01 (inline).

Reclocks the essential config to change the current channel used by the nrf2401a.

```
200 {
201     bit  my_store_gie = intcon.GIE;
202     kill_interrupts();
203
204     clear_bit(tris_array[rf_data_port - PORTA], rf_data_pin); // make data pin output
205
206     pic_rf_chip_enable(0);
207     pic_rf_chip_select(1);  // config mode
208
209     rf_current_channel = channel;
210     channel <<= 1;
211     channel |= rf_current_mode_receive;
212
213     pic_rf_send_byte(channel);
214
215     pic_rf_chip_select(0);  // config mode ended
216     pic_rf_chip_enable(1);
217
218     intcon.GIE = my_store_gie;
219 }
```

Here is the call graph for this function:

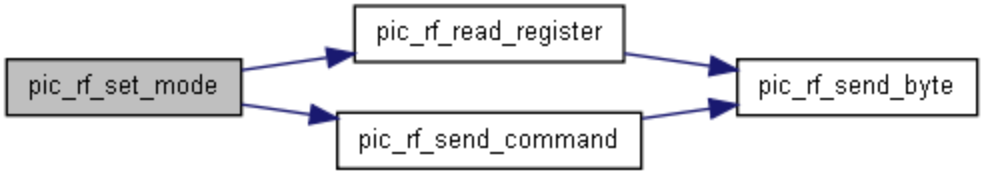


void pic_rf_set_mode (uns8 *mode*)

Pass RECEIVE_MODE or TRANSMIT_MODE to change current mode. Generally, you shouldn't need to call this routine. The library assumes you want to receive until you transmit, in which case it switches automatically to transmit mode and back to receive afterwards.

```
179 {
180     bit  my_store_gie = intcon.GIE;
181     kill_interrupts();
182 
183     make_output(rf_data_port, rf_data_pin); // make data pin output
184     pic_rf_chip_enable(0);
185     pic_rf_chip_select(1);  // config mode
186 
187     change_pin(rf_data_port, rf_data_pin, mode);    // send a zero
188     set_pin(rf_clk1_port, rf_clk1_pin);         // clock it in (positive edge)
189     clear_pin(rf_clk1_port, rf_clk1_pin);       // ready for next bit
190 
191     pic_rf_chip_select(0);  // config mode ended
192     pic_rf_chip_enable(1);
193 
194     rf_current_mode_receive = mode;
195 
196     intcon.GIE = my_store_gie;
197 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void pic_rf_setup ()

Set up ports and pins to correct input/output for communication with Nordic nRF24L01

Setup ports and pins for communication with nRF24L01.

Set up ports and pins to correct input/output for communication with Nordif nrf2401a

```
221                     {
222 
223     make_output(rf_data_port, rf_data_pin); // make data pin output
224     make_output(rf_cs_port, rf_cs_pin); // make cs pin output
225     make_output(rf_ce_port, rf_ce_pin); // make ce pin output
226     make_input (rf_dr1_port, rf_dr1_pin); // make dr1 pin input
227     make_output(rf_clk1_port, rf_clk1_pin); // make clk1 pin output
228 
229 }
```

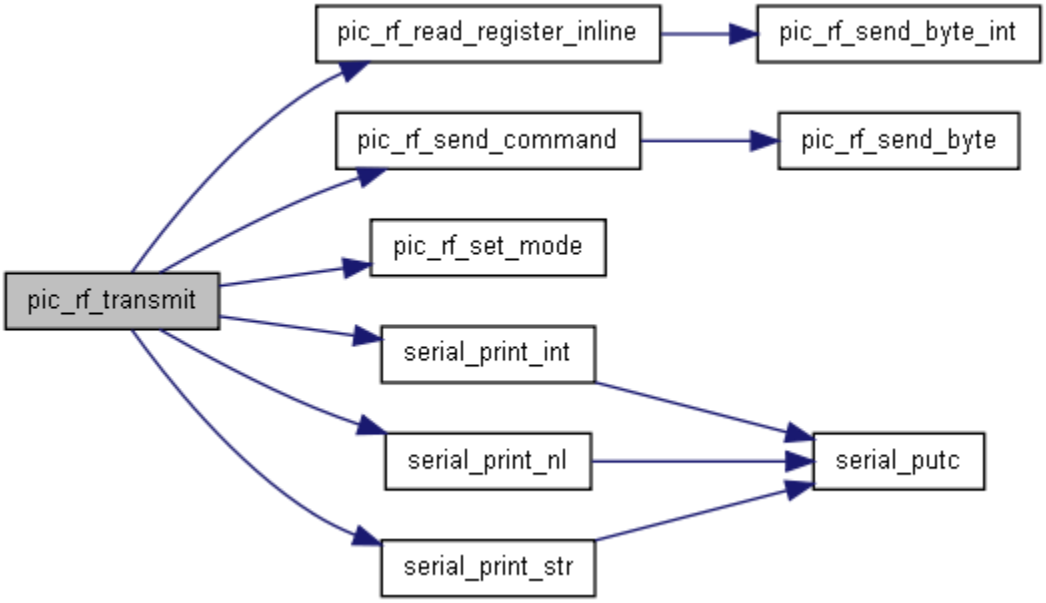
void pic_rf_transmit (uns8 * *data*, uns8 *bytes_to_transmit*)

Changes to transmit mode, clocks data into the nrf24L01 and hits the shockburst button. Returns to receive mode when finished.

```
326                                                     {
327 
328 uns8 byte_count, bit_count, temp, cd;
```

```
329     start_crit_sec();
330
331     pic_rf_set_mode(TRANSMIT_MODE);
332
333     //pic_rf_send_command (RF_FLUSH_TX, 0, 0    );
334     pic_rf_read_register_inline(RF_RD_REG_CD, &cd, 1);
335     serial_print_str("\n cd=");
336     serial_print_int(cd);
337     serial_print_nl();
338     pic_rf_send_command(RF_W_TX_PAYLOAD, data, bytes_to_transmit);
339
340     set_pin(rf_ce_port, rf_ce_pin);
341     delay_us(10);   // 10us pulse - send packet out on airways
342     clear_pin(rf_ce_port, rf_ce_pin);
343     delay_us(130);  // TX settling
344     pic_rf_set_mode(RECEIVE_MODE);  // go back to receive mode
345
346     end_crit_sec();
347 }
```

Here is the call graph for this function:



Variable Documentation

uns8 rf_current_channel = 2 [static]

Maintain state of current channel

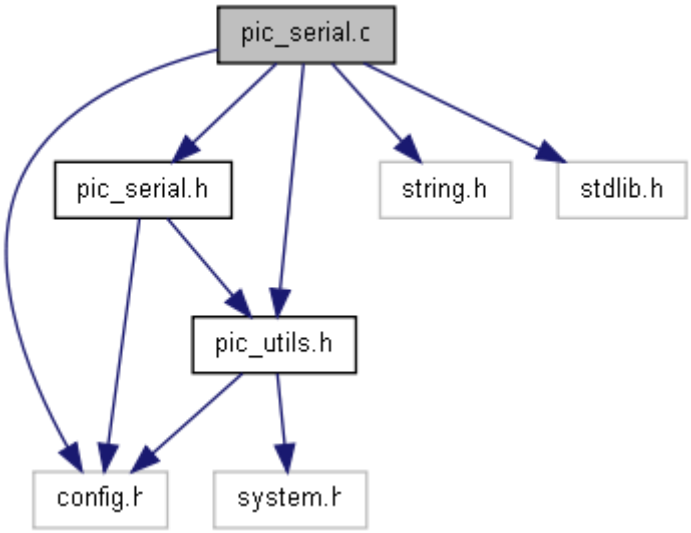
bit rf_current_mode_receive = 0 [static]

Maintain state of current mode (1 = receive mode)

---

pic_serial.c File Reference

Include dependency graph for pic_serial.c:



Functions

- uns8 bin2Hex (uns8 x)
- uns8 even_7bit_parity (uns8 in)
- *Change 7th bit (MSB) to give even parity.* uns8 odd_7bit_parity (uns8 in)
- *Change 7th bit (MSB) to give od parity.* uns8 serial_getc (void)
- *Retrieve a character from the serial port.* void serial_print_int (uns16 i)
- *Print a 16 bit number to the serial port.* void serial_print_int_hex (uns8 i)
- *Print an 8 bit number in hex to the serial port.* void serial_print_int_hex_16bit (uns16 i)
- *Print a 16 bit number in hex to the serial port.* void serial_print_nl ()
- *Print a newline.* void serial_print_spc ()
- *Print a space.* void serial_print_str (rom char *str)
- void serial_print_str (char *str)
- *Print a string out to the serial port.* void serial_print_var (char *str, uns16 i)
- void serial_putc (uns8 c)
- *Transmit a single character.* uns8 serial_rx_avail ()
- *Tests if the serial rx fifo has a character available.* void serial_rx_isr ()
- *Serial receive interrupt service routine.* void serial_setup (uns8 req_spbrg)
- *Configure the pic for serial communicaitons.* uns8 serial_tx_empty ()
- *Tests if the serial tx fifo is empty.* void serial_tx_isr ()

Serial transmit interrupt service routine. Variables

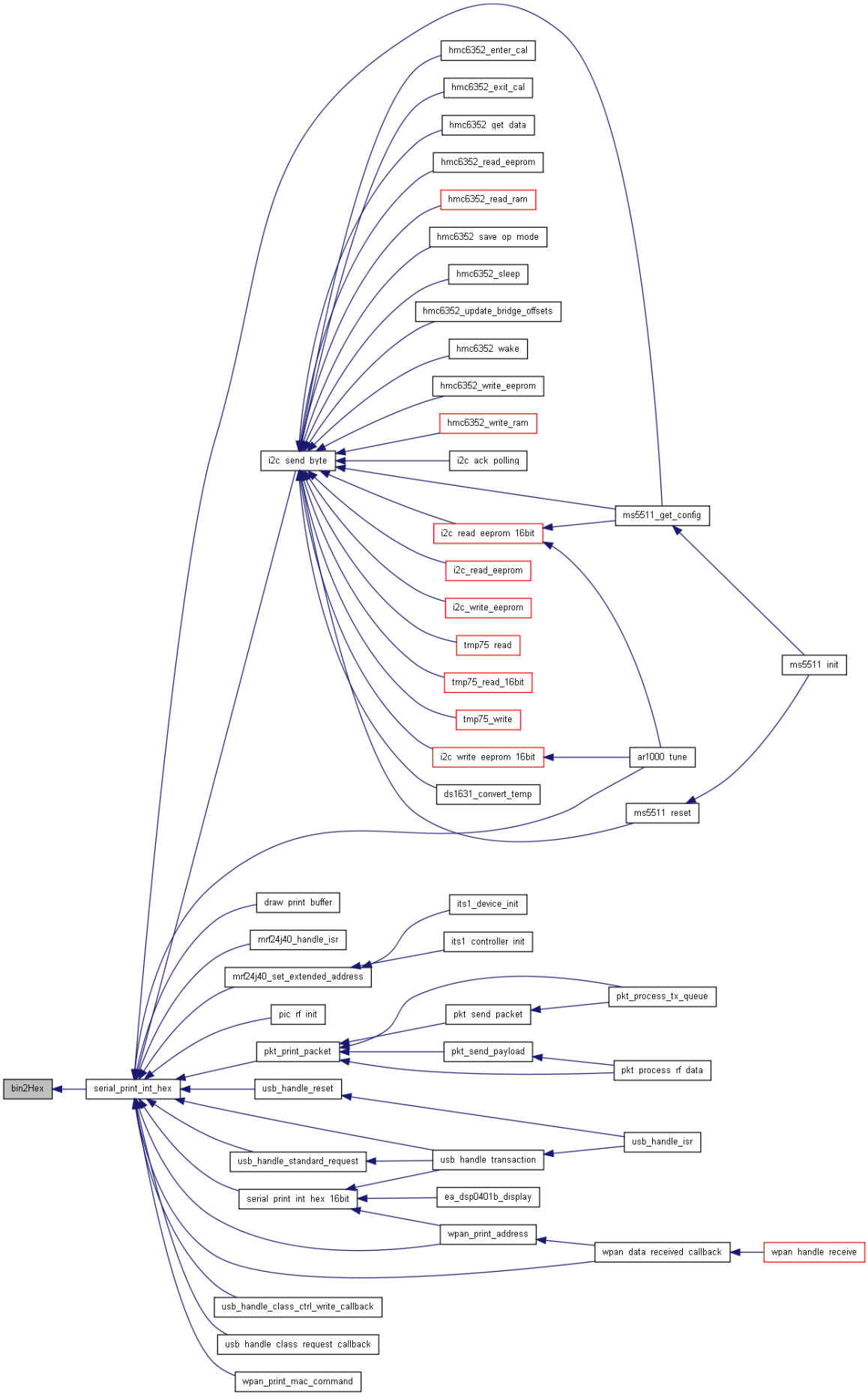
- uns8 rx_buffer [SERIAL_RX_BUFFER_SIZE]
- uns8 rx_end = 0
- uns8 rx_start = 0
- uns8 tx_buffer [SERIAL_TX_BUFFER_SIZE]
- uns8 tx_end = 0
- uns8 tx_start = 0

Function Documentation

uns8 bin2Hex (uns8 *x*) `[inline]`

```
130 {
131    if (x < 10) {
132      return '0' + x;
133    } else {
134      return 'A' -10 +  x;
135    }
136 }
```

Here is the caller graph for this function:


hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_read_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_update_bridge_offsets
hmc6352_wake
hmc6352_write_eeprom
hmc6352_write_ram
i2c_send_byte
i2c_ack_polling
ms5511_get_config
i2c_read_eeprom_16bit
i2c_read_eeprom
i2c_write_eeprom
tmp75_read
tmp75_read_16bit
tmp75_write
i2c_write_eeprom_16bit
ar1000_tune
ds1631_convert_temp
ms5511_init
ms5511_reset
draw_print_buffer
mrf24j40_handle_isr
mrf24j40_set_extended_address
pic_rf_init
pkt_print_packet
its1_device_init
its1_controller_init
pkt_send_packet
pkt_send_payload
pkt_process_tx_queue
pkt_process_rf_data
bin2Hex
serial_print_int_hex
usb_handle_reset
usb_handle_isr
usb_handle_standard_request
usb_handle_transaction
serial_print_int_hex_16bit
ea_dsp0401b_display
wpan_print_address
wpan_data_received_callback
wpan_handle_receive
usb_handle_class_ctrl_write_callback
usb_handle_class_request_callback
wpan_print_mac_command

uns8 even_7bit_parity (uns8 *in*)

```
63                                  {
64
65 uns8 t;
66
67  //return ((((((in)^(((in)<<4)|((in)>>4)))+0x41)|0x7C)+2)&0x80);
68
69
70 asm {
71
72     bcf     in,7  ;assume the parity is even
73       ;Note: for odd parity, use bsf
74  ; assume the bits in byte_to_send are abcdefgh
75
76     swapf   in,W  ;W = efghabcd
77     xorwf  _in,W  ;W = ea.fb.gc.hd.ea.fb.gc.hd
78                             ; where ea means e^a, etc
79     movwf  _t               ;
80     rlf    _t,F            ;t = fb.gc.hd.ea.fb.gc.hd.??
81     rlf    _t,F             ;t = gc.hd.ea.fb.gc.hd.??.ea
82     xorwf  _t,F             ;t = gcea.hdfb.gcea.hdfb.gcea.?.?
83                            ;again, gcea means g^c^e^a
84     rlf     t,W             ;w = hdfb.gcea.hdfb.gcea.hdfb.?.fb
85     xorwf  _t,W             ;w = abcdefgh.abcdefgh.....
86                            ;ie, the upper 5-bits of w each contain
87                            ;the parity calculation.
88     andlw  0x80            ;We only need one of them
89     bcf    in,7  ;set bit 7 to 0 in preparation for or-ing with our result
90     iorwf _in,F  ;copy it to the MSB of the byte to send.
91
92 }
93     return in;
94
95
96 }
```

uns8 odd_7bit_parity (uns8 *in*)

```
100                                 {
101
102 uns8 t;
103
104 asm {
105     bsf    _in,7  ;assume the parity is odd
106
107  ; assume the bits in "in" are abcdefgh
108
109     swapf  _in,W  ;W = efghabcd
110     xorwf  _in,W  ;W = ea.fb.gc.hd.ea.fb.gc.hd
111                             ; where ea means e^a, etc
112     movwf  _t               ;
113     rlf    _t,F            ;t = fb.gc.hd.ea.fb.gc.hd.??
114     rlf    _t,F             ;t = gc.hd.ea.fb.gc.hd.??.ea
115     xorwf  _t,F             ;t = gcea.hdfb.gcea.hdfb.gcea.?.?
116                            ;again, gcea means g^c^e^a
117     rlf    _t,W             ;w = hdfb.gcea.hdfb.gcea.hdfb.?.fb
118     xorwf  _t,W             ;w = abcdefgh.abcdefgh.....
119                            ;ie, the upper 5-bits of w each contain
120                            ;the parity calculation.
121     andlw  0x80            ;We only need one of them
122     bcf   _in,7  ;set bit 7 to 0 in preparation for or-ing with our result
123     iorwf _in,F  ;copy it to the MSB of the byte to send.
124 }
```

```
125     return in;
126 }
```

uns8 serial_getc (void)

Retrieve character from the serial port. Note that if there is nothing in the fifo, this function will wait until a character is received - and this will never happen if interrupts are turned off when this is called! So, be careful not to call getc during a critical section or during an ISR unless* you're sure there's something in the fifo. You can do this by calling the serial_rx_avail() routine. In any other situation, you can call getc() and happily wait for a character to arrive.

```
341 {
342     uns8 rx_char, rx_next;
343 
344     while(rx_end == rx_start);  // wait until there is something received
345 
346     start_crit_sec();   // make sure nobody else can muck with the buffer
347 
348     rx_char = rx_buffer[rx_start];  // get character from the front of the buffer
349     rx_start++; // increment fifo start
350     if (rx_start == SERIAL_RX_BUFFER_SIZE) {    // if we're at the end
351         rx_start = 0;   // then wrap to the beginning
352     }
353 
354     end_crit_sec(); // now they can muck with the buffer
355 
356     return (rx_char);   // return the result we first thought of
357 
358 }   // -- getc
```

Here is the caller graph for this function:

void serial_print_int (uns16 *i*)

Print a 16 bit unsigned number in decimal to the serial port

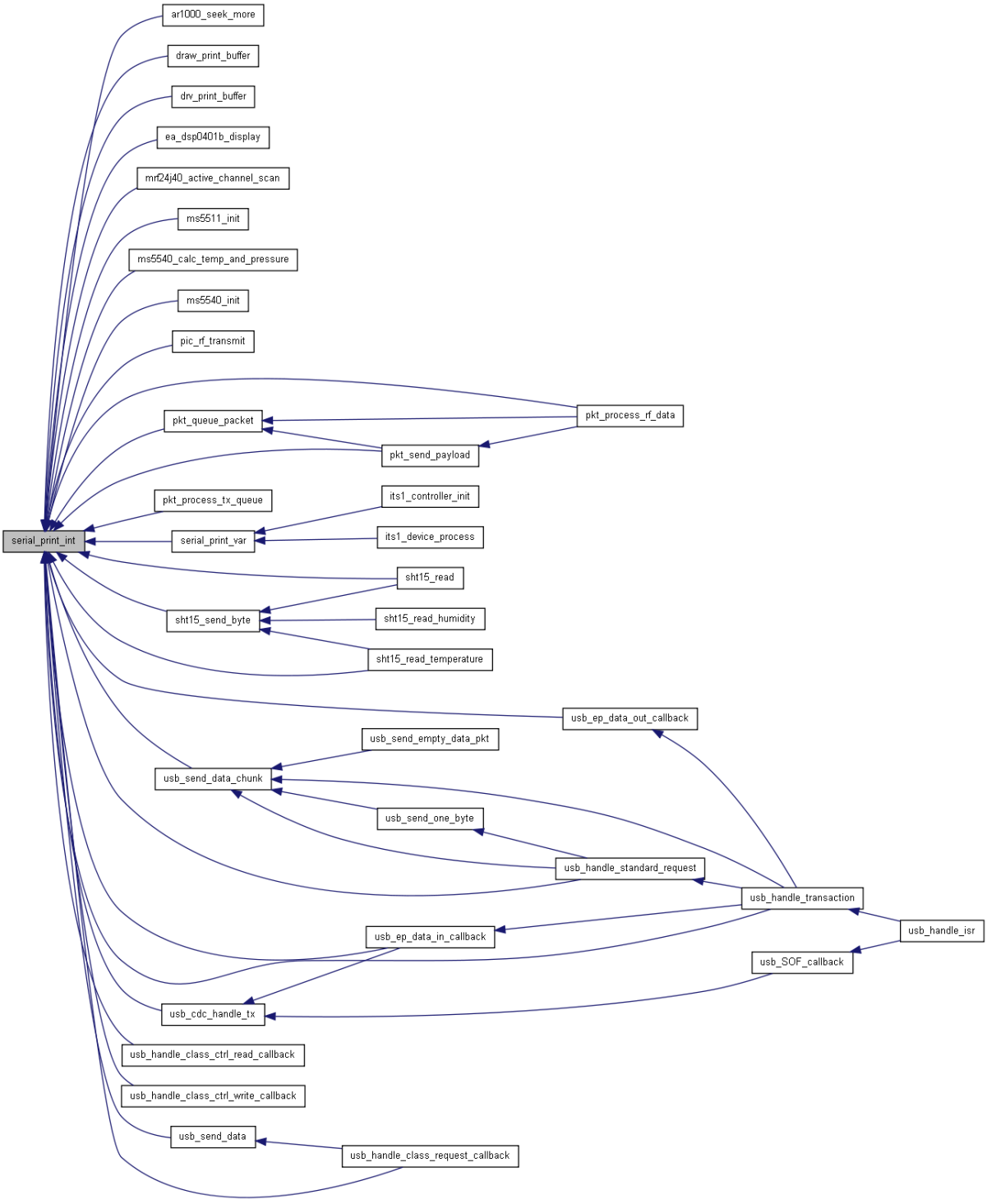
Parameters:

i the 16 bit number to be printed

```
386                                 {
387 
388 char buffer[6]; // up to 5 characters plus \0
389 uns8 count = 5;
390     buffer[5] = '\0';
391     do {
392         count--;
393         buffer[count] = '0' + i % 10;
394         i = i / 10;
395     } while (i > 0);
396     while (buffer[count]) {
397         serial_putc(buffer[count]);
398         count++;
399     }
400     //serial_print_str(&buffer[count]); //  print it out
401 //  for(count = 0 ; str[count] != 0; count++)
402  //   {
403  //   }
404 
405 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void serial_print_int_hex (uns8 *i*)

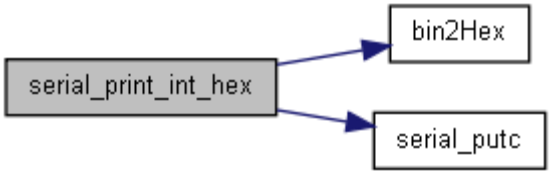
Print a 8 bit unsigned number in hex to the serial port

Parameters:

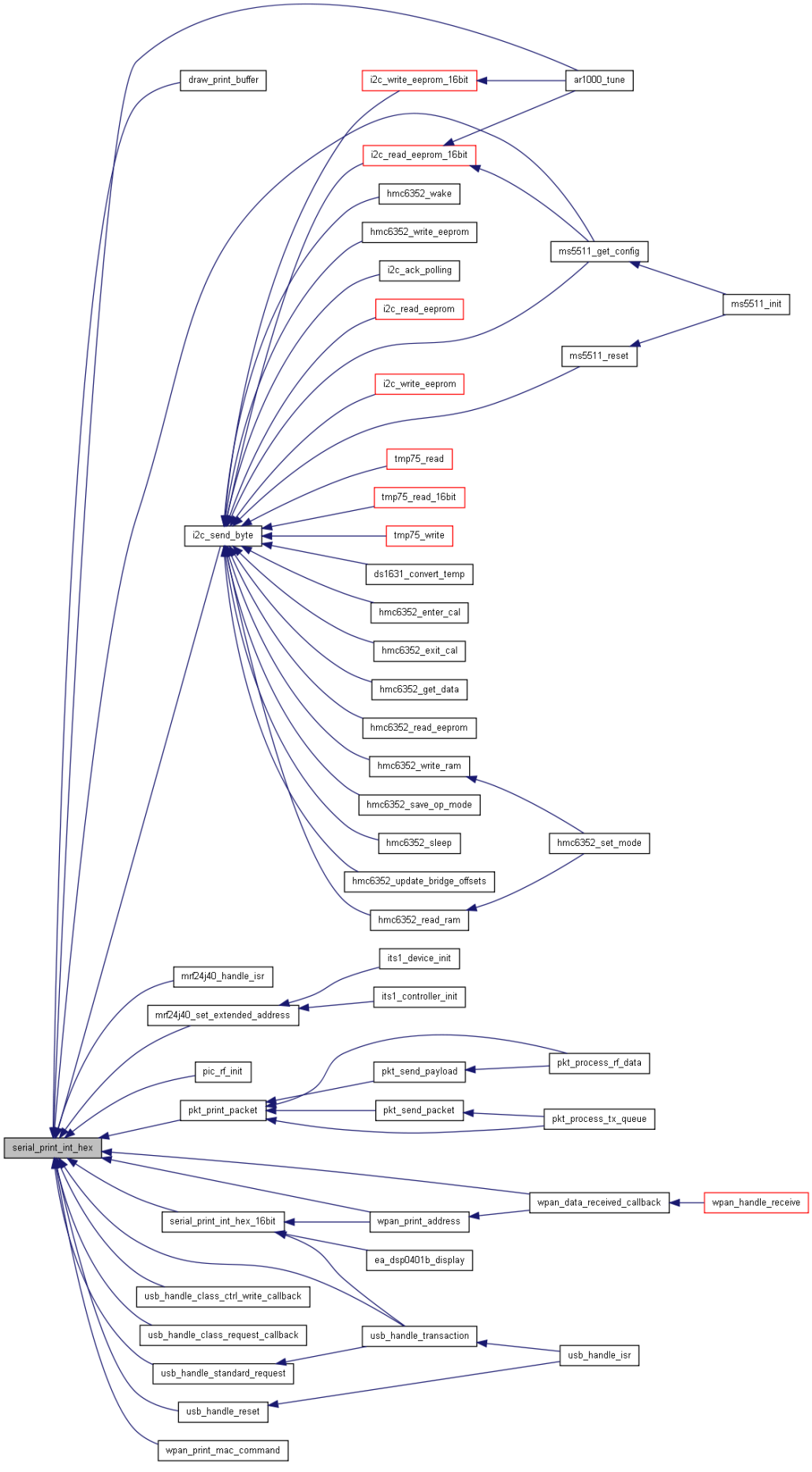
i 8 bit number to be printed

```
407                                     {
408
409     serial_putc(bin2Hex(i >> 4));
410     serial_putc(bin2Hex((i & 0x0f)));
411
412 }
```

Here is the call graph for this function:



Here is the caller graph for this function:


draw_print_buffer
i2c_write_eeprom_16bit
ar1000_tune
i2c_read_eeprom_16bit
hmc6352_wake
hmc6352_write_eeprom
ms5511_get_config
i2c_ack_polling
ms5511_init
i2c_read_eeprom
ms5511_reset
i2c_write_eeprom
tmp75_read
tmp75_read_16bit
i2c_send_byte
tmp75_write
ds1631_convert_temp
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_write_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_set_mode
hmc6352_update_bridge_offsets
hmc6352_read_ram
its1_device_init
mrf24j40_handle_isr
its1_controller_init
mrf24j40_set_extended_address
pic_rf_init
pkt_send_payload
pkt_process_rf_data
pkt_print_packet
pkt_send_packet
pkt_process_tx_queue
serial_print_int_hex
wpan_data_received_callback
wpan_handle_receive
serial_print_int_hex_16bit
wpan_print_address
ea_dsp0401b_display
usb_handle_class_ctrl_write_callback
usb_handle_class_request_callback
usb_handle_transaction
usb_handle_isr
usb_handle_standard_request
usb_handle_reset
wpan_print_mac_command

void serial_print_int_hex_16bit (uns16 *i*)

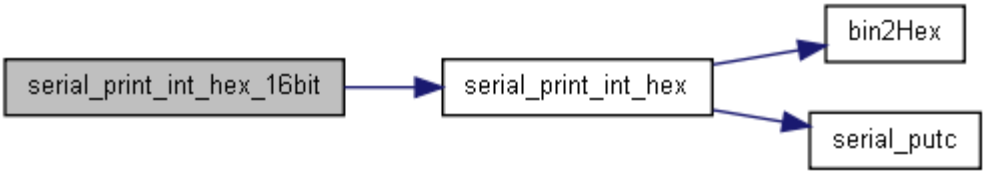
Print a 16 bit unsigned number in hex to the serial port

Parameters:

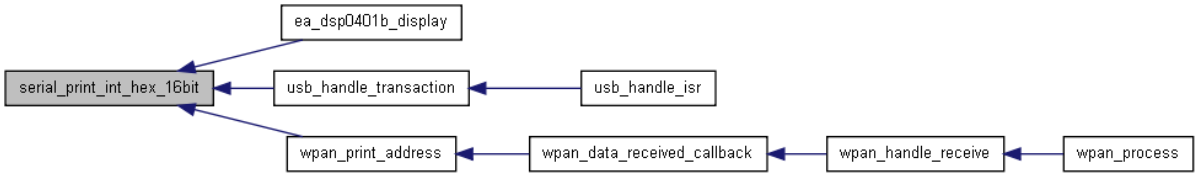
i 16 bit number to be printed

```
414                                             {
415     serial_print_int_hex(i >> 8);
416     serial_print_int_hex(i & 0xff);
417 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



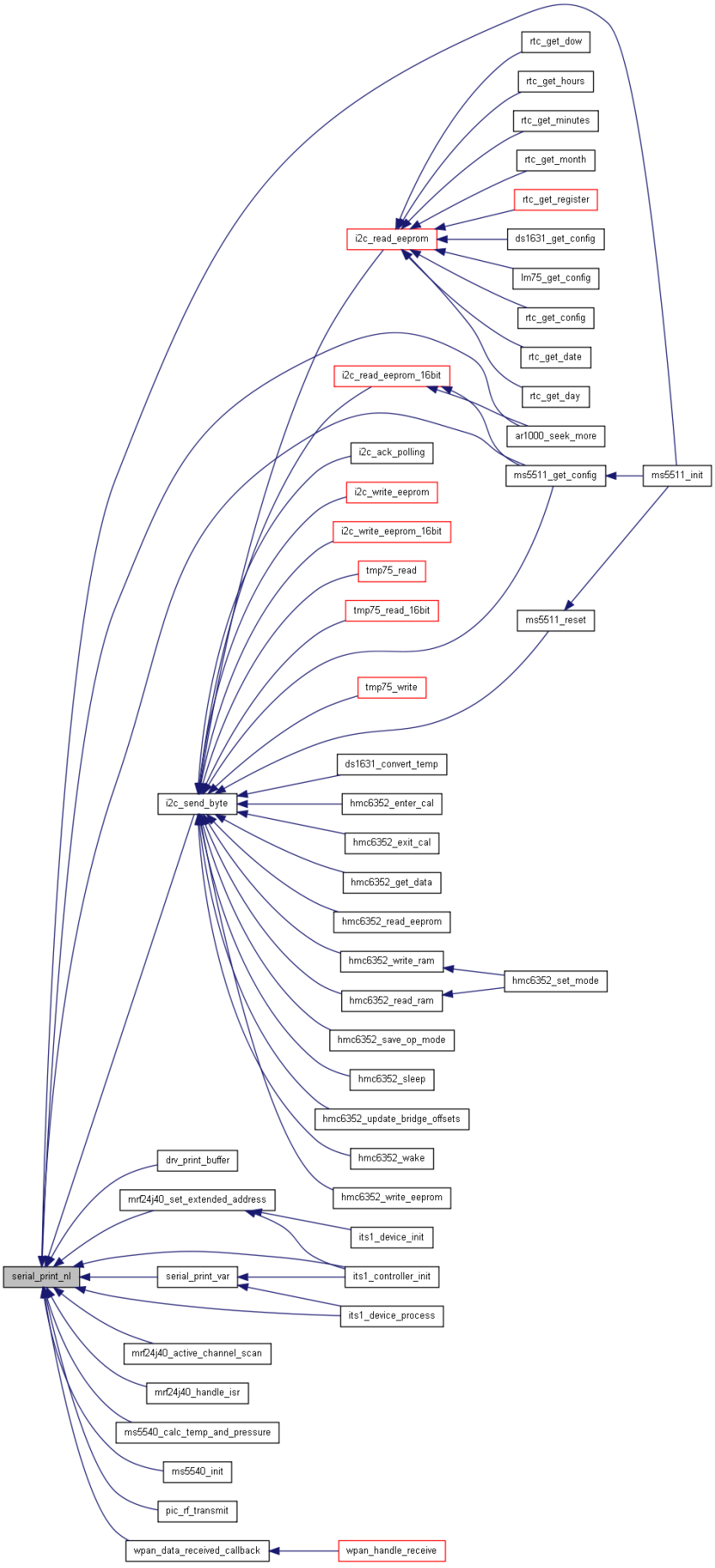
void serial_print_nl ()

Print a new line out the serial port - if you do this often, this routine can be used to save a couple of instructions. Always helps!

```
425                             {
426         serial_putc('\n');
427 }
```

Here is the call graph for this function:

Here is the caller graph for this function:


rtc_get_dow
rtc_get_hours
rtc_get_minutes
rtc_get_month
rtc_get_register
i2c_read_eeprom
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
i2c_read_eeprom_16bit
ar1000_seek_more
i2c_ack_polling
ms5511_get_config
ms5511_init
i2c_write_eeprom
i2c_write_eeprom_16bit
tmp75_read
tmp75_read_16bit
ms5511_reset
tmp75_write
ds1631_convert_temp
i2c_send_byte
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_write_ram
hmc6352_set_mode
hmc6352_read_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_update_bridge_offsets
hmc6352_wake
hmc6352_write_eeprom
drv_print_buffer
mrf24j40_set_extended_address
its1_device_init
serial_print_nl
serial_print_var
its1_controller_init
its1_device_process
mrf24j40_active_channel_scan
mrf24j40_handle_isr
ms5540_calc_temp_and_pressure
ms5540_init
pic_rf_transmit
wpan_data_received_callback
wpan_handle_receive

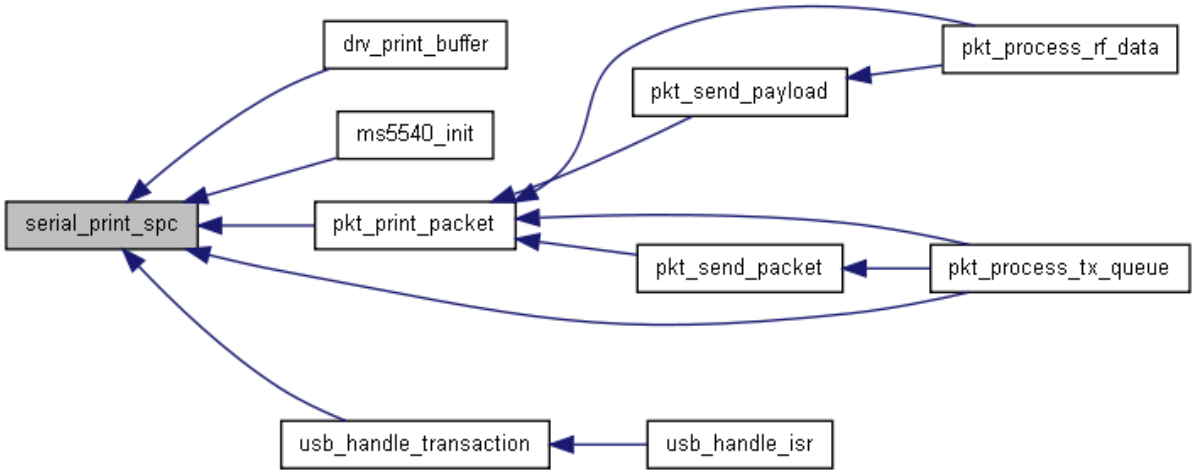
void serial_print_spc ()

Print a space out the serial port - if you do this often, this routine can be used to save a couple of instructions. Always helps!

```
420                             {
421         serial_putc(' ');
422 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void serial_print_str (rom char * *str*)

```
374                                         {
375
376 uns8 count;
377
378     for(count = 0 ; str[count] != 0; count++)
379     {
380         serial_putc(str[count]);
381     }
382 }
```

Here is the call graph for this function:

void serial_print_str (char * *str*)

Send a null terminated string out the serial port

Parameters:

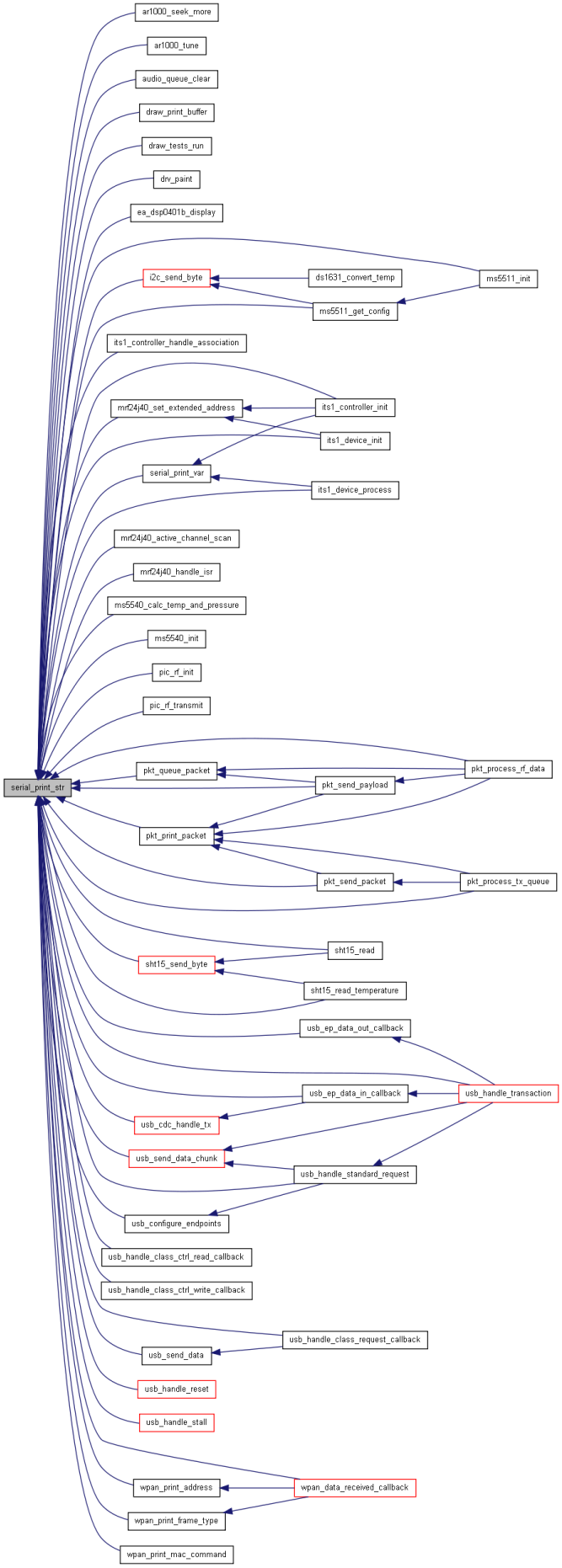
str the string to be sent

```
362                                     {
```

```
363
364 uns8 count;
365
366     for(count = 0 ; str[count] != 0; count++)
367     {
368         serial_putc(str[count]);
369     }
370 }
```

Here is the call graph for this function:

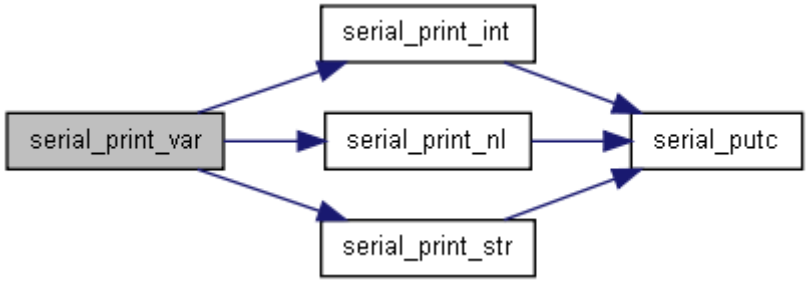
Here is the caller graph for this function:


ar1000_seek_more
ar1000_tune
audio_queue_clear
draw_print_buffer
draw_tests_run
drv_paint
ea_dsp0401b_display
i2c_send_byte
ds1631_convert_temp
ms5511_init
ms5511_get_config
its1_controller_handle_association
mrf24j40_set_extended_address
its1_controller_init
its1_device_init
serial_print_var
its1_device_process
mrf24j40_active_channel_scan
mrf24j40_handle_isr
ms5540_calc_temp_and_pressure
ms5540_init
pic_rf_init
pic_rf_transmit
pkt_queue_packet
pkt_process_rf_data
serial_print_str
pkt_send_payload
pkt_print_packet
pkt_send_packet
pkt_process_tx_queue
sht15_read
sht15_send_byte
sht15_read_temperature
usb_ep_data_out_callback
usb_ep_data_in_callback
usb_handle_transaction
usb_cdc_handle_tx
usb_send_data_chunk
usb_handle_standard_request
usb_configure_endpoints
usb_handle_class_ctrl_read_callback
usb_handle_class_ctrl_write_callback
usb_handle_class_request_callback
usb_send_data
usb_handle_reset
usb_handle_stall
wpan_print_address
wpan_data_received_callback
wpan_print_frame_type
wpan_print_mac_command

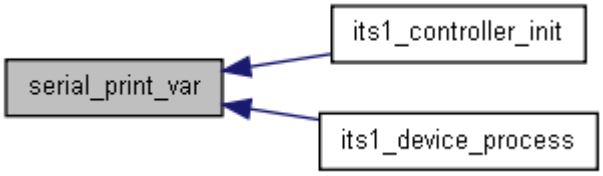
void serial_print_var (char * *str*, uns16 *i*)

```
429                                             {
430     serial_print_str(str);
431     serial_print_int(i);
432     serial_print_nl();
433 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void serial_putc (uns8 *c*)

Sends a single character out the serial connection. It is sent straight out if possible, otherwise put into the fifo. Note that if you fill the fifo while interrupts are off (eg, in an interrupt routine or a critical section) then this routine will hang the pic, since it's waiting for an interrupt to clear the fifo, which never comes... The moral is to keep your fifo big enough or don't send too much while interrupts are off (eg, in an interrupt response routine). Of course, you *can* send things in an ISR - just don't fill the fifo up.

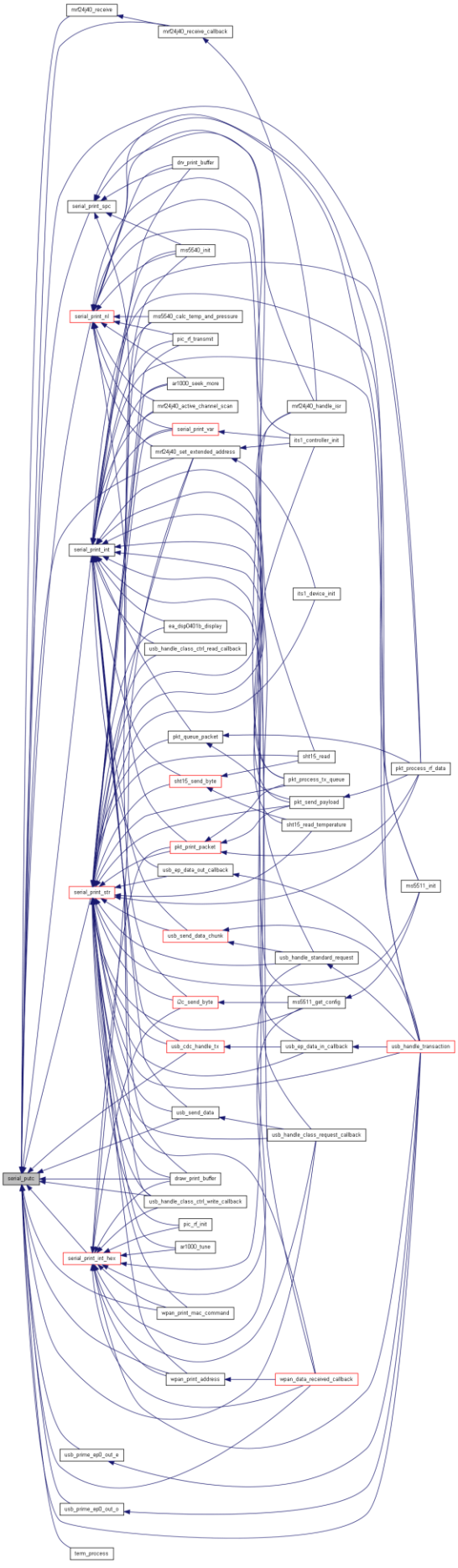
Parameters:

c the character to transmit

```
237 {
238 uns8 tx_next;
239 bit  my_store_gie;
240 #ifdef SERIAL IDE DEBUG
241 return;
242 #endif
243 
244     if ((tx_end == tx_start) &&  // Nothing in the fifo
245         test_bit(pir1, TXIF)) {  // And txreg is empty
246         txreg = c;               // then no need for fifo, just send straight out
247     } else { // else put it in the fifo
248         tx_next = tx_end + 1;   // get next buffer position
249         if (tx_next == SERIAL_TX_BUFFER_SIZE) { // if we're at the end
250             tx_next = 0;    // wrap to the beginning
251         }
252         #ifdef SERIAL_DISCARD_ON_TX_FULL_DURING_INT
253             if ((!intcon.GIE) && (tx_next == tx_start)) {
254                 return;
```

```
255             }
256         #endif
257         while (tx next == tx start) { // wait for clearing
258                                       // Note, if buffer is full
259                                       // this will wait for ever
260                                       // if interrupts are disabled!
261             #ifndef SERIAL_DISCARD_ON_TX_FULL_DURING_INT
262                 if (!intcon.GIE) {  // we're in an interrupt
263                     serial handle tx isr(); // so handle it ourselves
264                 }
265             #endif
266         }
267         my_store_gie = intcon.GIE;  // store interrupt state
268         kill interrupts();  // turn off global interrupts
269 
270         tx buffer[tx end] = c; // put it in
271         tx end = tx_next;  // move pointer along
272 
273         set_bit(pie1, TXIE); // turn on interrupt for transmitting
274         intcon.GIE = my_store_gie;  // restore interrupt state
275     } // -- else put it in the fifo
276 }
```

Here is the caller graph for this function:


mrf24j40_receive
mrf24j40_receive_callback
dbv_print_buffer
serial_print_spc
ms5540_init
serial_print_nl
ms5540_calc_temp_and_pressure
pic_rf_transmit
ar1000_seek_more
mrf24j40_active_channel_scan
mrf24j40_handle_isr
serial_print_var
its1_controller_init
mrf24j40_set_extended_address
serial_print_int
its1_device_init
ea_dip04016_display
usb_handle_class_ctrl_read_callback
pkt_queue_packet
sht15_read
pkt_process_rf_data
sht15_send_byte
pkt_process_tx_queue
pkt_send_payload
sht15_read_temperature
pkt_print_packet
usb_ep_data_out_callback
ms5611_init
serial_print_str
usb_send_data_chunk
usb_handle_standard_request
i2c_send_byte
ms5611_get_config
usb_cdc_handle_tx
usb_ep_data_in_callback
usb_handle_transaction
usb_send_data
usb_handle_class_request_callback
serial_putc
draw_print_buffer
usb_handle_class_ctrl_write_callback
pic_rf_init
ar1000_tune
serial_print_int_hex
wpan_print_mac_command
wpan_print_address
wpan_data_received_callback
usb_prime_ep0_out_s
usb_prime_ep0_out_n
term_process

uns8 serial_rx_avail (void)

Tests to see if the serial receive fifo has a character available. Useful to call before getc() if interrupts are not enabled in that section of code.

Returns:

true (non zero) if there are one or more characters waiting in the fifo queue, false (zero) otherwise

```
435 { return rx_start != rx_end; }
```

Here is the caller graph for this function:

void serial_rx_isr (void)

This routine needs to be called from your interrupt() routine when the receive hardware interrupt occurs in order to put received bytes into the fifo buffer.

```
301 {
302    uns8 rx_next;
303 
304 
305     if (test_bit(rcsta, OERR)) {    // overrun error?
306         clear_bit(rcsta, CREN); // clear error
307         _asm    {
308             MOVF    _rcreg,W    // clear any received characters
309             MOVF    _rcreg,W
310             MOVF    _rcreg,W
311         }
312         #ifdef SERIAL_DEBUG
313         rx_hard_overflow++; // increment error count if in debug mode
314         #endif
315         set_bit(rcsta, CREN);   // reset error indicator
316     } else {
317         if (test_bit(rcsta, FERR)) {    // framing error?
318             #ifdef SERIAL_DEBUG
319             rx_framing_error++; // increment error count if in debug mode
320             #endif
321         }
322         rx_next = rx_end + 1;   // get next buffer position
323         if (rx_next == SERIAL_RX_BUFFER_SIZE) { // if we're at the end
324             rx_next = 0;    // then wrap to the beginning
325         }
326         if (rx_next != rx_start) { // if space in the fifo
327             rx_buffer[rx_end] = rcreg; // put it in
328             rx_end = rx_next;  // and move pointer along
329         } else {    // else, there isn't space
330             _asm MOVF   _rcreg,W // and just clear it, we've lost it
331             #ifdef SERIAL_DEBUG
332             rx_soft_overflow++; // increment error count if in debug mode
333             #endif
334         }   // -- no space in the fifo
335     }   // -- no overrun error
336 }   // -- serial_load_rx
```

void serial_setup (uns8 *req_spbrg*)

Configures the pic and gets ready for interrupt-driven serial communications. Includes setting the tris bits appropriately, and getting the baud rate generator set up. After calling this you can immediately start sending and receiving bytes.

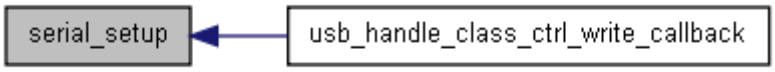
Parameters:

brgh See sprg defines earlier in pic_serial.h

```
143 {
144 #ifdef _PIC16F88
145     set_bit(trisb, 5);
146     set_bit(trisb, 2);
147     #define TRIS_SET
148 #endif
149 #ifdef  _PIC16F876A
150     set_bit(trisc,6);
151     set_bit(trisc,7);
152     #define TRIS_SET
153 #endif
154 #ifdef _PIC18F2620
155     set_bit(trisc,6);
156     set_bit(trisc,7);
157     #define TRIS_SET
158 #endif
159 #ifdef _PIC18F4520
160     set_bit(trisc,6);
161     set_bit(trisc,7);
162     #define TRIS_SET
163 #endif
164 #ifdef _PIC18F4550
165     set_bit(trisc,6);
166     set_bit(trisc,7);
167     #define TRIS_SET
168 #endif
169 #ifdef _PIC18F67J50
170     clear_bit(trisc,6);
171     set_bit(trisc,7);
172     #define TRIS_SET
173 #endif
174 #ifdef  _PIC18F25K20
175     set_bit(trisc,6);
176     set_bit(trisc,7);
177     #define TRIS_SET
178 #endif
179 #ifdef _PIC18F26K20
180     set_bit(trisc,6);
181     set_bit(trisc,7);
182     #define TRIS_SET
183 #endif
184 #ifdef _PIC18F14K50
185     set_bit(trisb,5);
186     #define TRIS_SET
187 #endif
188 #ifndef TRIS_SET
189     #warning "You must set tris bits for serial use yourself, I don't know your pic"
190     #warning "Please send your tris bits in so they can be included in the library"
191 #endif
192
193     //kill_interrupts();
194
195
196
197     txsta.BRGH = 1; // set baud rate generator (high/low speed)
198     #ifdef BRG16_REQUIRED
199         set_bit(baudcon, BRG16);
200         spbrg = req_spbrg & 0xff;
201         spbrgh = req_spbrg >> 8;
202     #else
203         spbrg = req spbrg;  // set serial port baud rate generator value
204     #endif
205     clear_bit(txsta, SYNC); // turn off syncronous reception
206     set_bit(rcsta, SPEN);   // enable serial port
207
208     // Configure tx
209
210     clear_bit(txsta, TX9);  // Turn off 9 bit receiption
```

```
211     clear_bit(txsta, TX9D); // Clear 9th bit data
212
213     set_bit(txsta, TXEN);   // enable sending of serial data
214
215     // configure rx
216
217     clear_bit(rcsta, RX9);  // disable 9 bit receiption
218     clear_bit(rcsta, FERR); // clear any framing errors
219
220     _asm    {   // clear rcreg buffer
221         MOVF     rcreg,W
222         MOVF    _rcreg,W
223         MOVF    _rcreg,W
224     }
225
226     clear_bit(rcsta, CREN);
227     set_bit(rcsta, CREN);   // pulse low to clear any errors
228
229     set_bit(pie1, RCIE);    // enable receive interrupt
230
231 }
```

Here is the caller graph for this function:



uns8 serial_tx_empty (void)

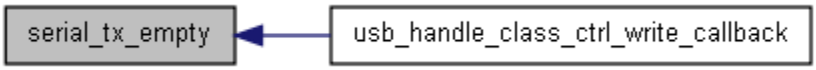
Tests to see if the serial transmit fifo is empty.

Returns:

true (non zero) if tx fifo is empty, false (zero) otherwise

```
436 { return tx_start == tx_end; }
```

Here is the caller graph for this function:

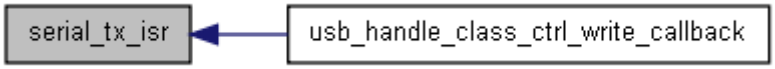


void serial_tx_isr (void)

This routine needs to be called from your interrupt() routine when the transmit hardware interrupt occurs in order to send bytes that are waiting in the fifo buffer.

```
281 {
282 uns8 tx_next;
283
284     if (tx_end == tx_start) { // anything in the fifo?
285        return; // nope
286     }
287     tx_next = tx_start + 1; // get next position
288     if (tx_next == SERIAL_TX_BUFFER_SIZE) { // if we're at the end of the buffer
289         tx_next = 0;    // wrap to the beginning
290     }
291     if (tx_end == tx_next) {    // if we've only got one character to send
292         clear_bit(pie1, TXIE);  // then turn off interrupts
293     }
294     txreg = tx_buffer[tx_start];    // transmit the character
295     tx_start = tx_next; // move start position of fifo
296
297 }   /* \ \  */
```

Here is the caller graph for this function:



Variable Documentation

uns8 rx_buffer[SERIAL_RX_BUFFER_SIZE]
Receive fifo

uns8 rx_end = 0
Receive fifo end point

uns8 rx_start = 0
Receive fifo start point

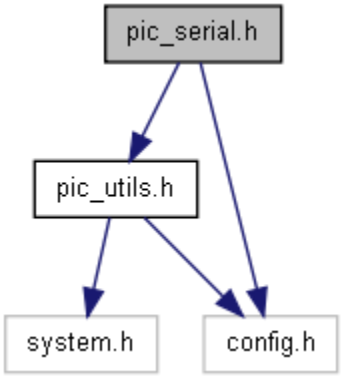
uns8 tx_buffer[SERIAL_TX_BUFFER_SIZE]
Transmit fifo

uns8 tx_end = 0
Transmit fifo end point

uns8 tx_start = 0
Transmit fifo start point

pic_serial.h File Reference

Interrupt driven fifo serial support.
Include dependency graph for pic_serial.h:



This graph shows which files directly or indirectly include this file:

Defines

- #define BRGH_HIGH_SPEED 1
- #define BRGH_LOW_SPEED 0
- #define serial_handle_rx_isr() if (pir1.RCIF) { serial_rx_isr(); pir1.RCIF = 0; }
- #define serial_handle_tx_isr() if (pir1.TXIF) { serial_tx_isr(); }
- #define serial_print_debug(string, variable) serial_print_str(string); serial_print_int(variable); serial_print_nl();

Functions

- uns8 even_7bit_parity (uns8 in)
- *Change 7th bit (MSB) to give even parity.* uns8 odd_7bit_parity (uns8 in)
- *Change 7th bit (MSB) to give od parity.* uns8 serial_getc (void)
- *Retrieve a character from the serial port.* void serial_print_int (uns16 i)
- *Print a 16 bit number to the serial port.* void serial_print_int_hex (uns8 i)
- *Print an 8 bit number in hex to the serial port.* void serial_print_int_hex_16bit (uns16 i)
- *Print a 16 bit number in hex to the serial port.* void serial_print_nl ()
- *Print a newline.* void serial_print_spc ()
- *Print a space.* void serial_print_str (char *str)
- *Print a string out to the serial port.* void serial_print_str_rom (rom char *str)
- *Print a rom string out to the serial port.* void serial_print_var (char *str, uns16 i)
- void serial_putc (uns8 c)
- *Transmit a single character.* uns8 serial_rx_avail (void)
- *Tests if the serial rx fifo has a character available.* void serial_rx_isr (void)
- *Serial receive interrupt service routine.* void serial_setup (uns8 req_spbrg)
- *Configure the pic for serial communicaitons.* uns8 serial_tx_empty (void)
- *Tests if the serial tx fifo is empty.* void serial_tx_full ()
- *Tests if the serial tx fifo is full.* void serial_tx_isr (void)

Serial transmit interrupt service routine.

Detailed Description

Put the following into your config.h

- - - - - - - - - - - - - - - - - - - pic_serial defines
- - - - - - - - - - - - - - - - - - -

define SERIAL_TX_BUFFER_SIZE 20 define SERIAL_RX_BUFFER_SIZE 4

Use this define if you want fine-grained control of what happens in the serial port define SERIAL_DEBUG_ON

Use this define if you are debugging in the IDE simulator and don't want it to hang waiting for serial interrupts that will never come... define SERIAL_IDE_DEBUG

Use thie define if you want to drop a character if the TX buffer is full, rather than the default behaviour, which is to wait until the TX buffer has a spare spot. define SERIAL_DISCARD_ON_TX_FULL_DURING_INT

- - - - - - - - - - - - - - - - - - -

Put the following in your ISR

serial_handle_tx_isr(); serial_handle_rx_isr();

Put the following in your system setup routine

serial_setup(uns8/16 req_spbrg);

Define Documentation

#define BRGH_HIGH_SPEED 1

#define BRGH_LOW_SPEED 0

#define serial_handle_rx_isr() if (pir1.RCIF) { serial_rx_isr(); pir1.RCIF = 0; }

include in your ISR

#define serial_handle_tx_isr() if (pir1.TXIF) { serial_tx_isr(); }

include in your ISR

#define serial_print_debug(string, variable) serial_print_str(string); serial_print_int(variable); serial_print_nl();

Function Documentation

uns8 even_7bit_parity (uns8 *in*)

```
63                                  {
64
65 uns8 t;
66
67  //return ((((((in)^(((in)<<4)|((in)>>4)))+0x41)|0x7C)+2)&0x80);
68
69
70 asm {
71
72     bcf    _in,7  ;assume the parity is even
73       ;Note: for odd parity, use bsf
74  ; assume the bits in byte to send are abcdefgh
75
76     swapf  _in,W  ;W = efghabcd
77     xorwf  _in,W  ;W = ea.fb.gc.hd.ea.fb.gc.hd
78                             ; where ea means e^a, etc
79     movwf  _t               ;
80     rlf    _t,F            ;t = fb.gc.hd.ea.fb.gc.hd.??
81     rlf    _t,F             ;t = gc.hd.ea.fb.gc.hd.??.ea
82     xorwf  _t,F             ;t = gcea.hdfb.gcea.hdfb.gcea.?.?
83                             ;again, gcea means g^c^e^a
84     rlf    _t,W             ;w = hdfb.gcea.hdfb.gcea.hdfb.?.fb
85     xorwf  _t,W             ;w = abcdefgh.abcdefgh.....
86                             ;ie, the upper 5-bits of w each contain
87                             ;the parity calculation.
88     andlw  0x80             ;We only need one of them
89     bcf   _in,7  ;set bit 7 to 0 in preparation for or-ing with our result
90     iorwf _in,F  ;copy it to the MSB of the byte to send.
91
92 }
93     return in;
94
95
96 }
```

uns8 odd_7bit_parity (uns8 *in*)

```
100                                 {
101
102 uns8 t;
```

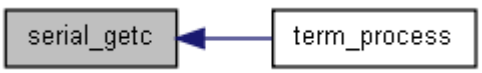
```
103
104 asm {
105     bsf     in,7  ;assume the parity is odd
106
107  ; assume the bits in "in" are abcdefgh
108
109     swapf  _in,W  ;W = efghabcd
110     xorwf  _in,W  ;W = ea.fb.gc.hd.ea.fb.gc.hd
111                           ; where ea means e^a, etc
112     movwf  _t              ;
113     rlf     t,F           ;t = fb.gc.hd.ea.fb.gc.hd.??
114     rlf    _t,F            ;t = gc.hd.ea.fb.gc.hd.??.ea
115     xorwf  _t,F            ;t = gcea.hdfb.gcea.hdfb.gcea.?.?
116                           ;again, gcea means g^c^e^a
117     rlf     t,W            ;w = hdfb.gcea.hdfb.gcea.hdfb.?.fb
118     xorwf  _t,W            ;w = abcdefgh.abcdefgh.....
119                           ;ie, the upper 5-bits of w each contain
120                           ;the parity calculation.
121     andlw  0x80           ;We only need one of them
122     bcf   _in,7  ;set bit 7 to 0 in preparation for or-ing with our result
123     iorwf _in,F  ;copy it to the MSB of the byte to send.
124 }
125     return in;
126 }
```

uns8 serial_getc (void)

Retrieve character from the serial port. Note that if there is nothing in the fifo, this function will wait until a character is received - and this will never happen if interrupts are turned off when this is called! So, be careful not to call getc during a critical section or during an ISR unless* you're sure there's something in the fifo. You can do this by calling the serial_rx_avail() routine. In any other situation, you can call getc() and happily wait for a character to arrive.

```
341 {
342     uns8 rx_char, rx_next;
343
344     while(rx_end == rx_start);  // wait until there is something received
345
346     start_crit_sec();   // make sure nobody else can muck with the buffer
347
348     rx_char = rx_buffer[rx_start];  // get character from the front of the buffer
349     rx_start++; // increment fifo start
350     if (rx_start == SERIAL_RX_BUFFER_SIZE) {    // if we're at the end
351         rx_start = 0;   // then wrap to the beginning
352     }
353
354     end_crit_sec(); // now they can muck with the buffer
355
356     return (rx_char);   // return the result we first thought of
357
358 }   // -- getc
```

Here is the caller graph for this function:



void serial_print_int (uns16 *i*)

Print a 16 bit unsigned number in decimal to the serial port

Parameters:

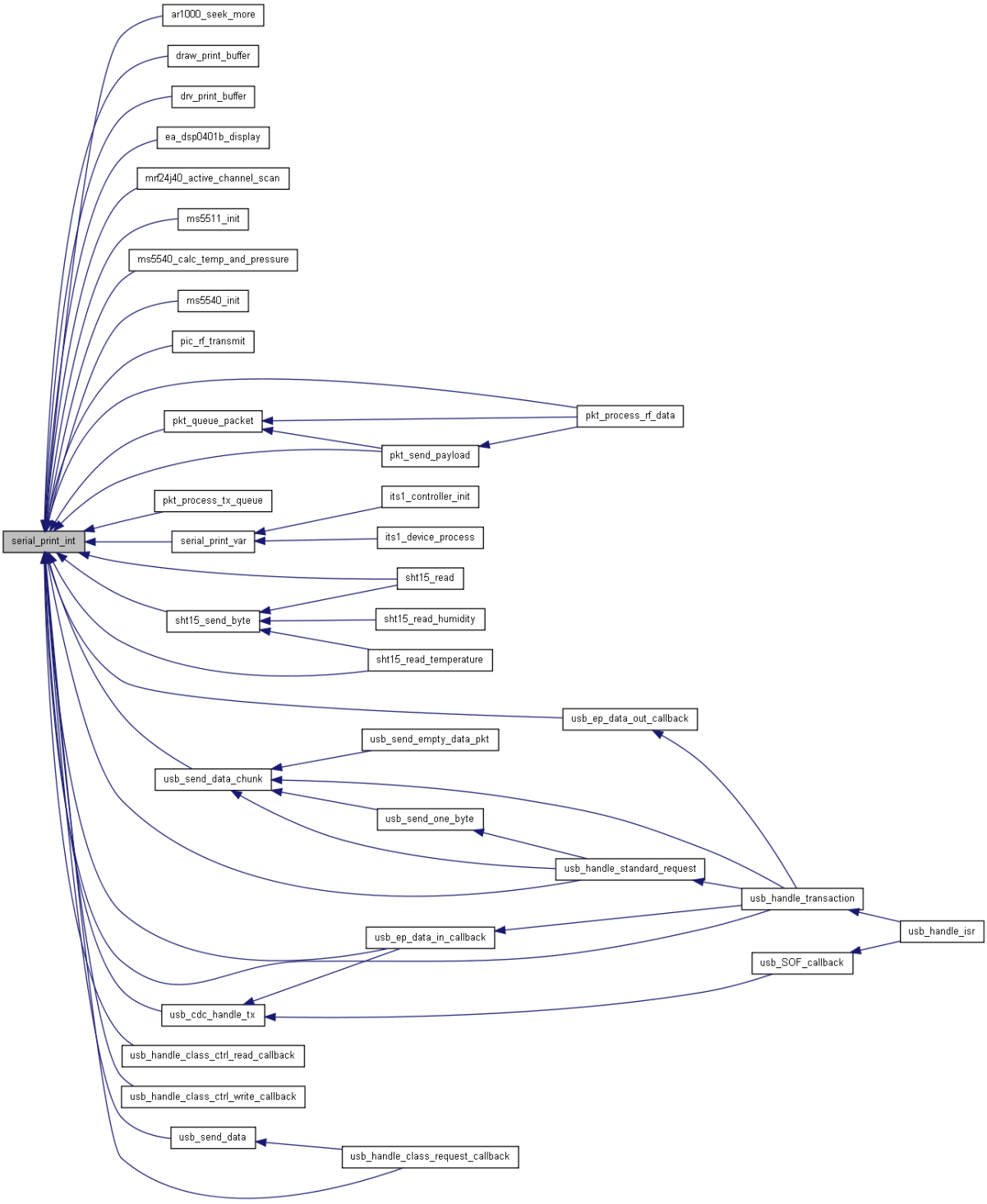
i the 16 bit number to be printed

```
386                                 {
387
```

```
388 char buffer[6]; // up to 5 characters plus \0
389 uns8 count = 5;
390     buffer[5] = '\0';
391     do {
392         count--;
393         buffer[count] = '0' + i % 10;
394         i = i / 10;
395     } while (i > 0);
396     while (buffer[count]) {
397         serial_putc(buffer[count]);
398         count++;
399     }
400     //serial_print_str(&buffer[count]); //  print it out
401 //  for(count = 0 ; str[count] != 0; count++)
402  //   {
403  //   }
404
405 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void serial_print_int_hex (uns8 *i*)

Print a 8 bit unsigned number in hex to the serial port

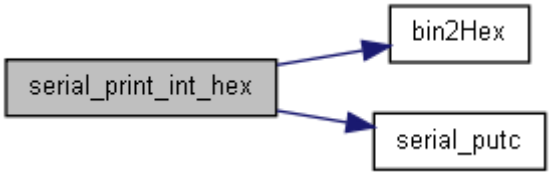
Parameters:

i 8 bit number to be printed

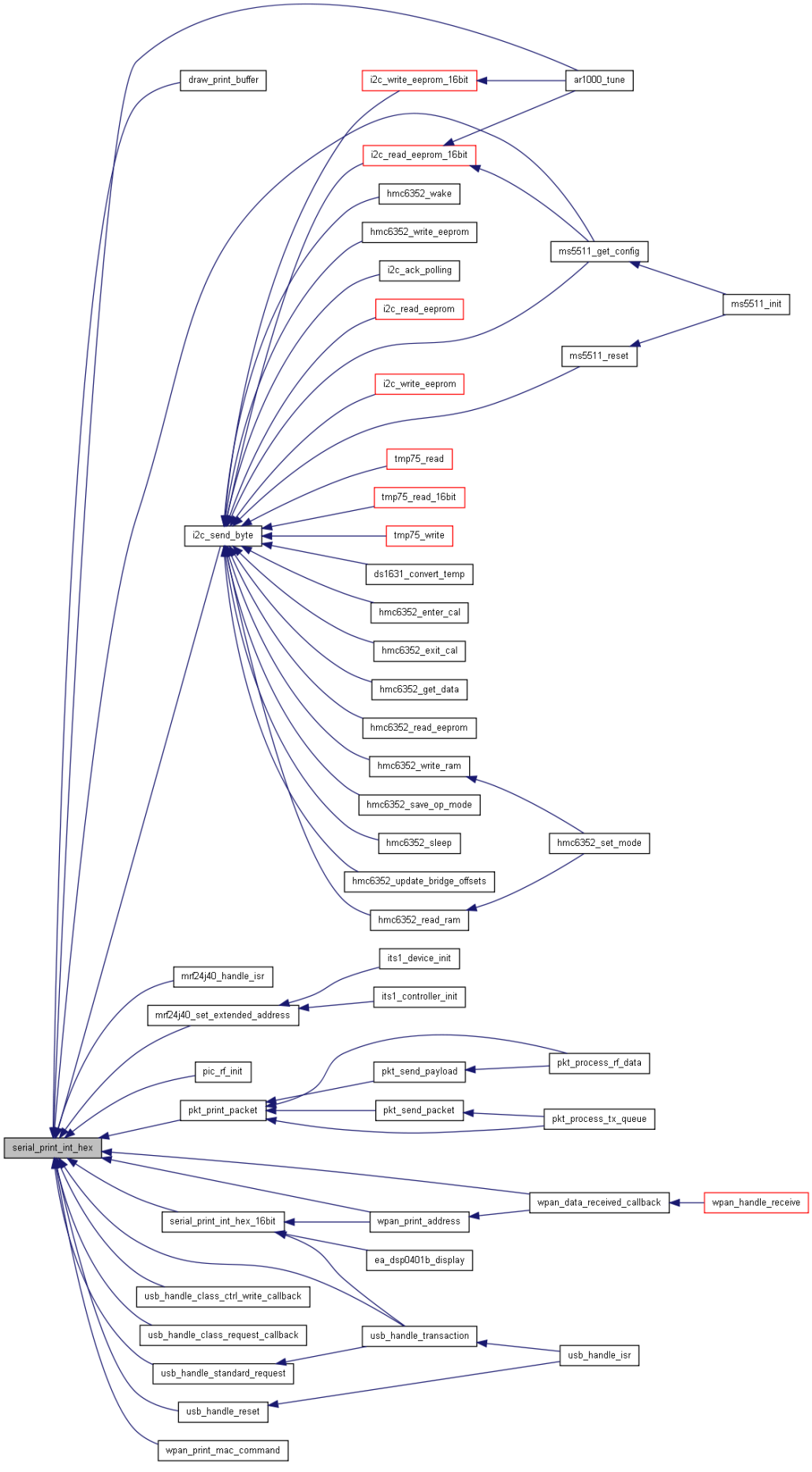
```
407                                     {
408
```

```
409     serial_putc(bin2Hex(i >> 4));
410     serial_putc(bin2Hex((i & 0x0f)));
411
412 }
```

Here is the call graph for this function:



Here is the caller graph for this function:


draw_print_buffer
i2c_write_eeprom_16bit
ar1000_tune
i2c_read_eeprom_16bit
hmc6352_wake
hmc6352_write_eeprom
ms5511_get_config
i2c_ack_polling
ms5511_init
i2c_read_eeprom
ms5511_reset
i2c_write_eeprom
tmp75_read
tmp75_read_16bit
i2c_send_byte
tmp75_write
ds1631_convert_temp
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_write_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_set_mode
hmc6352_update_bridge_offsets
hmc6352_read_ram
its1_device_init
mrf24j40_handle_isr
its1_controller_init
mrf24j40_set_extended_address
pic_rf_init
pkt_send_payload
pkt_process_rf_data
pkt_print_packet
pkt_send_packet
pkt_process_tx_queue
serial_print_int_hex
wpan_data_received_callback
wpan_handle_receive
serial_print_int_hex_16bit
wpan_print_address
ea_dsp0401b_display
usb_handle_class_ctrl_write_callback
usb_handle_class_request_callback
usb_handle_transaction
usb_handle_isr
usb_handle_standard_request
usb_handle_reset
wpan_print_mac_command

void serial_print_int_hex_16bit (uns16 *i*)

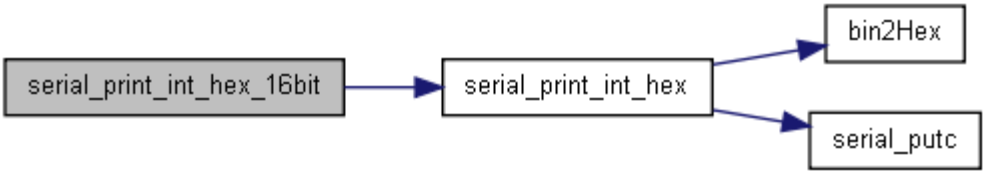
Print a 16 bit unsigned number in hex to the serial port

Parameters:

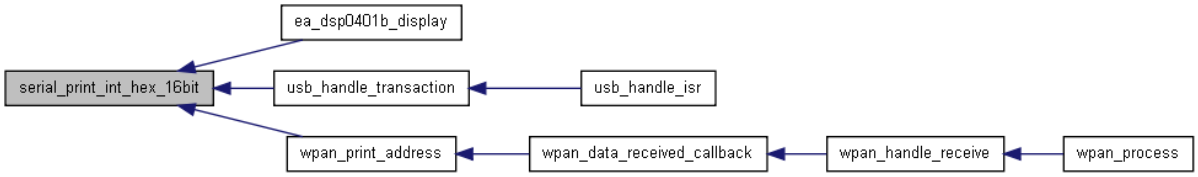
i 16 bit number to be printed

```
414                                             {
415     serial_print_int_hex(i >> 8);
416     serial_print_int_hex(i & 0xff);
417 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



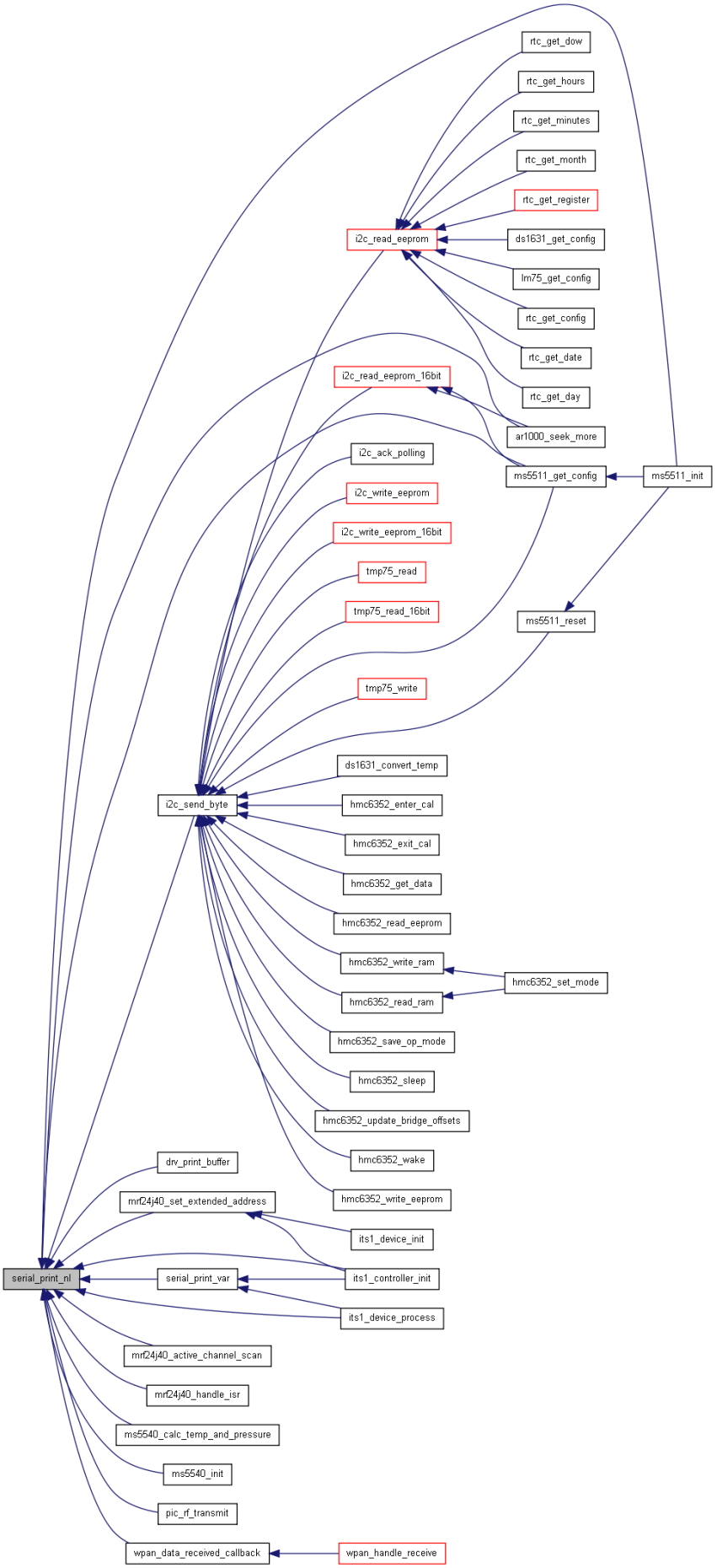
void serial_print_nl ()

Print a new line out the serial port - if you do this often, this routine can be used to save a couple of instructions. Always helps!

```
425                             {
426         serial_putc('\n');
427 }
```

Here is the call graph for this function:

Here is the caller graph for this function:


rtc_get_dow
rtc_get_hours
rtc_get_minutes
rtc_get_month
rtc_get_register
i2c_read_eeprom
ds1631_get_config
lm75_get_config
rtc_get_config
rtc_get_date
rtc_get_day
i2c_read_eeprom_16bit
ar1000_seek_more
ms5511_get_config
ms5511_init
i2c_ack_polling
i2c_write_eeprom
i2c_write_eeprom_16bit
tmp75_read
tmp75_read_16bit
ms5511_reset
tmp75_write
ds1631_convert_temp
i2c_send_byte
hmc6352_enter_cal
hmc6352_exit_cal
hmc6352_get_data
hmc6352_read_eeprom
hmc6352_write_ram
hmc6352_set_mode
hmc6352_read_ram
hmc6352_save_op_mode
hmc6352_sleep
hmc6352_update_bridge_offsets
hmc6352_wake
hmc6352_write_eeprom
drv_print_buffer
mrf24j40_set_extended_address
its1_device_init
serial_print_nl
serial_print_var
its1_controller_init
its1_device_process
mrf24j40_active_channel_scan
mrf24j40_handle_isr
ms5540_calc_temp_and_pressure
ms5540_init
pic_rf_transmit
wpan_data_received_callback
wpan_handle_receive

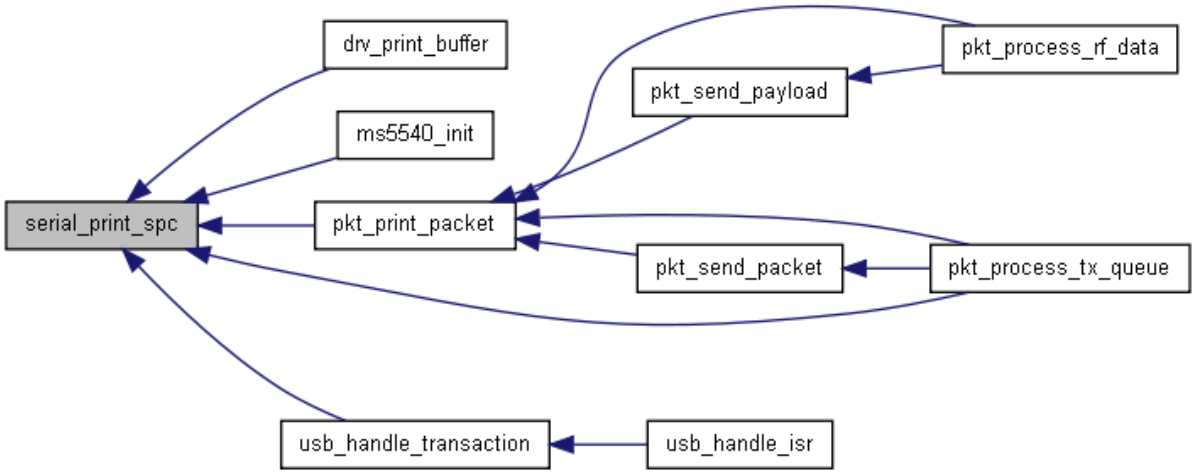
void serial_print_spc ()

Print a space out the serial port - if you do this often, this routine can be used to save a couple of instructions. Always helps!

```
420                             {
421         serial_putc(' ');
422 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void serial_print_str (char * *str*)

Send a null terminated string out the serial port

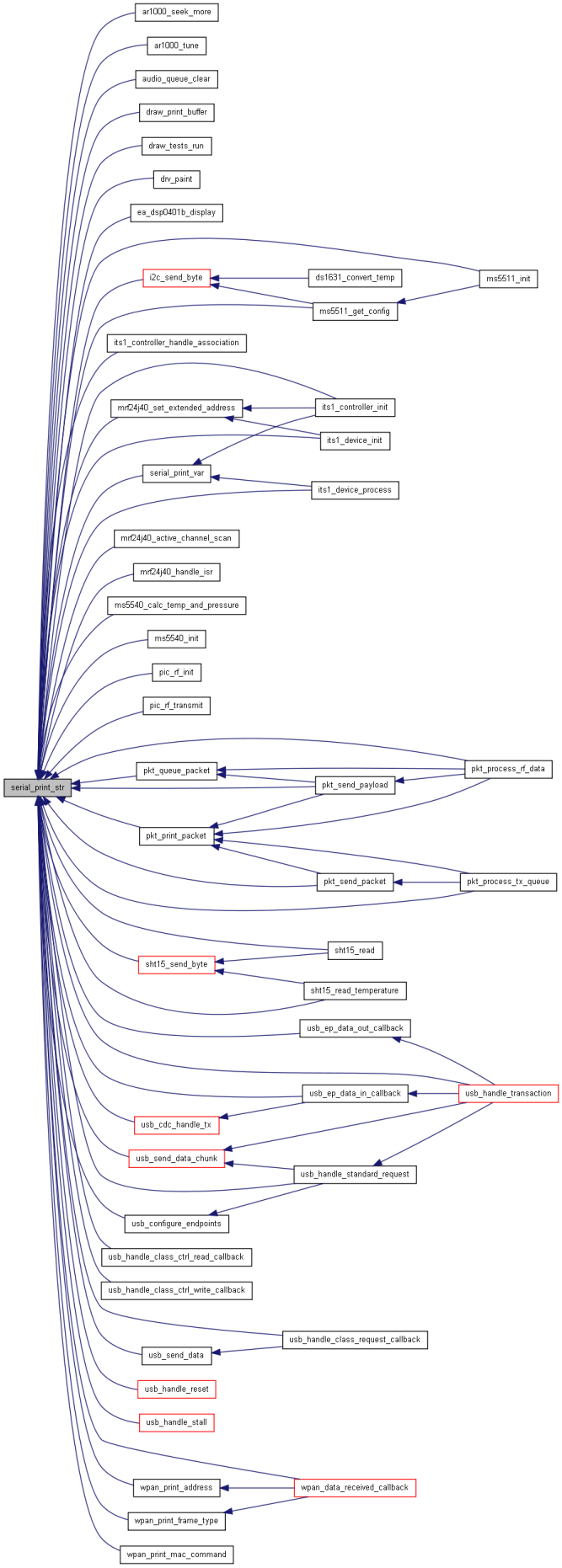
Parameters:

str the string to be sent

```
362                                     {
363
364 uns8 count;
365
366     for(count = 0 ; str[count] != 0; count++)
367     {
368         serial_putc(str[count]);
369     }
370 }
```

Here is the call graph for this function:

Here is the caller graph for this function:


ar1000_seek_more
ar1000_tune
audio_queue_clear
draw_print_buffer
draw_tests_run
drv_paint
ea_dsp0401b_display
i2c_send_byte
ds1631_convert_temp
ms5511_init
ms5511_get_config
its1_controller_handle_association
mrf24j40_set_extended_address
its1_controller_init
its1_device_init
serial_print_var
its1_device_process
mrf24j40_active_channel_scan
mrf24j40_handle_isr
ms5540_calc_temp_and_pressure
ms5540_init
pic_rf_init
pic_rf_transmit
serial_print_str
pkt_queue_packet
pkt_process_rf_data
pkt_send_payload
pkt_print_packet
pkt_send_packet
pkt_process_tx_queue
sht15_read
sht15_send_byte
sht15_read_temperature
usb_ep_data_out_callback
usb_ep_data_in_callback
usb_handle_transaction
usb_cdc_handle_tx
usb_send_data_chunk
usb_handle_standard_request
usb_configure_endpoints
usb_handle_class_ctrl_read_callback
usb_handle_class_ctrl_write_callback
usb_handle_class_request_callback
usb_send_data
usb_handle_reset
usb_handle_stall
wpan_print_address
wpan_data_received_callback
wpan_print_frame_type
wpan_print_mac_command

void serial_print_str_rom (rom char * *str*)

Send a null terminated rom string out the serial port

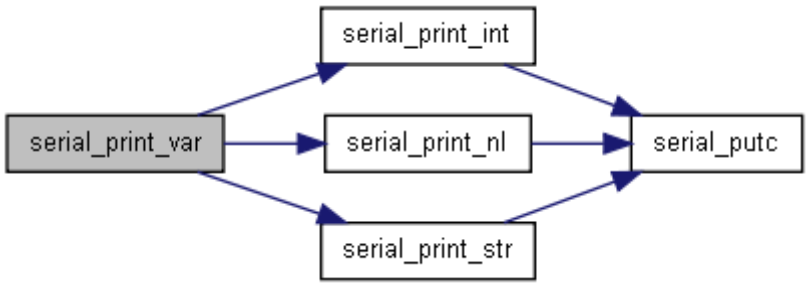
Parameters:

str the rom string to be sent

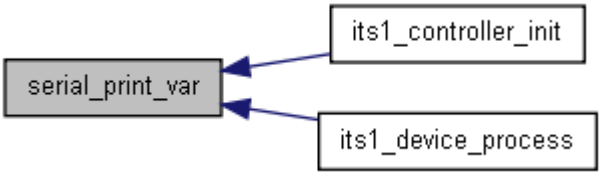
void serial_print_var (char * *str*, uns16 *i*)

```
429                                             {
430     serial_print_str(str);
431     serial_print_int(i);
432     serial_print_nl();
433 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void serial_putc (uns8 *c*)

Sends a single character out the serial connection. It is sent straight out if possible, otherwise put into the fifo. Note that if you fill the fifo while interrupts are off (eg, in an interrupt routine or a critical section) then this routine will hang the pic, since it's waiting for an interrupt to clear the fifo, which never comes... The moral is to keep your fifo big enough or don't send too much while interrupts are off (eg, in an interrupt response routine). Of course, you *can* send things in an ISR - just don't fill the fifo up.

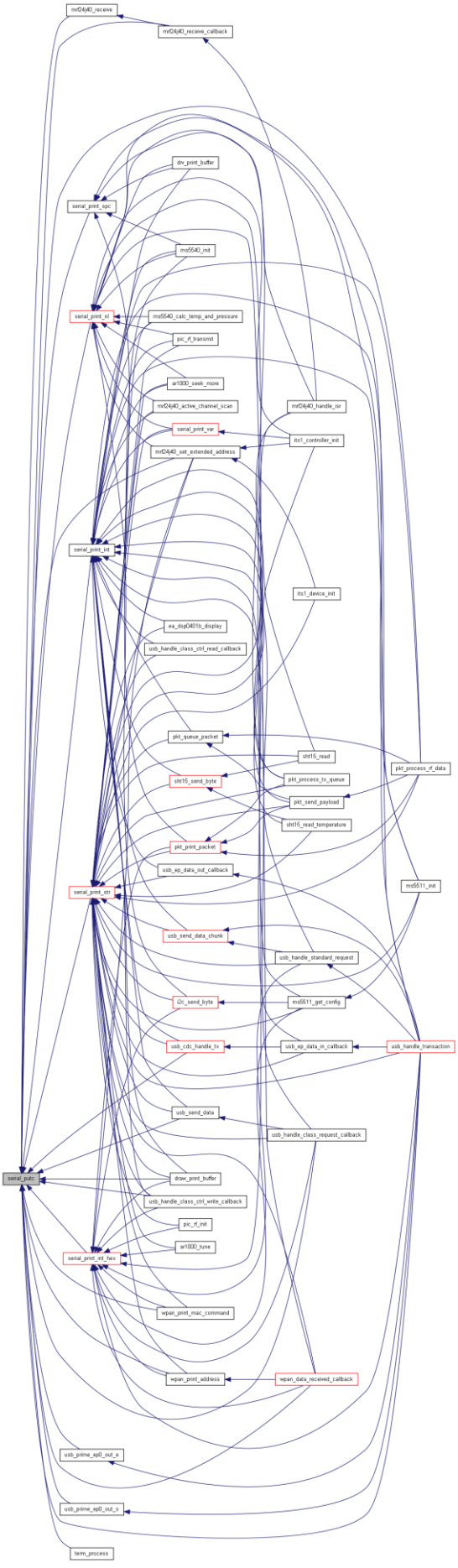
Parameters:

c the character to transmit

```
237 {
238 uns8 tx_next;
239 bit  my_store_gie;
240 #ifdef SERIAL_IDE_DEBUG
241 return;
242 #endif
243 
244     if ((tx_end == tx_start) &&  // Nothing in the fifo
245         test_bit(pir1, TXIF)) {  // And txreg is empty
246         txreg = c;               // then no need for fifo, just send straight out
247     } else { // else put it in the fifo
```

```
248         tx_next = tx_end + 1;   // get next buffer position
249         if (tx_next == SERIAL_TX_BUFFER_SIZE) { // if we're at the end
250             tx_next = 0;    // wrap to the beginning
251         }
252         #ifdef SERIAL_DISCARD_ON_TX_FULL_DURING_INT
253             if ((!intcon.GIE) && (tx_next == tx_start)) {
254                 return;
255             }
256         #endif
257         while (tx_next == tx_start) { // wait for clearing
258                                       // Note, if buffer is full
259                                       // this will wait for ever
260                                       // if interrupts are disabled!
261             #ifndef SERIAL_DISCARD_ON_TX_FULL_DURING_INT
262                 if (!intcon.GIE) {  // we're in an interrupt
263                     serial_handle_tx_isr(); // so handle it ourselves
264                 }
265             #endif
266         }
267         my_store_gie = intcon.GIE;  // store interrupt state
268         kill_interrupts();  // turn off global interrupts
269 
270         tx_buffer[tx_end] = c; // put it in
271         tx_end = tx_next;  // move pointer along
272 
273         set_bit(pie1, TXIE); // turn on interrupt for transmitting
274         intcon.GIE = my_store_gie;  // restore interrupt state
275     } // -- else put it in the fifo
276 }
```

Here is the caller graph for this function:


mrf24j40_receive
mrf24j40_receive_callback
dbv_print_buffer
serial_print_spc
ms5540_init
serial_print_nl
ms5540_calc_temp_and_pressure
pic_rf_transmit
ar1000_seek_more
mrf24j40_active_channel_scan
mrf24j40_handle_isr
serial_print_var
its1_controller_init
mrf24j40_set_extended_address
serial_print_int
its1_device_init
ea_dog04016_display
usb_handle_class_ctrl_read_callback
pkt_queue_packet
sht15_read
pkt_process_rf_data
sht15_send_byte
pkt_process_tx_queue
pkt_send_payload
sht15_read_temperature
pkt_print_packet
usb_ep_data_out_callback
ms5611_init
serial_print_str
usb_send_data_chunk
usb_handle_standard_request
i2c_send_byte
ms5611_get_config
usb_cdc_handle_tx
usb_ep_data_in_callback
usb_handle_transaction
usb_send_data
usb_handle_class_request_callback
serial_putc
draw_print_buffer
usb_handle_class_ctrl_write_callback
pic_rf_init
ar1000_tune
serial_print_int_hex
wpan_print_mac_command
wpan_print_address
wpan_data_received_callback
usb_prime_ep0_out_s
usb_prime_ep0_out_n
term_process

uns8 serial_rx_avail (void)

Tests to see if the serial receive fifo has a character available. Useful to call before getc() if interrupts are not enabled in that section of code.

Returns:

true (non zero) if there are one or more characters waiting in the fifo queue, false (zero) otherwise

```
435 { return rx_start != rx_end; }
```

Here is the caller graph for this function:

void serial_rx_isr (void)

This routine needs to be called from your interrupt() routine when the receive hardware interrupt occurs in order to put received bytes into the fifo buffer.

```
301 {
302    uns8 rx_next;
303 
304 
305     if (test_bit(rcsta, OERR)) {    // overrun error?
306         clear_bit(rcsta, CREN); // clear error
307         _asm    {
308             MOVF    _rcreg,W    // clear any received characters
309             MOVF    _rcreg,W
310             MOVF    _rcreg,W
311         }
312         #ifdef SERIAL_DEBUG
313         rx_hard_overflow++; // increment error count if in debug mode
314         #endif
315         set_bit(rcsta, CREN);   // reset error indicator
316     } else {
317         if (test_bit(rcsta, FERR)) {    // framing error?
318             #ifdef SERIAL_DEBUG
319             rx_framing_error++; // increment error count if in debug mode
320             #endif
321         }
322         rx_next = rx_end + 1;   // get next buffer position
323         if (rx_next == SERIAL_RX_BUFFER_SIZE) { // if we're at the end
324             rx_next = 0;    // then wrap to the beginning
325         }
326         if (rx_next != rx_start) { // if space in the fifo
327             rx_buffer[rx_end] = rcreg; // put it in
328             rx_end = rx_next;  // and move pointer along
329         } else {    // else, there isn't space
330             _asm MOVF   _rcreg,W // and just clear it, we've lost it
331             #ifdef SERIAL_DEBUG
332             rx_soft_overflow++; // increment error count if in debug mode
333             #endif
334         }   // -- no space in the fifo
335     }   // -- no overrun error
336 }   // -- serial_load_rx
```

void serial_setup (uns8 *req_spbrg*)

Configures the pic and gets ready for interrupt-driven serial communications. Includes setting the tris bits appropriately, and getting the baud rate generator set up. After calling this you can immediately start sending and receiving bytes.

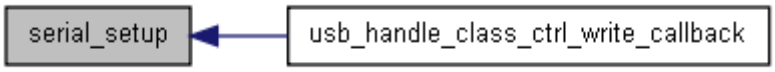
Parameters:

brgh See sprg defines earlier in pic_serial.h

```
143 {
144 #ifdef _PIC16F88
145     set_bit(trisb, 5);
146     set_bit(trisb, 2);
147     #define TRIS_SET
148 #endif
149 #ifdef  _PIC16F876A
150     set_bit(trisc,6);
151     set_bit(trisc,7);
152     #define TRIS_SET
153 #endif
154 #ifdef _PIC18F2620
155     set_bit(trisc,6);
156     set_bit(trisc,7);
157     #define TRIS_SET
158 #endif
159 #ifdef _PIC18F4520
160     set_bit(trisc,6);
161     set_bit(trisc,7);
162     #define TRIS_SET
163 #endif
164 #ifdef _PIC18F4550
165     set_bit(trisc,6);
166     set_bit(trisc,7);
167     #define TRIS_SET
168 #endif
169 #ifdef _PIC18F67J50
170     clear_bit(trisc,6);
171     set_bit(trisc,7);
172     #define TRIS_SET
173 #endif
174 #ifdef  _PIC18F25K20
175     set_bit(trisc,6);
176     set_bit(trisc,7);
177     #define TRIS_SET
178 #endif
179 #ifdef _PIC18F26K20
180     set_bit(trisc,6);
181     set_bit(trisc,7);
182     #define TRIS_SET
183 #endif
184 #ifdef _PIC18F14K50
185     set_bit(trisb,5);
186     #define TRIS_SET
187 #endif
188 #ifndef TRIS_SET
189     #warning "You must set tris bits for serial use yourself, I don't know your pic"
190     #warning "Please send your tris bits in so they can be included in the library"
191 #endif
192
193     //kill_interrupts();
194
195
196
197     txsta.BRGH = 1; // set baud rate generator (high/low speed)
198     #ifdef BRG16_REQUIRED
199         set_bit(baudcon, BRG16);
200         spbrg = req_spbrg & 0xff;
201         spbrgh = req_spbrg >> 8;
202     #else
203         spbrg = req spbrg;  // set serial port baud rate generator value
204     #endif
205     clear_bit(txsta, SYNC); // turn off syncronous reception
206     set_bit(rcsta, SPEN);   // enable serial port
207
208     // Configure tx
209
210     clear_bit(txsta, TX9);  // Turn off 9 bit receiption
```

```
211     clear_bit(txsta, TX9D); // Clear 9th bit data
212
213     set bit(txsta, TXEN);   // enable sending of serial data
214
215     // configure rx
216
217     clear_bit(rcsta, RX9);  // disable 9 bit receiption
218     clear_bit(rcsta, FERR); // clear any framing errors
219
220     _asm    {   // clear rcreg buffer
221         MOVF     rcreg,W
222         MOVF    _rcreg,W
223         MOVF    _rcreg,W
224     }
225
226     clear_bit(rcsta, CREN);
227     set_bit(rcsta, CREN);   // pulse low to clear any errors
228
229     set_bit(pie1, RCIE);    // enable receive interrupt
230
231 }
```

Here is the caller graph for this function:



uns8 serial_tx_empty (void)

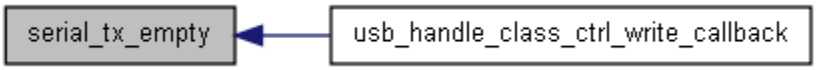
Tests to see if the serial transmit fifo is empty.

Returns:

true (non zero) if tx fifo is empty, false (zero) otherwise

```
436 { return tx_start == tx_end; }
```

Here is the caller graph for this function:



void serial_tx_full ()

Tests to see if the serial transmit fifo is full.

Returns:

true (non zero) if tx fifo is full, false (zero) otherwise

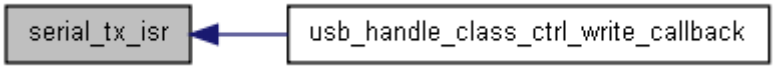
void serial_tx_isr (void)

This routine needs to be called from your interrupt() routine when the transmit hardware interrupt occurs in order to send bytes that are waiting in the fifo buffer.

```
281 {
282 uns8 tx_next;
283
284     if (tx_end == tx_start) { // anything in the fifo?
285        return; // nope
286     }
287     tx_next = tx_start + 1; // get next position
288     if (tx_next == SERIAL_TX_BUFFER_SIZE) { // if we're at the end of the buffer
289         tx_next = 0;    // wrap to the beginning
290     }
291     if (tx_end == tx_next) {    // if we've only got one character to send
292         clear_bit(pie1, TXIE);  // then turn off interrupts
```

```
293     }
294     txreg = tx_buffer[tx_start];    // transmit the character
295     tx_start = tx_next; // move start position of fifo
296
297 }   /* \ \  */
```

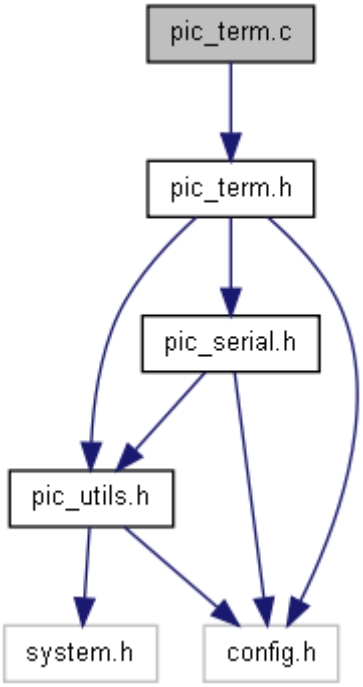
Here is the caller graph for this function:



---

pic_term.c File Reference

Include dependency graph for pic_term.c:



Functions

- void term_init ()
- void term_process ()

Variables

- uns8 buffer_len
- uns8 term_buffer [TERM_BUFFER_SIZE]

Function Documentation

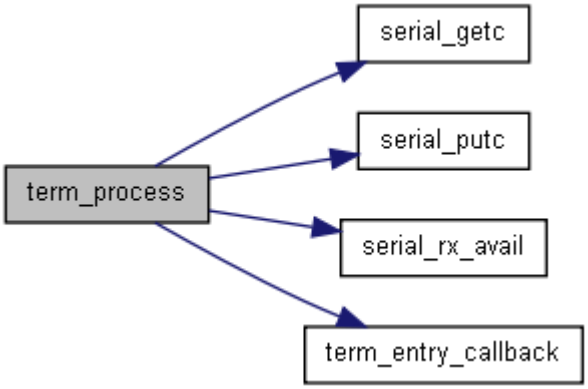
void term_init ()

```
82                  {
83
84      term_buffer[0] = '\0';
85      buffer_len = 0;
86
87  }
```

void term_process ()

```
43                      {
44
45      uns8 len,   // length of string
46           rec;   // received character
47
48      if (serial_rx_avail()) {
49
50          rec = serial_getc();    // get the character from the fifo
51
52          #ifdef TERM_ALLOW_BOOSTBLOADER
53              if (rec == MAGIC_BOOSTBLOADER_REQUEST) {
54                  boostbloader();
55              }
56          #endif
57
58
59          if (rec == '\r') {  // did we press return?
60
61              if (buffer_len > 0) {
62                  term_entry_callback(term_buffer);
63              }
64              term_buffer[0] = '\0';  // clear the buffer
65              buffer_len = 0;
66              #ifdef TERM_ECHO_INPUT
67                  serial_putc('\n');  // print new line
68                  serial_putc('>');   // print the prompt
69              #endif
70          } else {
71              if (buffer_len < TERM_BUFFER_SIZE + 2) {    // space for null and new character
72                  #ifdef TERM_ECHO_INPUT
73                      serial_putc(rec);   // print it out so we can see what we typed
74                  #endif
75                  term_buffer[buffer_len++] = rec;
76                  term_buffer[buffer_len] = '\0';
77              }
78          }
79      }
80  }
```

Here is the call graph for this function:



Variable Documentation

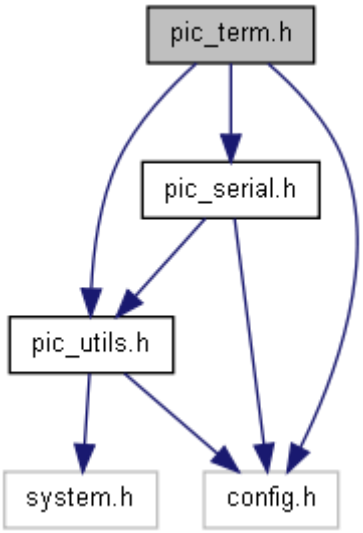
uns8 buffer_len

uns8 term_buffer[TERM_BUFFER_SIZE]

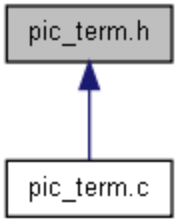
---

pic_term.h File Reference

Pic terminal routines.
Include dependency graph for pic_term.h:



This graph shows which files directly or indirectly include this file:



Functions

- void term_entry_callback (uns8 *term_buffer)
- void term_init ()
- void term_process ()

Detailed Description

Allows the use of a serial terminal for interaction with the PIC

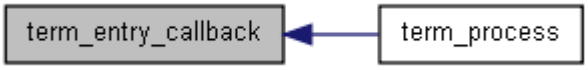
Put the following into your config.h

```
// - - - - - - - - - - - - - - - - - - - -
// pic_term defines
// - - - - - - - - - - - - - - - - - - - -

#define TERM_BUFFER_SIZE    10

// Reset (go to bootloader) if magic character received
#define TERM_ALLOW_BOOSTBLOADER
// Echo typing so user can see what they're doing
#define TERM_ECHO_INPUT

// - - - - - - - - - - - - - - - - - - - -

// Create term_entry_callback in your own code
// void term_entry_callback(uns8 *term_buffer);
```

Function Documentation

void term_entry_callback (uns8 * *term_buffer*)

Here is the caller graph for this function:



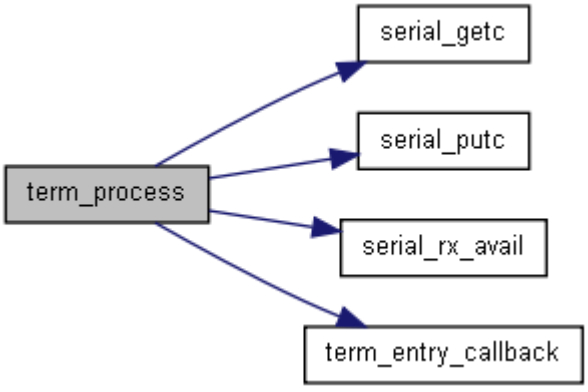
void term_init ()

```
82                  {
83
84      term_buffer[0] = '\0';
85      buffer_len = 0;
86
87  }
```

void term_process ()

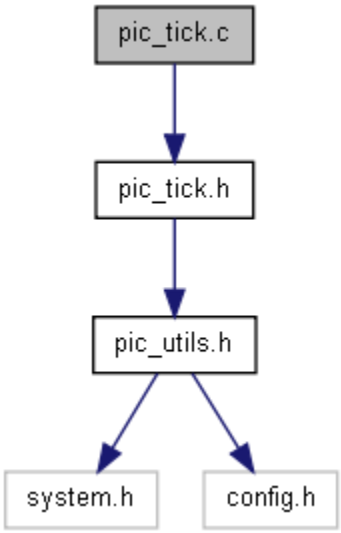
```
43                      {
44
45      uns8 len,   // length of string
46           rec;   // received character
47
48      if (serial_rx_avail()) {
49
50          rec = serial_getc();    // get the character from the fifo
51
52          #ifdef TERM_ALLOW_BOOSTBLOADER
53              if (rec == MAGIC_BOOSTBLOADER_REQUEST) {
54                  boostbloader();
55              }
56          #endif
57
58
59          if (rec == '\r') {  // did we press return?
60
61              if (buffer_len > 0) {
62                  term_entry_callback(term_buffer);
63              }
64              term_buffer[0] = '\0';  // clear the buffer
65              buffer_len = 0;
66              #ifdef TERM_ECHO_INPUT
67                  serial_putc('\n');  // print new line
68                  serial_putc('>');   // print the prompt
69              #endif
70          } else {
71              if (buffer_len < TERM_BUFFER_SIZE + 2) {    // space for null and new character
72                  #ifdef TERM_ECHO_INPUT
73                      serial_putc(rec);   // print it out so we can see what we typed
74                  #endif
75                  term_buffer[buffer_len++] = rec;
76                  term_buffer[buffer_len] = '\0';
77              }
78          }
79      }
80  }
```

Here is the call graph for this function:



pic_tick.c File Reference

Include dependency graph for pic_tick.c:



Functions

- void handle_tick ()
- *Call this routine to increment tick count.* uns16 tick_calc_diff (uns16 a, uns16 b)
- *Calculate the tick time difference between two values.* uns16 tick_get_count ()
- *Return current tick count.* void timer_0_callback ()

Timer 0 callback function.

Function Documentation

void handle_tick ()

Typically called during the interrupt routine of a timer to increment the tick count. Note this routine assumes that interupts are off - which is always the case in an interrupt sub routine.

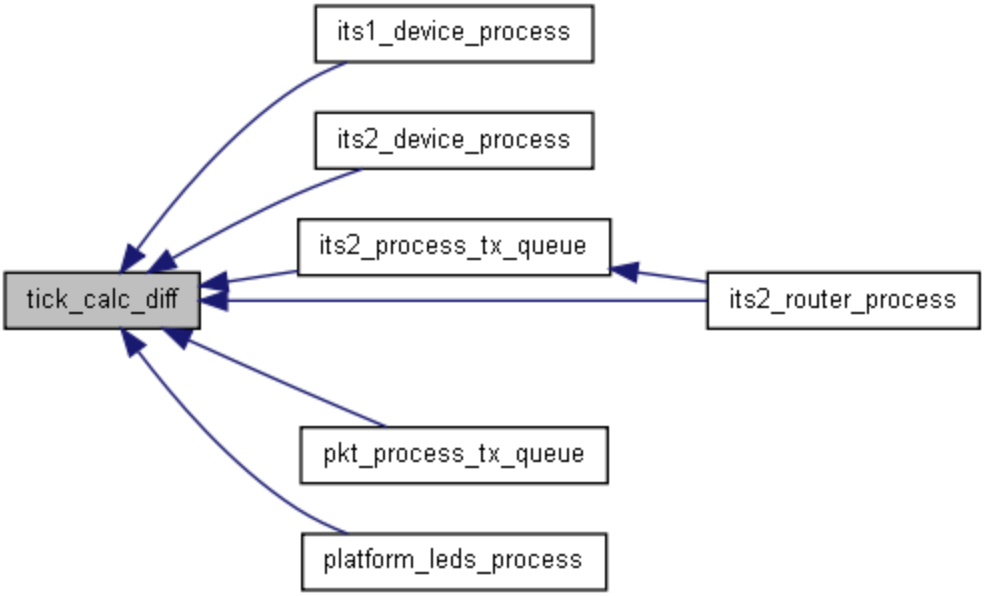
```
66                      {
67     tick++;  // we assume that interrupts are off at this point
68 }
```

uns16 tick_calc_diff (uns16 *a*, uns16 *b*)

Calculates how many ticks have elapsed between two tick values. Covers cases where the tick count wraps beyond its 16 bit value.

```
58                                      {
59     if (a <= b) {   // simple case
60         return b-a;
61     } else {
62         return 65535 - a + b + 1;   // more complex case
63     }
64 }
```

Here is the caller graph for this function:

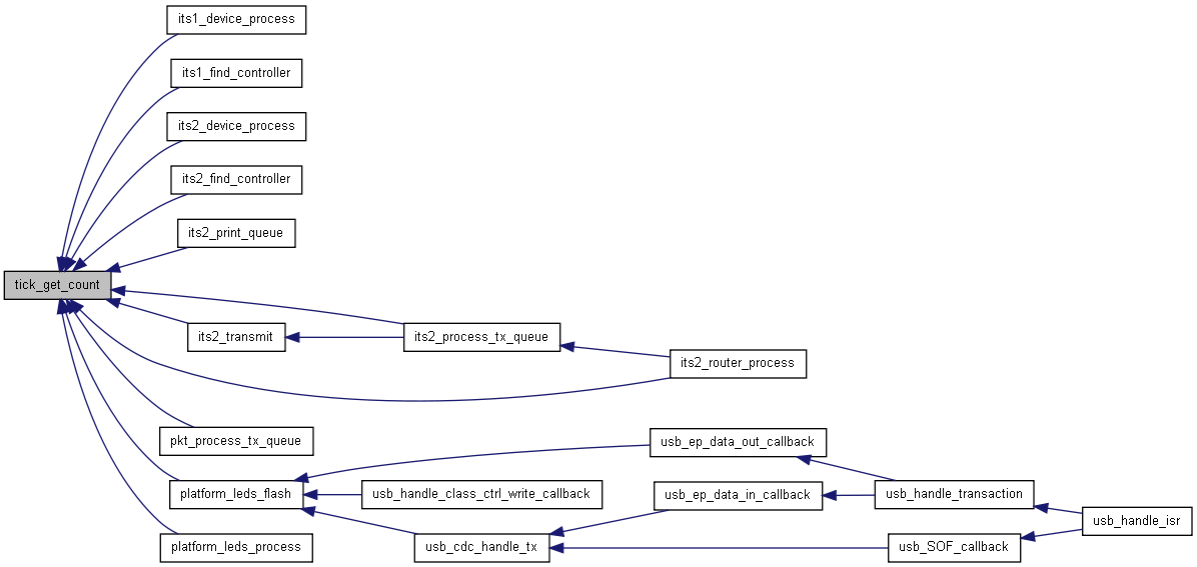


uns16 tick_get_count ()

Returns the current tick count. Thread and interrupt safe.

```
47                          {
48
49 uns16 result;
50
51     start_crit_sec();
52     result = tick;  // Grab a copy
53
54     end_crit_sec(); // interrupts back to normal
55     return result;  // return the result
56 }
```

Here is the caller graph for this function:



void timer_0_callback ()

When a timer 0 interrupt occurs, after handling the interupt and timing issues, this callback function is executed. You will need to define this subroutine in your code, otherwise linking will fail.

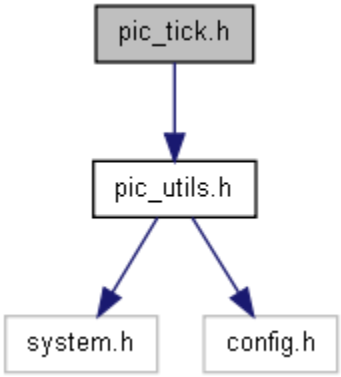
```
43                          {
44      handle_tick_inline();
45  }
```

Here is the call graph for this function:

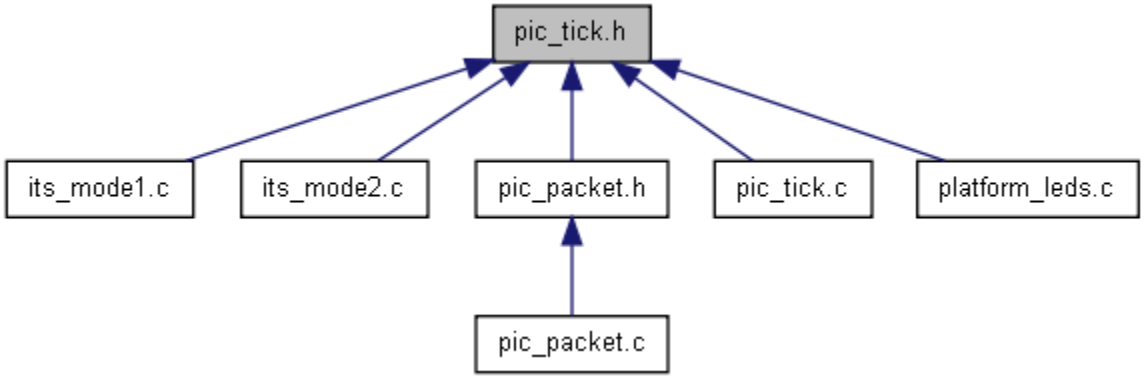
---

pic_tick.h File Reference

Timer helper routines.
Include dependency graph for pic_tick.h:



This graph shows which files directly or indirectly include this file:



Functions

- void handle_tick ()
- *Call this routine to increment tick count.* void handle_tick_inline ()

- *Call this routine to increment tick count - inline version.* uns16 tick_calc_diff (uns16 a, uns16 b)
- *Calculate the tick time difference between two values.* uns16 tick_get_count ()

Return current tick count. Variables

- static uns16 tick = 0

Detailed Description

Function Documentation

void handle_tick ()

Typically called during the interrupt routine of a timer to increment the tick count. Note this routine assumes that interupts are off - which is always the case in an interrupt sub routine.

```
66                      {
67     tick++;  // we assume that interrupts are off at this point
68 }
```

void handle_tick_inline () [inline]

Typically called during the interrupt routine of a timer to increment the tick count. Note this routine assumes that interupts are off - which is always the case in an interrupt sub routine. Inline version so you don't use up one stack level

```
84                                      {
85     tick++;  // we assume that interrupts are off at this point
86 }
```

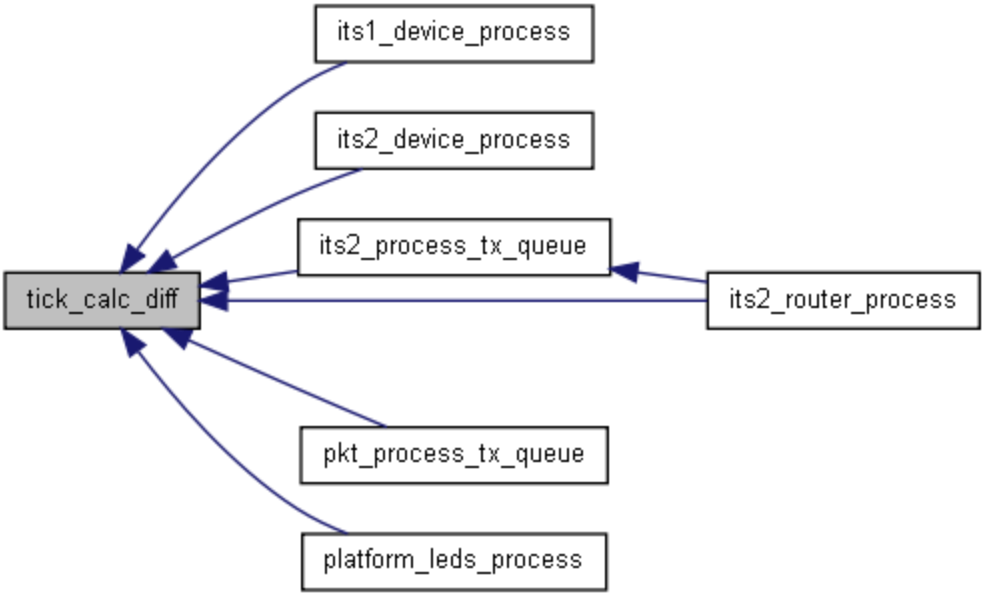
Here is the caller graph for this function:

uns16 tick_calc_diff (uns16 *a*, uns16 *b*)

Calculates how many ticks have elapsed between two tick values. Covers cases where the tick count wraps beyond its 16 bit value.

```
58                                              {
59     if (a <= b) {   // simple case
60         return b-a;
61     } else {
62         return 65535 - a + b + 1;   // more complex case
63     }
64 }
```

Here is the caller graph for this function:

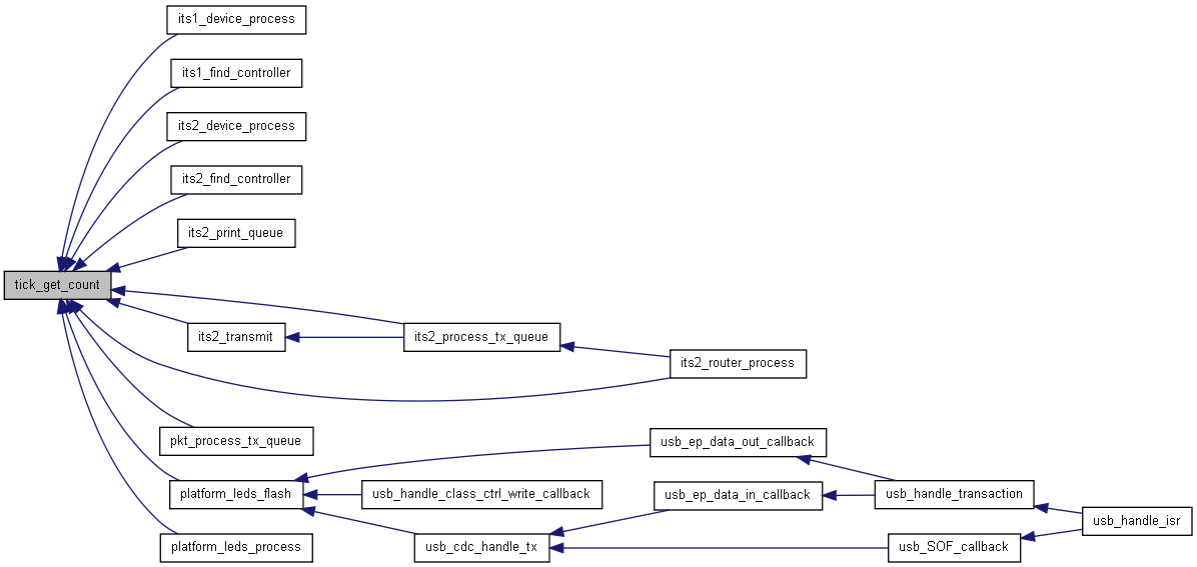


uns16 tick_get_count ()

Returns the current tick count. Thread and interrupt safe.

```
47                          {
48
49 uns16 result;
50
51     start_crit_sec();
52     result = tick;  // Grab a copy
53
54     end_crit_sec(); // interrupts back to normal
55     return result;  // return the result
56 }
```

Here is the caller graph for this function:



Variable Documentation

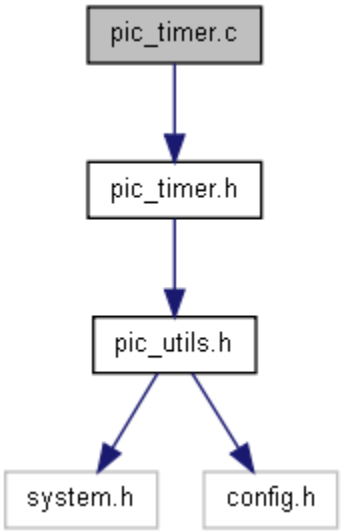
uns16 tick = 0 `[static]`

Global tick counter

---

pic_timer.c File Reference

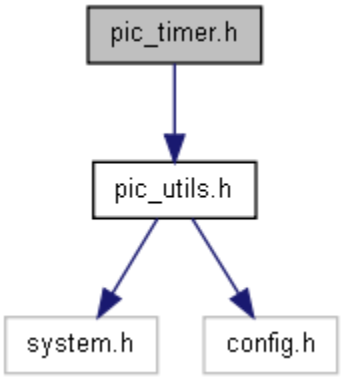
Include dependency graph for pic_timer.c:



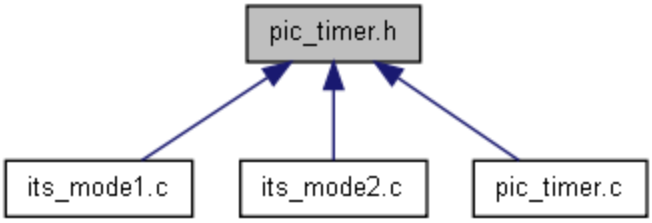
---

pic_timer.h File Reference

Access to timer 0.
Include dependency graph for pic_timer.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define TIMER_16BIT_MODE 0
- #define TIMER_8BIT_MODE 1
- #define TIMER_PRESCALER_1_TO_128 0x06
- #define TIMER_PRESCALER_1_TO_16 0x03
- #define TIMER_PRESCALER_1_TO_2 0x00
- #define TIMER_PRESCALER_1_TO_256 0x07
- #define TIMER_PRESCALER_1_TO_32 0x04
- #define TIMER_PRESCALER_1_TO_4 0x01
- #define TIMER_PRESCALER_1_TO_64 0x05
- #define TIMER_PRESCALER_1_TO_8 0x02
- #define TIMER_PRESCALER_OFF 0xff

Functions

- void timer_0_callback ()
- *Timer 0 callback function.* void timer_setup_0 (bit mode_16_bit, uns8 prescaler_setting, uns16 timer_start_value)
- *Setup timer zero with starting values.* void timer_start_0 ()
- *Start timer 0.* void timer_stop_0 ()

Stop timer 0.

Detailed Description

Define Documentation

#define TIMER_16BIT_MODE 0

Timer mode for devices where this is applicable (16bit timer)

#define TIMER_8BIT_MODE 1

Timer mode for devices where this is applicable (8bit timer)

#define TIMER_PRESCALER_1_TO_128 0x06

#define TIMER_PRESCALER_1_TO_16 0x03

#define TIMER_PRESCALER_1_TO_2 0x00

#define TIMER_PRESCALER_1_TO_256 0x07

#define TIMER_PRESCALER_1_TO_32 0x04

#define TIMER_PRESCALER_1_TO_4 0x01

#define TIMER_PRESCALER_1_TO_64 0x05

#define TIMER_PRESCALER_1_TO_8 0x02

#define TIMER_PRESCALER_OFF 0xff

Function Documentation

void timer_0_callback ()

When a timer 0 interrupt occurs, after handling the interupt and timing issues, this callback function is executed. You will need to define this subroutine in your code, otherwise linking will fail.

```
43                          {
44      handle_tick_inline();
45  }
```

Here is the call graph for this function:

void timer_setup_0 (bit *mode_16_bit*, uns8 *prescaler_setting*, uns16 *timer_start_value*)

Turns off timer zero, congures 16/8bit mode (only for 18f devices), prescaler setting and start value (which will be loaded on each reset.

void timer_start_0 ()

Kicks off timer 0. In pic18 devices this will turn the timer on, on pic16 devices this will turn on timer0 interrupts.

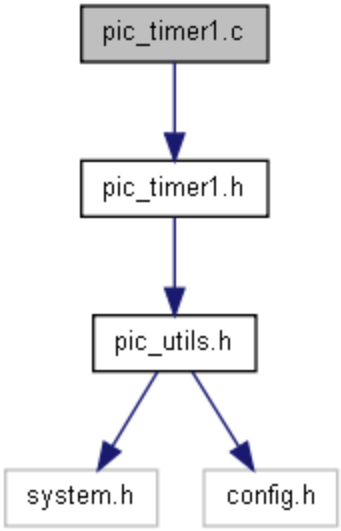
void timer_stop_0 ()

Stops timer 0. In pic18 devices, this will switch the timer off. On pic16 devices this will merely turn off the interrupt and the timer will continue running.

---

pic_timer1.c File Reference

Include dependency graph for pic_timer1.c:



Functions

- void timer_setup_1 (uns8 prescaler_setting, uns16 timer_start_value)
- *Setup timer 1 with starting values.* void timer_start_1 ()
- *Start timer 1.* void timer_stop_1 ()

Stop timer 1. Variables

- uns16 timer_1_start_value = 0

Function Documentation

void timer_setup_1 (uns8 *prescaler_setting*, uns16 *timer_start_value*)

Turns off timer 1, sets prescaler setting and start value (which will be loaded on each reset).

```
43                                                                          {
44
45      clear_bit(t1con, TMR1ON);    // turn off timer if it was on so we can get it set up
46      clear_bit(t1con, TMR1CS);    // Internal instruction cycle clock
47      #ifdef _PIC18
48          set_bit(t1con, RD16);
49      #endif
50      t1con &= 0b11001111;
51      t1con |= prescaler_setting;
52      timer_1_start_value = timer_start_value;
53      set_bit(pie1, TMR1IE);  // Turn on timer 0 interrupts
54  }
```

void timer_start_1 ()

Kicks off timer 1.

```
56                          {
57      tmr1h = timer_1_start_value >> 8;
58      tmr1l = timer_1_start_value & 0xff;
59      set_bit(t1con, TMR1ON);
60  }
```

void timer_stop_1 ()

Stops timer 1.

```
63                      {
64     clear_bit(t1con, TMR1ON);
65 }
```

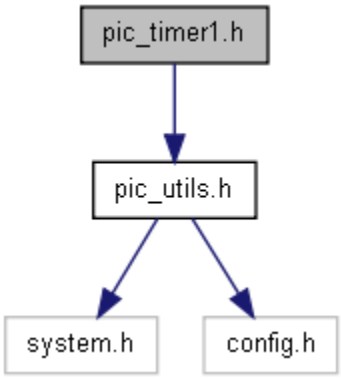
Variable Documentation

uns16 timer_1_start_value = 0

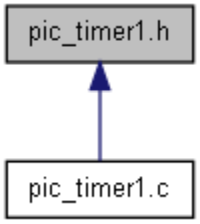
---

pic_timer1.h File Reference

Timer 1 support.
Include dependency graph for pic_timer1.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define TIMER1_PRESCALER_1_TO_2 0b00010000
- #define TIMER1_PRESCALER_1_TO_4 0b00100000
- #define TIMER1_PRESCALER_1_TO_8 0b00110000
- #define TIMER1_PRESCALER_OFF 0b00000000

Functions

- void timer_1_callback ()

- *Timer 1 callback function.* void timer_handle_1_isr ()
- *handle timer 1 in interrupt service routine* void timer_setup_1 (uns8 prescaler_setting, uns16 timer_start_value)
- *Setup timer 1 with starting values.* void timer_start_1 ()
- *Start timer 1.* void timer_stop_1 ()

Stop timer 1. Variables

- uns16 timer_1_start_value

Detailed Description

Define Documentation

#define TIMER1_PRESCALER_1_TO_2 0b00010000

#define TIMER1_PRESCALER_1_TO_4 0b00100000

#define TIMER1_PRESCALER_1_TO_8 0b00110000

#define TIMER1_PRESCALER_OFF 0b00000000

Function Documentation

void timer_1_callback ()

When a timer 1 interrupt occurs, after handling the interupt and timing issues, this callback function is executed. You will need to define this subroutine in your code, otherwise linking will fail.

Here is the caller graph for this function:

void timer_handle_1_isr () [inline]

Call this routine in your interrupt subroutine to automatically service timer 1 interrupts if they have occursed.

```
94                                  {
95     uns16 start_value;
96     if (test_bit(pir1, TMR1IF)) {    // interrupt?
97         #ifdef _PIC16
98             start_value = tmr1l + timer_1_start_value  + 8; // adjust start value
99             tmr1h = start_value >> 8;    // set high value
100             tmr1l = start_value & 0xff; // set low value (must be done in this order)
101         #else
102             start_value = tmr1l + timer_1_start_value  + 8; // adjust start value
103 
104             tmr1h = start_value >> 8;    // set high value
105             tmr1l = start_value & 0xff; // set low value (must be done in this order)
106         #endif
107         timer_1_callback(); // call the callback
108         clear_bit( pir1, TMR1IF ); //clear timer 0 interrupt bit
109     }
110 }
```

Here is the call graph for this function:

void timer_setup_1 (uns8 *prescaler_setting*, uns16 *timer_start_value*)

Turns off timer 1, sets prescaler setting and start value (which will be loaded on each reset).

```
43                                                                          {
44
45      clear bit(t1con, TMR1ON);   // turn off timer if it was on so we can get it set up
46      clear_bit(t1con, TMR1CS);   // Internal instruction cycle clock
47      #ifdef _PIC18
48          set_bit(t1con, RD16);
49      #endif
50      t1con &= 0b11001111;
51      t1con |= prescaler_setting;
52      timer_1_start_value = timer_start_value;
53      set_bit(pie1, TMR1IE);  // Turn on timer 0 interrupts
54  }
```

void timer_start_1 ()

Kicks off timer 1.

```
56                      {
57      tmr1h = timer_1_start_value >> 8;
58      tmr1l = timer_1_start_value & 0xff;
59      set_bit(t1con, TMR1ON);
60  }
```

void timer_stop_1 ()

Stops timer 1.

```
63                      {
64      clear_bit(t1con, TMR1ON);
65  }
```

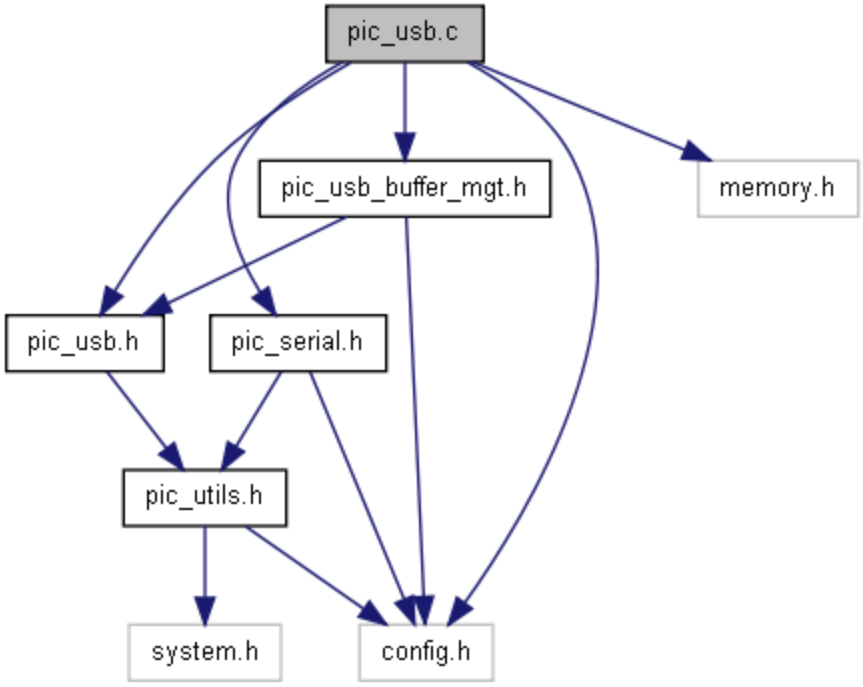
Variable Documentation

uns16 timer_1_start_value

---

pic_usb.c File Reference

Include dependency graph for pic_usb.c:



Functions

- void turn_usb_ints_on ()
- *Turn on USB interrupts.* void usb_configure_endpoints ()
- void usb_enable_module ()
- *Enables the USB hardware and starts USB negotiations.* usb_state_type usb_get_state ()
- *Query the current state of the USB connection.* void usb_handle_isr ()
- *Handle USB interrupts.* void usb_handle_reset ()
- void usb_handle_stall ()
- void usb_handle_standard_request (setup_data_packet sdp)
- void usb_handle_transaction (uns8 stat)
- void usb_prime_ep0_out_e ()
- void usb_prime_ep0_out_o ()
- void usb_send_data (uns8 ep, uns8 *data, uns8 send_count, bit first)
- *Send data over an endpoint pipe.* void usb_send_data_chunk ()
- void usb_send_empty_data_pkt ()
- *Send an empty data packet.* void usb_send_one_byte (uns8 data)
- void usb_setup ()
- *Setup USB hardware ready for use.* void usb_stall_ep0 ()
- *Send a stall on control transfer endpoint.* void usb_stall_on_in ()

Variables

- uns8 buffer_byte
- control_mode_type control_mode
- buffer_descriptor * delivery_bd
- uns8 * delivery_buffer
- uns8 delivery_buffer_size
- uns16 delivery_bytes_max_send
- uns16 delivery_bytes_sent

- uns16 delivery_bytes_to_send
- uns8 * delivery_ptr
- uns8 usb_address
- setup_data_packet usb_sdp
- usb_state_type usb_state = st_POWERED
- usb_status_type usb_status

Function Documentation

void turn_usb_ints_on ()

If you are using interrupt-driven code (generally the best way of doing things) you can turn on USB interrupts using turn_usb_ints_on(). Don't forget that you will also need to call turn_global_ints_on() as well. Typically this is called in your system setup routine.

```
962                          {
963
964     set_bit(uie,  STALLIE); // interrupt on stall
965     set_bit(uie,  TRNIE);   //   on transaction complete
966     set_bit(uie,  URSTIE);  //   on reset
967     set_bit(pie2, USBIE);   // general USB interrupts
968     #ifdef USB_CALLBACK_ON_SOF
969         set_bit(uie, SOFIE);
970     #endif
971 }
```

void usb_configure_endpoints ()

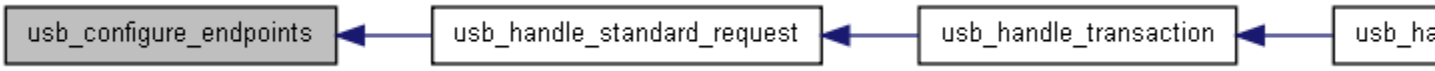
```
66                                  {
67
68 #ifdef USB_DEBUG
69     serial print str("Config eps ");
70 #endif
71 #ifdef USB_EP1
72     set_bit  (uep1, EPHSHK);    // EP handshaking on
73     #ifdef USB EP1 OUT SIZE
74         set_bit(uep1, EPOUTEN); // EP OUT enabled
75     #else
76         clear_bit(uep1, EPOUTEN);   // EP OUT disabled
77     #endif
78     #ifdef USB_EP1_IN_SIZE
79         set_bit(uep1, EPINEN);  // EP IN enabled
80     #else
81         clear bit(uep1, EPINEN);    // EP IN disabled
82     #endif
83     set_bit  (uep1, EPCONDIS);  // control transfers off
84
85     // for IN
86     #ifdef USB_EP1_IN_SIZE
87         set_bit(bd1in.stat, DTS);   // turn on data togle sync TOGGLE
88         clear_bit(bd1in.stat, KEN); // clear the keep bit
89         clear_bit(bd1in.stat, INCDIS);  // clear the increment disable
90         clear_bit(bd1in.stat, DTSEN);
91         clear_bit(bd1in.stat, BSTALL);  // clear stall bit
92         clear_bit(bd1in.stat, BC9);
93         clear_bit(bd1in.stat, BC8);
94
95         clear_bit(bd1in.stat, UOWN);    // uC owns the buffer
96     #endif
97     // for OUT
98     #ifdef USB EP1 OUT SIZE
99         bd1out.count = USB_EP1_OUT_SIZE;
100         bd1out.addr  = USB_EP1_OUT_ADDR;
```

```
101
102         clear_bit(bd1out.stat, DTS);    // turn on data togle sync TOGGLE
103         clear_bit(bd1out.stat, KEN);    // clear the keep bit
104         clear_bit(bd1out.stat, INCDIS); // clear the increment disable
105         clear_bit(bd1out.stat, DTSEN);
106         clear_bit(bd1out.stat, BSTALL); // clear stall bit
107         clear_bit(bd1out.stat, BC9);
108         clear_bit(bd1out.stat, BC8);
109         set_bit  (bd1out.stat, UOWN);   // SIE owns the buffer
110
111     #endif
112
113
114 #endif
115
116 #ifdef USB_EP2
117     set_bit  (uep2, EPHSHK);    // EP handshaking on
118     #ifdef USB_EP2_OUT_SIZE
119         set_bit(uep2, EPOUTEN); // EP OUT enabled
120     #else
121         clear_bit(uep2, EPOUTEN);   // EP OUT disabled
122     #endif
123     #ifdef USB_EP2_IN_SIZE
124         set_bit(uep2, EPINEN);  // EP IN enabled
125     #else
126         clear_bit(uep2, EPINEN);    // EP IN disabled
127     #endif
128     set bit  (uep2, EPCONDIS);  // control transfers off
129
130     // for IN
131     #ifdef USB_EP2_IN_SIZE
132
133         set_bit(bd2in.stat, DTS);   // turn on data togle sync TOGGLE
134         clear_bit(bd2in.stat, KEN); // clear the keep bit
135         clear_bit(bd2in.stat, INCDIS);  // clear the increment disable
136         clear_bit(bd2in.stat, DTSEN);
137         clear_bit(bd2in.stat, BSTALL);  // clear stall bit
138         clear_bit(bd2in.stat, BC9);
139         clear_bit(bd2in.stat, BC8);
140
141         clear_bit(bd2in.stat, UOWN);    // uC owns the buffer
142     #endif
143     // for OUT
144     #ifdef USB EP2 OUT SIZE
145         bd2out.count = USB_EP2_OUT_SIZE;
146         bd2out.addr  = USB_EP2_OUT_ADDR;
147
148         clear_bit(bd2out.stat, DTS);    // turn on data togle sync TOGGLE
149         clear_bit(bd2out.stat, KEN);    // clear the keep bit
150         clear_bit(bd2out.stat, INCDIS); // clear the increment disable
151         clear_bit(bd2out.stat, DTSEN);
152         clear_bit(bd2out.stat, BSTALL); // clear stall bit
153         clear_bit(bd2out.stat, BC9);
154         clear_bit(bd2out.stat, BC8);
155         set_bit  (bd2out.stat, UOWN);   // SIE owns the buffer
156     #endif
157 #endif
158
159 #ifdef USB_EP3
160     set_bit  (uep3, EPHSHK);    // EP handshaking on
161     #ifdef USB EP3 OUT SIZE
162         set_bit(uep3, EPOUTEN); // EP OUT enabled
163     #else
164         clear_bit(uep3, EPOUTEN);   // EP OUT disabled
165     #endif
166     #ifdef USB_EP3_IN_SIZE
167         set_bit(uep3, EPINEN);  // EP IN enabled
168     #else
169         clear bit(uep3, EPINEN);    // EP IN disabled
170     #endif
171     set_bit  (uep3, EPCONDIS);  // control transfers off
```

```
172     // for IN
173     #ifdef USB_EP3_IN_SIZE
174         set bit(bd3in.stat, DTS);   // ready for toggle
175         clear_bit(bd3in.stat, KEN); // clear the keep bit
176         clear_bit(bd3in.stat, INCDIS);  // clear the increment disable
177         clear_bit(bd3in.stat, DTSEN);
178         clear_bit(bd3in.stat, BSTALL);  // clear stall bit
179         clear_bit(bd3in.stat, BC9);
180         clear_bit(bd3in.stat, BC8);
181
182         clear bit(bd3in.stat, UOWN);    // uC owns the buffer
183     #endif
184     // for OUT
185     #ifdef USB_EP3_OUT_SIZE
186         bd3out.count = USB EP3 OUT SIZE;
187         bd3out.addr  = USB_EP3_OUT_ADDR;
188
189         clear_bit(bd3out.stat, DTS);    // turn on data togle sync TOGGLE
190         clear_bit(bd3out.stat, KEN);    // clear the keep bit
191         clear_bit(bd3out.stat, INCDIS); // clear the increment disable
192         clear_bit(bd3out.stat, DTSEN);
193         clear_bit(bd3out.stat, BSTALL); // clear stall bit
194         clear_bit(bd3out.stat, BC9);
195         clear_bit(bd3out.stat, BC8);
196         set_bit  (bd3out.stat, UOWN);   // SIE owns the buffer
197     #endif
198
199 #endif
200
201 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void usb_enable_module ()

After you've called usb_setup(), you can call usb_enable_module() whenever you're ready for USB negotiations to occur. Normally, this would need to occur relatively quickly after power-up if your PIC is powered by USB and it's purpose is to talk over USB. This is normally called from your main() routine once all other configuration is done.

Once the USB module has successfully negotiated a connection with the host, usb_device_configured_callback() will be called if you have requested this in your config.h file. This will indicate a successful connection. Because of the way USB works, there is no way to tell that it *hasn't* worked, except via a timer - if you haven't had a good connection in several seconds, you can assume it has failed (although this may just mean the user is hunting for a driver disk etc).

```
1033                            {
1034     uir = 0;
1035     set_bit(ucon, USBEN);   // enable USB serial interface engine (SIE)
1036     usb_state = st_DEFAULT;
1037 }
```

usb_state_type usb_get_state ()

Returns the USB state, either powered, default, address or connect. This is updated as the negotiation progresses. It may be useful to query this if, after a suitable time-out, the connection has not been made.

```
1039                                {
1040     return usb_state;
1041 }
```

void usb_handle_isr ()

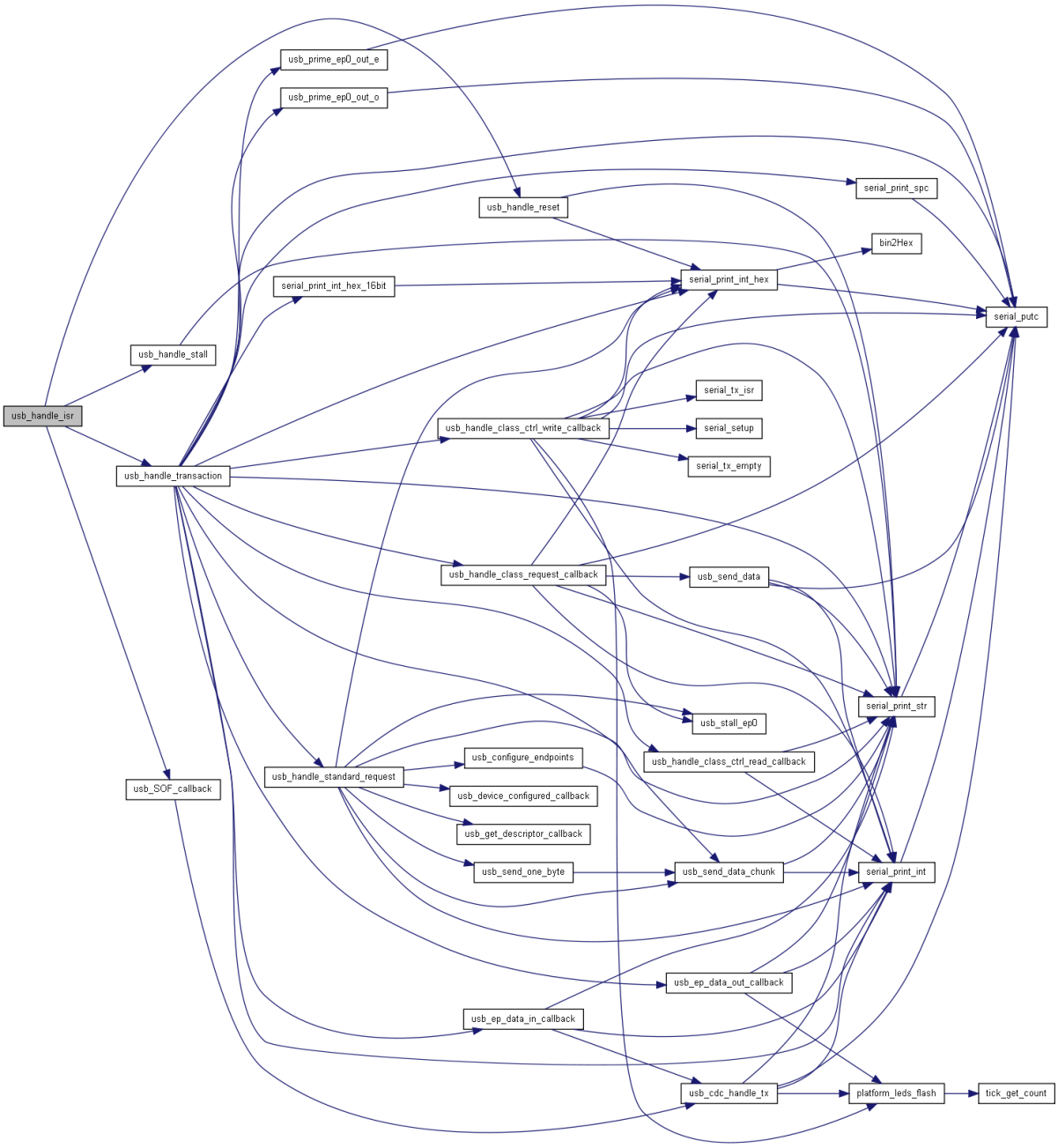
usb_handle_isr() should be inserted in your interrupt service routine. Alternatively, if you have reason not to want to do interrupt-driven USB, for example, a bootloader, you can poll this routine.

Make sure you call turn_usb_ints() and turn_global_ints_on() to ensure interrupts occur.

It will check for any of the USB interrupt flags and handle: USB transactions, USB reset, USB stall, USB Start Of Frame (including calling usb_SOF_callback() if configured in your config.h and most importantly USB transaction, which is where all the hard work is done.

```
928                         {
929 
930     while (test_bit(pir2, USBIF)) {
931 
932         while (test_bit(uir, TRNIF)) {
933             uns8 stat = ustat;
934             clear_bit(uir, TRNIF);
935             usb_handle_transaction(stat);
936         }
937 
938         if (test_bit(uir, URSTIF)) {
939             usb_handle_reset();
940             clear_bit(uir, URSTIF);
941         }
942 
943         if (test_bit(uir, STALLIF)) {
944             usb_handle_stall();
945             clear_bit(uir, STALLIF);
946         }
947         if (test_bit(uir, SOFIF)) {
948             #ifdef USB_CALLBACK_ON_SOF
949                 #ifdef ufrm
950                     usb_SOF_callback(ufrm);
951                 #else
952                     usb_SOF_callback(ufrmh << 8 | ufrml);
953                 #endif
954                 clear_bit(uir, SOFIF);
955             #endif
956         }
957 
958         clear_bit(pir2, USBIF);
959     }
960 }
```

Here is the call graph for this function:

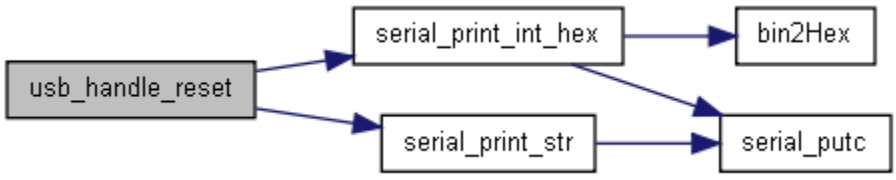


void usb_handle_reset ()

```
853                         {
854     usb_address = 0;
855     //uaddr = 0;
856 
857     control_mode = cm_IDLE;
858     usb_status   = us_IDLE;
859 
860     // clear fifo
861     clear_bit(uir, TRNIF);
862     clear_bit(uir, TRNIF);
863     clear_bit(uir, TRNIF);
```

```
864     clear_bit(uir, TRNIF);
865
866     // init buffers
867
868
869     // EP0 OUT
870     bd0out_e.count = USB_EP0_OUT_E_SIZE;    // 8 byte buffer
871     bd0out_e.addr = USB_EP0_OUT_E_ADDR;
872
873     clear_bit(bd0out_e.stat, DTS);  // turn on data togle sync TOGGLE
874     clear_bit(bd0out_e.stat, KEN);  // clear the keep bit
875     clear_bit(bd0out_e.stat, INCDIS);   // clear the increment disable
876     clear_bit  (bd0out_e.stat, DTSEN);  // !!!!!!
877     clear_bit(bd0out_e.stat, BSTALL);   // clear stall bit
878     clear_bit(bd0out_e.stat, BC9);
879     clear_bit(bd0out_e.stat, BC8);
880
881     set_bit  (bd0out_e.stat, UOWN); // SIE owns the buffer
882                                     // since we expect frist transaction to be SETUP
883     // EP0 OUT
884     bd0out_o.count = USB_EP0_OUT_O_SIZE;    // 8 byte buffer
885     bd0out_o.addr = USB_EP0_OUT_O_ADDR;
886
887     clear_bit(bd0out_o.stat, DTS);  // turn on data togle sync TOGGLE
888     clear_bit(bd0out_o.stat, KEN);  // clear the keep bit
889     clear_bit(bd0out_o.stat, INCDIS);   // clear the increment disable
890     clear_bit  (bd0out_o.stat, DTSEN);  // !!!!!!
891     clear_bit(bd0out_o.stat, BSTALL);   // clear stall bit
892     clear_bit(bd0out_o.stat, BC9);
893     clear_bit(bd0out_o.stat, BC8);
894
895     set_bit  (bd0out_o.stat, UOWN); // SIE owns the buffer
896                                     // since we expect frist transaction to be SETUP
897
898
899
900     // EP0 IN
901     bd0in.count = USB_EP0_IN_SIZE;  // 8 byte buffer
902     bd0in.addr = USB_EP0_IN_ADDR;
903     clear_bit(bd0in.stat, DTS); // turn on data togle sync TOGGLE
904     clear_bit(bd0in.stat, KEN); // clear the keep bit
905     clear_bit(bd0in.stat, INCDIS);  // clear the increment disable
906     clear_bit(bd0in.stat, BSTALL);  // clear stall bit
907     clear_bit(bd0in.stat, BC9);
908     clear_bit(bd0in.stat, BC8);
909
910     clear_bit(bd0in.stat, UOWN);    // uC owns the buffer
911
912     #ifdef USB_DEBUG
913         serial_print_str("\nR ");
914         serial_print_int_hex(uir);
915     #endif
916 }
```

Here is the call graph for this function:

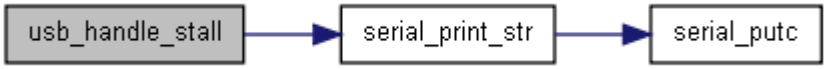


Here is the caller graph for this function:

void usb_handle_stall ()

```
918                             {
919     #ifdef USB_DEBUG
920         serial_print_str(" U:Stall ");
921     #endif
922     clear_bit(bd0in.stat, UOWN);
923     clear_bit(bd0in.stat, BSTALL);
924 }
```

Here is the call graph for this function:



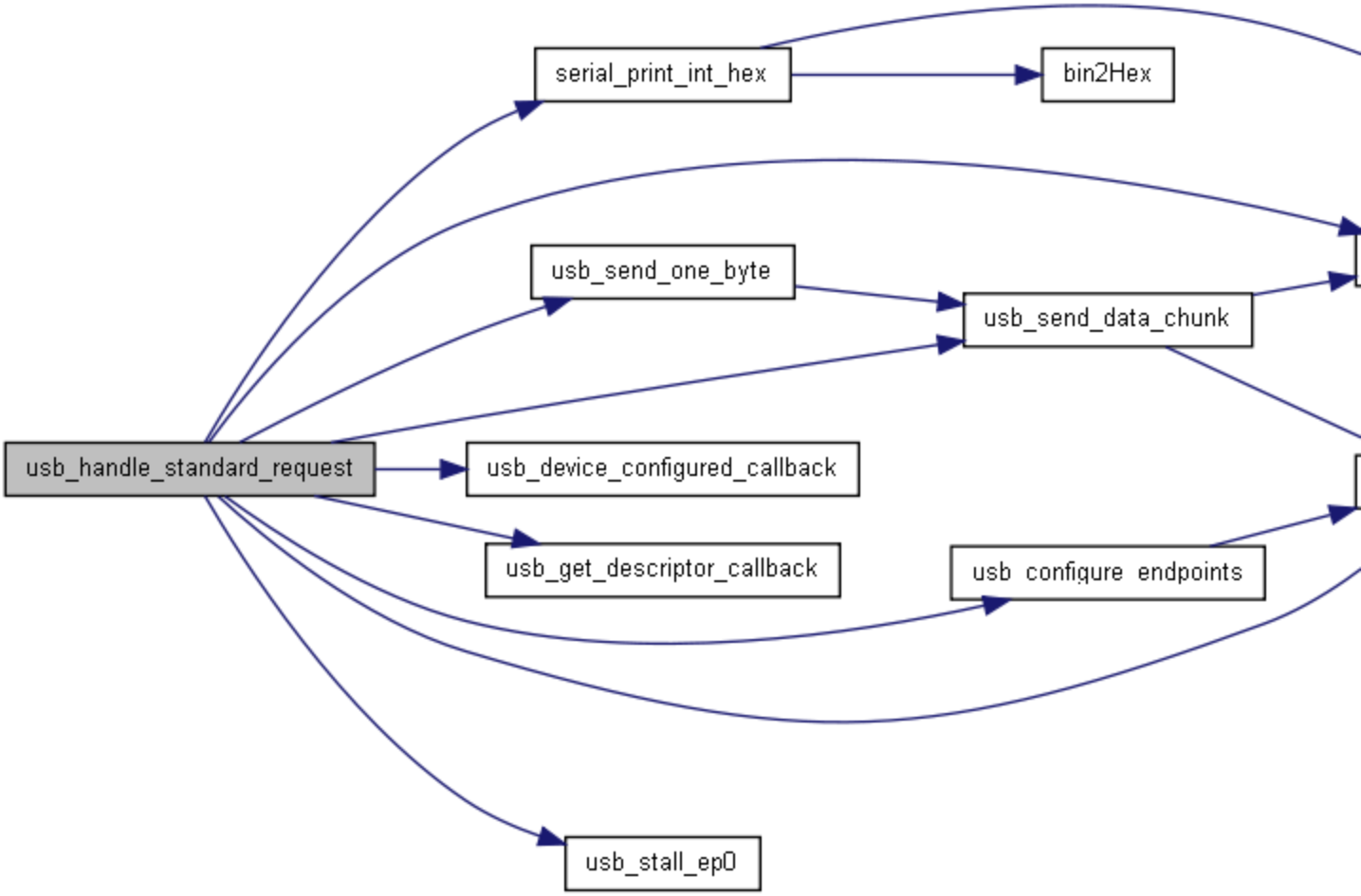
Here is the caller graph for this function:

void usb_handle_standard_request (setup_data_packet *sdp*)

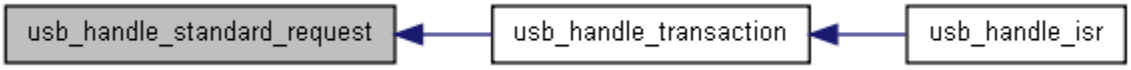
```
428                                                         {
429
430     switch (sdp.bRequest) {
431         case req_Get_Descriptor:
432             #ifdef USB_DEBUG
433                 serial_print_str(" GD: ");
434             #endif
435             uns8 descriptor_type = sdp.wValue >> 8; // high byte is descriptor
436             uns8 descriptor_num  = sdp.wValue & 0xff; // low byte is particular descriptor
437             #ifdef USB_DEBUG
438                 serial_print_int(descriptor_type);
439             #endif
440             usb_get_descriptor_callback(descriptor_type, descriptor_num, &delivery_ptr,
&delivery_bytes_to_send);
441             if (delivery_ptr != 0) {    // we've got something
442                 control_mode = cm_CTRL_READ_DATA_STAGE;
443                 delivery_bytes_max_send = sdp.wLength;  // maximum host wants
444                 delivery_bytes_sent = 0;    // clear our sent countr
445                 delivery_buffer_size = USB_EP0_IN_SIZE;
446                 delivery_bd = &bd0in;
447                 delivery_buffer = (uns8 *)USB_EP0_IN_ADDR;
448                 clear_bit(bd0in.stat, DTS); // ready to get toggled
449                 usb_send_data_chunk();
450             } else {
451                 #ifdef USB_DEBUG
452                     serial_print_str(" <stall> ");
453                 #endif
454                 usb_stall_ep0();
455             }
456
457             break;
458         case req_Set_Address:
459             usb_address = sdp.wValue & 0xff;
460             #ifdef USB_DEBUG
461                 serial_print_str(" SA:");
462                 serial_print_int_hex(usb_address);
463             #endif
464             usb_status = us_SET_ADDRESS;
465
466             // Send a status ack - when we confirm that, THEN change address
```

```
467             usb_send_status_ack();
468             control_mode = cm_CTRL_WRITE_SENDING_STATUS;
469             break;
470         case req_Set_Configuration:
471             #ifdef USB_DEBUG
472                 serial_print_str(" SC: ");
473             #endif
474 
475             //sdp.wValue & 0xff; // MORE WORK see p136
476             usb_configure_endpoints();
477             // and do a call back to let the app know we're ready
478             #ifdef USB_CALLBACK_ON_DEVICE_CONFIGURED
479                 usb_device_configured_callback();
480             #endif
481 
482             //set_bit(ucon, PPBRST);    // reset ping pong buffers to even
483             //clear_bit(ucon, PPBRST);
484 
485             usb_send_status_ack();
486             control_mode = cm_CTRL_WRITE_SENDING_STATUS;
487 
488             usb_state = st_CONFIGURED;
489 
490             // reset ping pong buffers
491 
492 
493             // device is up - so now get the endpoints happy
494             break;
495         case req_Get_Interface:
496                 #ifdef USB_DEBUG
497                     serial_print_str(" GI ");
498                 #endif
499                 control_mode = cm_CTRL_READ_DATA_STAGE;
500                 usb_send_one_byte(1);
501         case req_Get_Status:
502                 // this is wrong - needs to send two bytes !!
503                 #ifdef USB_DEBUG
504                     serial_print_str(" GS ");
505                 #endif
506 
507                 #ifdef USB_SELF_POWERED
508                     usb_send_one_byte(1);
509                 #else
510                     usb_send_one_byte(0);   // bus powered
511                 #endif
512                 break;
513         default:
514             #ifdef USB_DEBUG
515                 serial_print_str(" ??SR ");
516                 serial_print_int(sdp.bRequest);
517             #endif
518             break;
519 
520     }
521 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_handle_transaction (uns8 *stat*)

!usb_stall_on_in();

```
524                                     {
525
526 uns8 end_point, pid;
527 bit even;
528 uns8 count;
529
530     even = !test_bit(stat, PPBI);
531     end_point = stat >> 3;
532
533     if (test_bit(stat, DIR)) {  // IN
534         #ifdef USB_DEBUG
535             serial_print_str("\nDI[");
536                 for (count=0; count < 8; count++) {
537                     serial_print_int_hex(buffer_0_in[count]);
538                     serial_putc(']');
539                 }
```

```
540         #endif
541         pid = (bd0in.stat >> 2) & 0x0f; // mask out pid
542
543     } else {    // OUT
544         if (even) {
545             pid = (bd0out_e.stat >> 2) & 0x0f;  // mask out pid
546         } else {
547             pid = (bd0out_o.stat >> 2) & 0x0f;  // mask out pid
548         }
549         #ifdef USB_DEBUG
550             serial_print_str("\nDO ");
551             if (even) {
552                 for (count=0; count < 8; count++) {
553                     serial_print_int_hex(buffer_0_out_e[count]);
554                     serial_putc(' ');
555                 }
556             } else {
557                 for (count=0; count < 8; count++) {
558                     serial_print_int_hex(buffer_0_out_o[count]);
559                     serial_putc(' ');
560                 }
561             }
562
563         #endif
564
565     }
566     #ifdef USB_DEBUG
567         if (end_point != 0) {
568             serial_putc('E');
569             serial_print_int_hex(end_point);
570             serial_print_spc();
571         }
572     #endif
573
574
575
576     if (end_point == 0) {
577         if (!(test_bit(stat, DIR))) {
578             #ifdef USB_DEBUG_HIGH
579             if (even) {
580                 serial_print_str(" (e) ");
581             } else {
582                 serial_print_str(" (o) ");
583             }
584             #endif
585         }
586         #ifdef USB  _DEBUG_HIGH
587             serial_print_str(" bytes: ");
588             if (test bit(stat, DIR)) {
589                 serial_print_int(bd0in.count);
590             } else {
591                 if (even) {
592                     serial_print_int(bd0out_e.count);
593                 } else {
594                     serial_print_int(bd0out_o.count);
595                 }
596             }
597         #endif
598         if (pid == pid_SETUP) {
599
600             #ifdef USB DEBUG
601                 serial_print_str(" Setup ");
602             #endif
603             if (even) {
604             //  for (count=0; count < 8; count++) {
605             //      serial_print_int_hex(buffer_0_out_e[count]);
606             //      serial_putc(' ');
607             //  }
608                 memcpy(/*dst*/ (void*)&usb_sdp, /*src*/ (void *)&buffer_0_out_e, 8);
609                 usb_prime_ep0_out_e();
610             } else {
```

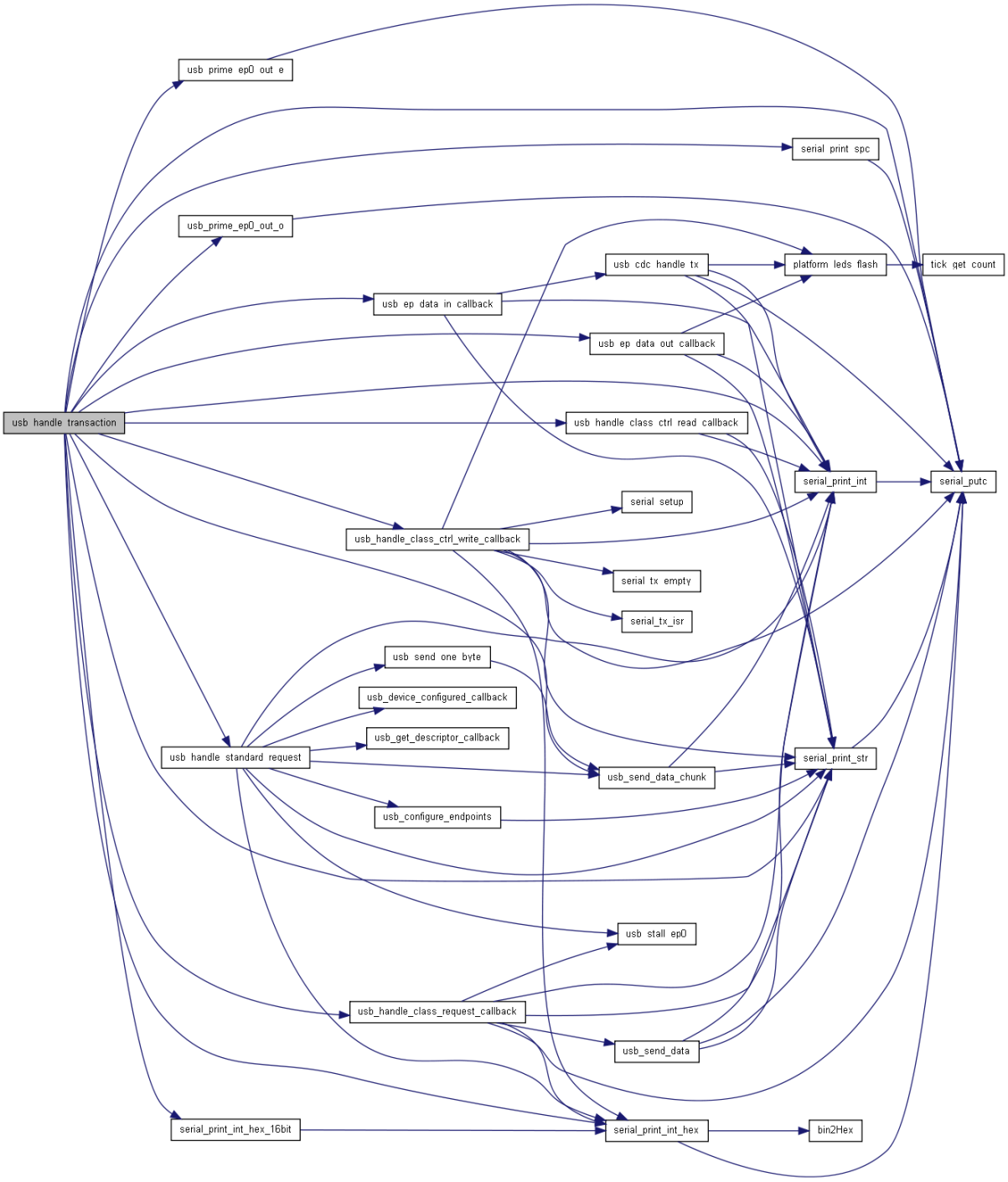
```
611             //  for (count=0; count < 8; count++) {
612             //      serial_print_int_hex(buffer_0_out_o[count]);
613             //      serial_putc(' ');
614             //  }
615                 memcpy(/*dst*/ (void*)&usb_sdp, /*src*/ (void *)&buffer_0_out_o, 8);
616                 usb_prime_ep0_out_o();
617             }
618
619             // We may have issued a stall on previous transfer, so need to grab the buffer
back for
620             // our use. Presumably this happens so you can stall continuously. However, we
really want
621             // it back.
622
623             clear_bit  (bd0in.stat, UOWN);  // SIE owns the buffer
624
625             #ifdef USB_DEBUG
626
627             serial_print_str("RT= ");
628             serial_print_int_hex(usb_sdp.bmRequestType);
629             serial_print_str(" rq=");
630             serial_print_int_hex(usb_sdp.bRequest);
631             serial_print_str(" Va=");
632             serial_print_int_hex_16bit(usb_sdp.wValue);
633             serial_print_str(" In=");
634             serial_print_int_hex_16bit(usb_sdp.wIndex);
635             serial_print_str(" Ln=");
636             serial_print_int_hex_16bit(usb_sdp.wLength);
637
638
639
640                 serial_putc(' ');
641                 if (test_bit(usb_sdp.bmRequestType, DATA_STAGE_DIR)) {
642                     serial_print_str(" Data=IN");
643                 } else {
644                     serial_print_str(" Data=OUT/NO");
645                 }
646             #endif
647             //serial_print_str(" len: ");
648             //serial_print_int(usb_sdp.wLength);
649             // Is it a standard request?
650             //serial_putc('\\');
651
652             if (!test_bit(usb_sdp.bmRequestType, REQUEST_TYPE1) && // std request
653                 !test_bit(usb_sdp.bmRequestType, REQUEST_TYPE0)) {
654                 //serial_print_str(" Std req ");
655                 if ((usb_sdp.bmRequestType & 0b00011111) == 0) {
656                     //serial_print_str(" 2dev ");
657                 } else if ((usb_sdp.bmRequestType & 0b00011111) == 1) {
658                     //serial_print_str(" 2int ");
659                 } else if ((usb_sdp.bmRequestType & 0b00011111) == 0b00011) {
660                     //serial_print_str(" 2oth ");
661                 }
662                 #ifdef USB_DEBUG
663                     serial_print_str(" std ");
664                 #endif
665                 usb_handle_standard_request(usb_sdp);
666             } else if (!test_bit(usb_sdp.bmRequestType, REQUEST_TYPE1) && // class request
667                     test_bit(usb_sdp.bmRequestType, REQUEST_TYPE0)) {
668                     #ifdef USB_DEBUG
669                         serial_print_str(" class ");
670                     #endif
671                 #ifdef USB_CALLBACK_ON_CLASS_CTRL
672                     usb_handle_class_request_callback(usb_sdp);
673                 #endif
674             } else {
675                 //serial_print_str(" req no ");
676                 //serial_print_int(usb_sdp.bRequest);
677                 // serial_print_spc();
678                 #ifdef USB_DEBUG
679                     serial_print_str(" ??req t=");
```

```
680                     serial_print_int(usb_sdp.bmRequestType);
681                     serial_putc(' ');
682                 #endif
683             }
684             clear_bit(ucon, PKTDIS);
685 
686         } else if (pid == pid_IN) {
687             #ifdef USB_DEBUG
688                 serial_print_str(" IN ");
689             #endif
690             if (control_mode == cm_CTRL_READ_DATA_STAGE) {
691                 // it's ours, so send next chunk
692                 usb_send_data_chunk();
693             } else if (control_mode == cm_CTRL_WRITE_SENDING_STATUS) {
694                 //serial_print_str("std ");
695                 control_mode = cm_IDLE;
696                 if (usb_status == us_SET_ADDRESS) {
697                     #ifdef USB_DEBUG
698                         serial_print_str(" addr to ");
699                         serial_print_int(uaddr);
700                     #endif
701                     usb_state = st_ADDRESS;
702                     uaddr = usb_address;
703                     usb_status = us_IDLE;
704                 }
705                 #ifdef USB_DEBUG
706                     serial_print_str(" ----\n");
707                 #endif
708             } else if (control_mode == cm_CTRL_READ_AWAITING_STATUS) {
709                 // Must have been last IN of the read, so still waiting for status
710                 #ifdef USB_DEBUG
711                     serial_print_str(" last read, waiting status");
712                 #endif
714                 //usb_prime_ep0_out();  // !! NEW
715                 //???
716                 nop(); // boostc bug
717             } else if (control_mode == cm_CTRL_READ_DATA_STAGE_CLASS) {
718                 #ifdef USB_DEBUG
719                     serial_print_str(" ctrl read data stage class - more to come? ");
720                 #endif
721                 // Must be more to come
722                 #ifdef USB_CALLBACK_ON_CLASS_CTRL
723                     usb_handle_class_ctrl_read_callback();
724                 #else
725                     nop(); // otherwise boostc bug
726                 #endif
727             } else {
728                 #ifdef USB_DEBUG
729                     serial_print_str(" ?? cm=");
730                     serial_print_int((uns8)control_mode);
731                     serial_print_spc();
732                 #else
733                     nop(); // boostc bug
734                 #endif
735 
736             }
737         } else if (pid == pid_ACK) {
738             #ifdef USB_DEBUG
739                 serial_print_str("****A\n");
740             #endif
741             if (control_mode == cm_CTRL_READ_DATA_STAGE) {
742                 #ifdef USB_DEBUG
743                     serial_print_str(" &2 ");
744                 #endif
745                 usb_send_data_chunk();
746             } else if (control_mode == cm_CTRL_READ_DATA_STAGE_CLASS) {
747                 #ifdef USB_CALLBACK_ON_CLASS_CTRL
748                     usb_handle_class_ctrl_read_callback();
749                 #else
750                     nop(); // boostc bug
751                 #endif
```

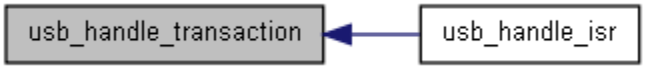
```
752             } else if (control_mode == cm_CTRL_WRITE_SENDING_STATUS) {
753                 #ifdef USB_DEBUG
754                     serial_print_str(" st sent ");
755                 #endif
756                 control_mode = cm_IDLE;
757             } else if (control_mode == cm_CTRL_READ_AWAITING_STATUS) {
758                 #ifdef USB_DEBUG
759                     serial_print_str(" now what? ");
760                 #endif
761                 control_mode = cm_IDLE;
762             }
763 
764 
765         } else if (pid == pid_OUT) {
766             uns8 count;
767             uns8 *buffer;
768 
769             if (even) {
770                 count = bd0out_e.count;
771                 buffer = &buffer_0_out_e;
772                 usb_prime_ep0_out_e();
773             } else {
774                 count = bd0out_o.count;
775                 buffer = &buffer_0_out_o;
776                 usb_prime_ep0_out_o();
777             }
778 
779             // We've done an out
780             //serial_print_str(" OUT ");
781             if (control_mode == cm_CTRL_READ_AWAITING_STATUS) {
782                 #ifdef USB_DEBUG
783                     serial_print_str(" ----\n");
784                 #endif
785                 control_mode = cm_IDLE;
786             } else if (control_mode == cm_CTRL_WRITE_DATA_STAGE_CLASS) {
787                 #ifdef USB_CALLBACK_ON_CLASS_CTRL
788                     usb_handle_class_ctrl_write_callback(buffer, count);
789                     // !! should include bc bits here for total count
790                     // this only works for 8 bit data packets
791                 #else
792                     nop(); // boostc bug
793                 #endif
794             } else if (control_mode == cm_CTRL_READ_DATA_STAGE) {
795                 //serial_print_str(" Status early ");
796                 control_mode = cm_IDLE;
797             } else {
798 
799                 #ifdef USB_DEBUG
800                     serial_print_str("??unk pid OUT ");
801                     serial_print_int((uns8)control_mode);
802                 #endif
803             }
804 
805         } else {
806             #ifdef USB_DEBUG
807                 serial_print_str(" UKPID = ");
808                 serial_print_int(pid);
809             #endif
810         }
811 
812     } else {    // Not endpoint 0
813         buffer_descriptor *bd;
814         if (test_bit(stat, DIR)) {
815             // in
816             #ifdef USB_EP_DATA_CALLBACK
817                 bd = ep_in_bd_location[end_point];
818                 usb_ep_data_in_callback(end_point, bd->count);
819             #else
820                 nop();
821             #endif
822         } else { // out
```

```
823
824             bd = ep_out_bd_location[end_point];
825             // issue callback
826             #ifdef USB_EP_DATA_CALLBACK
827                 usb_ep_data_out_callback(end_point, ep_out_buffer_location[end_point],
828                                          bd->count);
829             #endif
830             // re-prime endpoint
831             bd->count = ep_out_buffer_size[end_point];
832             bd->addr  = (uns16)ep_out_buffer_location[end_point];
833
834             // Address shouldn't change, so don't need to update it
835
836             clear_bit(bd->stat, DTS);   // turn on data togle sync TOGGLE
837             clear_bit(bd->stat, KEN);   // clear the keep bit
838             clear_bit(bd->stat, INCDIS);    // clear the increment disable
839             clear_bit(bd->stat, DTSEN);
840             clear_bit(bd->stat, BSTALL);    // clear stall bit
841             clear_bit(bd->stat, BC9);
842             clear_bit(bd->stat, BC8);
843             set_bit  (bd->stat, UOWN);  // SIE owns the buffer
844
845
846         }
847     }
848
849
850
851 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

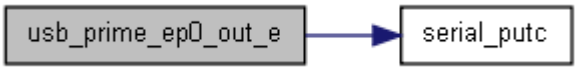


void usb_prime_ep0_out_e ()

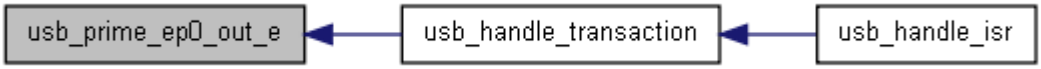
! clear

```
384                             {
385
386     bd0out_e.count = USB_EP0_OUT_E_SIZE;
387     bd0out_e.addr = USB_EP0_OUT_E_ADDR;
388     //changed from clear to set
389     clear_bit(bd0out_e.stat, DTS);  // turn on data togle sync TOGGLE
390     clear_bit(bd0out_e.stat, KEN);  // clear the keep bit
391     clear_bit(bd0out_e.stat, INCDIS);   // clear the increment disable
393     clear_bit(bd0out_e.stat, DTSEN);
394     clear_bit(bd0out_e.stat, BSTALL);   // clear stall bit
395     clear_bit(bd0out_e.stat, BC9);
396     clear_bit(bd0out_e.stat, BC8);
397
398     set_bit  (bd0out_e.stat, UOWN); // SIE owns the buffer
399
400     #ifdef USB_DEBUG
401         serial_putc('P');
402         serial_putc('E');
403     #endif
404 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

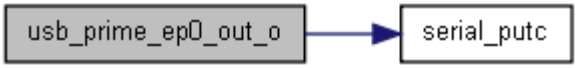


void usb_prime_ep0_out_o ()

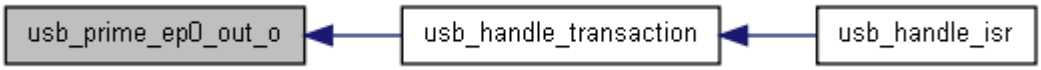
! clear

```
406                             {
407
408     bd0out_o.count = USB_EP0_OUT_O_SIZE;
409     bd0out_o.addr = USB_EP0_OUT_O_ADDR;
410     //changed from clear to set
411     clear_bit(bd0out_o.stat, DTS);  // turn on data togle sync TOGGLE
412     clear_bit(bd0out_o.stat, KEN);  // clear the keep bit
413     clear_bit(bd0out_o.stat, INCDIS);   // clear the increment disable
415     clear_bit(bd0out_o.stat, DTSEN);
416     clear_bit(bd0out_o.stat, BSTALL);   // clear stall bit
417     clear_bit(bd0out_o.stat, BC9);
418     clear_bit(bd0out_o.stat, BC8);
419
420     set_bit  (bd0out_o.stat, UOWN); // SIE owns the buffer
421
422     #ifdef USB_DEBUG
423         serial_putc('P');
424         serial_putc('O');
425     #endif
426 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_send_data (uns8 *ep*, uns8 * *data*, uns8 *send_count*, bit *first*)

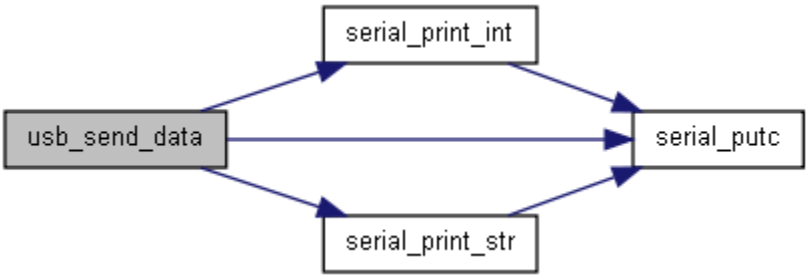
Use this routine to send data across the USB pipe.

Parameters:

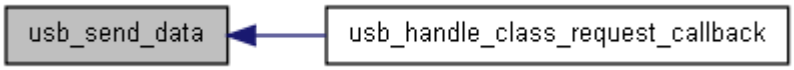
ep Endpoint that the data should be sent from
data pointer to the data
send_count the number of bytes to send
first True if this is the first in a series of sends. Generally, this can be set to False, since it will automatically be set to the right value on endpoint creation. However, in the case of control transfers, the data stage needs to have the first parameter set to True to ensure the DTS bit is set correctly.

```
211                                                                 {
212 uns8 count;
213 buffer_descriptor *bd;
214 uns8 *buffer;
215
216     // this is going to be an IN transaction
217     #ifdef USB_DEBUG
218         serial_print_str("Send:EP");
219         serial_print_int(ep);
220         serial_putc(' ');
221     #endif
222     // need to grab buffer descriptor
223     buffer = ep_in_buffer_location[ep];
224
225     bd = ep_in_bd_location[ep];
226
227     if (test_bit(bd->stat, UOWN)) {
228     #ifdef USB_DEBUG
229         serial_print_str(" !Adon't own it! ");
230     #endif
231         return;
232     }
233
234     count = 0;
235     while ((count < send_count)) {
236         buffer[count] = data[count];
237         count++;
238     }
239
240
241     bd->count = count;
242     bd->addr = (uns16)buffer;
243     if (first) {
244         clear_bit(bd->stat, DTS);   // So when it flips, will end up set
245     }
246
247     toggle_bit(bd->stat, DTS);  // flip the DTS bit
248     clear_bit(bd->stat, KEN);   // clear the keep bit
249     clear_bit(bd->stat, INCDIS);    // clear the increment disable
250     set_bit   (bd->stat, DTSEN);
251     clear_bit(bd->stat, BSTALL);    // clear stall bit
252     clear_bit(bd->stat, BC9);
253     clear_bit(bd->stat, BC8);
254
255     set_bit   (bd->stat, UOWN);  // SIE owns the buffer
256 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

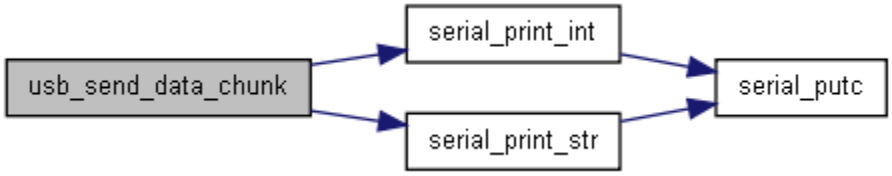


void usb_send_data_chunk ()

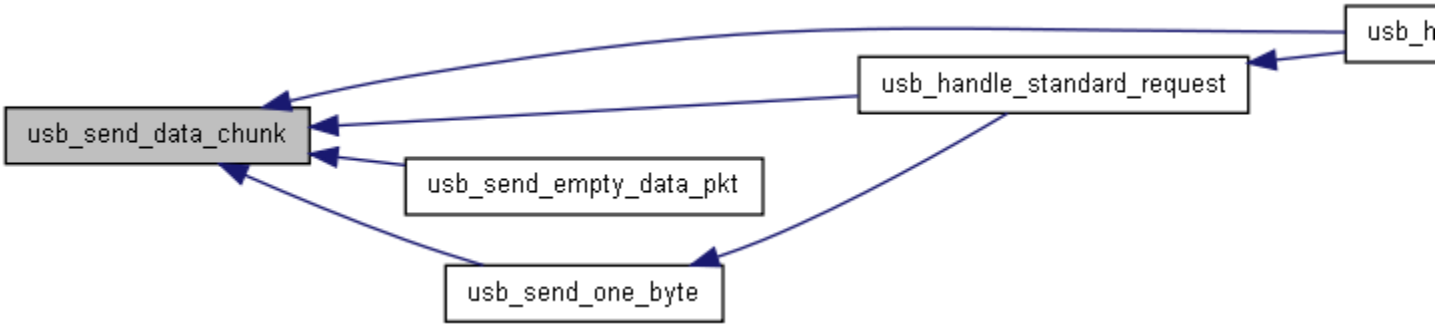
```
260                             {
261
262 uns8 count;
263 uns8 now = 0;
264
265     if (test bit(bd0in.stat, UOWN)) {
266     #ifdef USB_DEBUG
267         serial print str(" !Bdon't own it! ");
268     #endif
269         return;
270     }
271
272     count = 0;
273     while ((count < delivery buffer size) &&
274            (delivery bytes sent < delivery bytes to send) &&
275            (delivery bytes sent < delivery bytes max send)) {
276         delivery buffer[count] = *delivery ptr;
277         delivery ptr++;
278         delivery bytes sent++;
279         count++;
280         now++;
281     }
282     #ifdef USB DEBUG
283         serial print str(" wrote ");
284         serial print int(now);
285     #endif
286     if ((count < delivery buffer size) &&
287         ((delivery bytes sent == delivery bytes max send) ||
288         (delivery bytes sent == delivery bytes to send)))  {
289         #ifdef USB_DEBUG_x
290         serial print str(" c=");
291         serial print int(count);
292         serial print str(" dbsz=");
293         serial print int(delivery buffer size);
294
295         serial print str(" dbs=");
296         serial print int(delivery bytes sent);
297         serial print str(" dbms=");
298         serial print int(delivery bytes max send);
299         serial print str(" dbts=");
300         serial print int(delivery bytes to send);
301         #endif
302         if (control mode != cm CTRL WRITE SENDING STATUS) {
303             control mode = cm CTRL READ AWAITING STATUS;    // we're done with data stage
```

```
304             #ifdef USB_DEBUG_HIGH
305                 serial_print_str(" st2READWAITSTAT ");
306             #endif
307         }
308     }
309 
310 
311     bd0in.count = count;
312     bd0in.addr = (uns16)&buffer_0_in;
313     //serial_print_str("Buffer=");
314     //serial_print_int_hex_16bit(bd0in.addr);
315     toggle_bit(bd0in.stat, DTS);
316     //these not required for 14k50, but random things seem to happen if you don't clear them
317     clear_bit(bd0in.stat, KEN);     // clear the keep bit
318     clear_bit(bd0in.stat, INCDIS);  // clear the increment disable
319     set_bit  (bd0in.stat, DTSEN);
320     clear_bit(bd0in.stat, BSTALL);  // clear stall bit
321     clear_bit(bd0in.stat, BC9);
322     clear_bit(bd0in.stat, BC8);
323 
324     set_bit  (bd0in.stat, UOWN);    // SIE owns the buffer
325     /*
326     delivery_bd->count = count;
327     delivery_bd->addr = (uns16)delivery_buffer;
328     toggle_bit(delivery_bd->stat, DTS);
329     clear_bit(delivery_bd->stat, KEN);  // clear the keep bit
330     clear_bit(delivery_bd->stat, INCDIS);   // clear the increment disable
331     set_bit  (delivery_bd->stat, DTSEN);
332     clear_bit(delivery_bd->stat, BSTALL);   // clear stall bit
333     clear_bit(delivery_bd->stat, BC9);
334     clear_bit(delivery_bd->stat, BC8);
335 
336     set_bit  (delivery_bd->stat, UOWN); // SIE owns the buffer
337     */
338 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



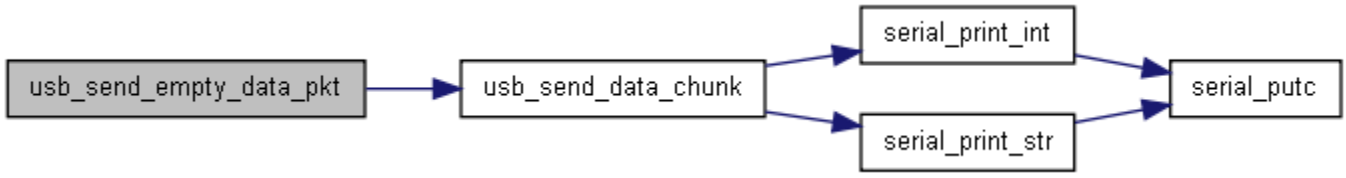
void usb_send_empty_data_pkt ()

Use this routine to send an data across the USB pipe on endpoint 0. This is the equivalent of sending a status acknowledge.

```
341                                 {
```

```
342     delivery_buffer_size = USB_EP0_IN_SIZE;
343     delivery_bd = &bd0in;
344     delivery_buffer = &buffer_0_in;
345     delivery_bytes_sent = 0;
346     delivery_bytes_to_send = 0;
347     delivery_bytes_max_send = 0;
348     delivery_ptr = (uns8 *) 0;
349     clear_bit(bd0in.stat, DTS); // ready to get toggled
350     usb_send_data_chunk();
351 }
```

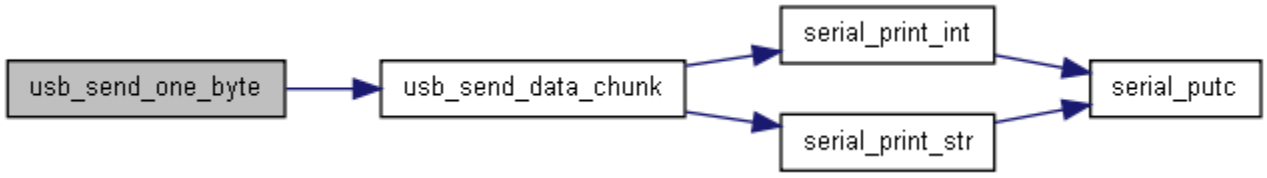
Here is the call graph for this function:



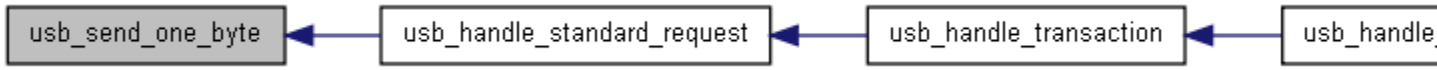
void usb_send_one_byte (uns8 *data*)

```
370                                     {
371     delivery_buffer_size = USB_EP0_IN_SIZE;
372     delivery_bytes_sent = 0;
373     delivery_bytes_to_send = 1;
374     delivery_bytes_max_send = 0;
375     buffer_byte = data;
376     delivery_ptr = (uns8 *) &buffer_byte;
377     delivery_bd = &bd0in;
378     delivery_buffer = &buffer_0_in;
379
380     clear_bit(bd0in.stat, DTS); // ready to get toggled
381     usb_send_data_chunk();
382 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_setup ()

usb_setup() configures the PIC USB hardware ready for use and prepares the internal data structures used to keep track of where the endpoint buffers are.

After calling usb_setup(), you are ready to call usb_enable_module() to actually start USB negotiations. Ensure that you have usb_handle_isr() in your interrupt service routine.

```
973                 {
974
975     usb_state = st_POWERED;
976
```

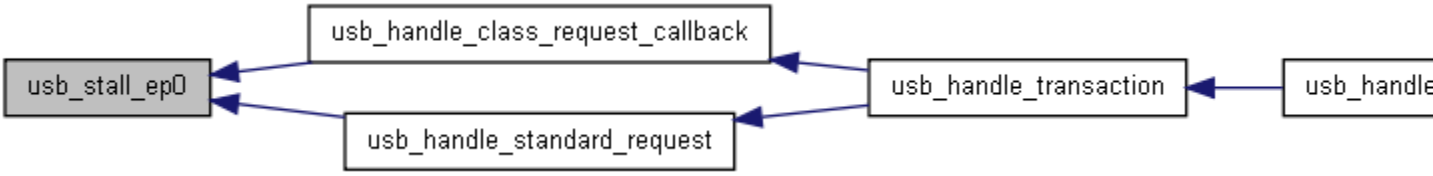
```
977     // init hardware
978     #ifdef UTRDIS
979         clear bit(ucfg, UTRDIS);    // enable internal tranceiver
980     #endif
981     set_bit  (ucfg, FSEN);  // clear for low speed, set for high speed
982     set_bit  (ucfg, UPUEN); // enable on-chip pull-ups
983
984     clear_bit(ucfg, PPB1);  // double buffering for EP0 OUT
985     set_bit(ucfg, PPB0);
986
987     // if using ping pong buffers, need to do this:
988     set_bit(ucon, PPBRST);  // reset ping pong buffers to even
989     clear_bit(ucon, PPBRST);
990
991     // init endpoint 0
992
993     set_bit(uep0,   EPHSHK);    // EP0 handshaking on
994     set_bit(uep0,   EPOUTEN);   // EP0 OUT enable
995     set_bit(uep0,   EPINEN);    // EP0 IN enable
996     clear_bit(uep0, EPCONDIS);  // EP0 control transfers on (and IN and OUT)
997
998     // init interrupts
999     // Config buffer descriptor table
1000
1001     ep_out_bd_location[0] = &bd0out_e;
1002     #if USB_HIGHEST_EP >= 1
1003         ep_out_bd_location[1] = &bd1out;
1004     #endif
1005     #if USB_HIGHEST_EP >= 2
1006         ep_out_bd_location[2] = &bd2out;
1007     #endif
1008     #if USB_HIGHEST_EP >= 3
1009         ep_out_bd_location[3] = &bd3out;
1010     #endif
1011     #if USB_HIGHEST_EP >= 4
1012         ep_out_bd_location[4] = &bd4out;
1013     #endif
1014
1015     ep_in_bd_location[0] = &bd0in;
1016     #if USB_HIGHEST_EP >= 1
1017         ep_in_bd_location[1] = &bd1in;
1018     #endif
1019     #if USB_HIGHEST_EP >= 2
1020         ep_in_bd_location[2] = &bd2in;
1021     #endif
1022     #if USB_HIGHEST_EP >= 3
1023         ep_in_bd_location[3] = &bd3in;
1024     #endif
1025     #if USB_HIGHEST_EP >= 4
1026         ep_in_bd_location[4] = &bd4in;
1027     #endif
1028
1029
1030 }
```

void usb_stall_ep0 ()

Use this routine to send a stall on the control transfer endpoint - usually used to indicate that the requested function is not available.

```
203                     {
204     set_bit(bd0in.stat, BSTALL);    // stall
205     set_bit(bd0in.stat, UOWN);  // SIE owns the buffer
206 //  set_bit(bd0out.stat, BSTALL);   // stall
207 //  set_bit(bd0out.stat, UOWN); // SIE owns the buffer
208     // ??set_bit(uep0,EPSTALL);
209 }
```

Here is the caller graph for this function:



void usb_stall_on_in ()

```
353                         {
354
355     clear_bit(bd0in.stat, DTS); // ready to get toggled
356
357     clear_bit(bd0in.stat, KEN); // clear the keep bit
358     clear_bit(bd0in.stat, INCDIS);  // clear the increment disable
359     clear_bit(bd0in.stat, DTSEN);
360     set_bit(bd0in.stat, BSTALL);    // clear stall bit
361     clear_bit(bd0in.stat, BC9);
362     clear_bit(bd0in.stat, BC8);
363
364     set_bit  (bd0in.stat, UOWN);    // SIE owns the buffer
365 }
```

Variable Documentation

uns8 buffer_byte

control_mode_type control_mode

Store the control mode state

buffer_descriptor* delivery_bd

uns8* delivery_buffer

uns8 delivery_buffer_size

uns16 delivery_bytes_max_send

uns16 delivery_bytes_sent

uns16 delivery_bytes_to_send

uns8* delivery_ptr

uns8 usb_address

Store the usb address

setup_data_packet usb_sdp

Store the last setup data packet

usb_state_type usb_state = st_POWERED

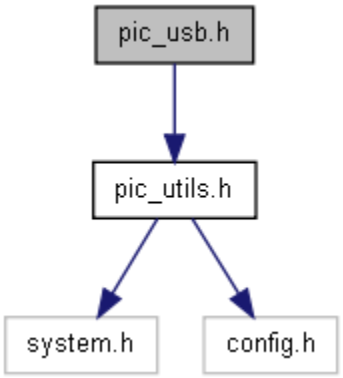
Store the current USB device state

usb_status_type usb_status

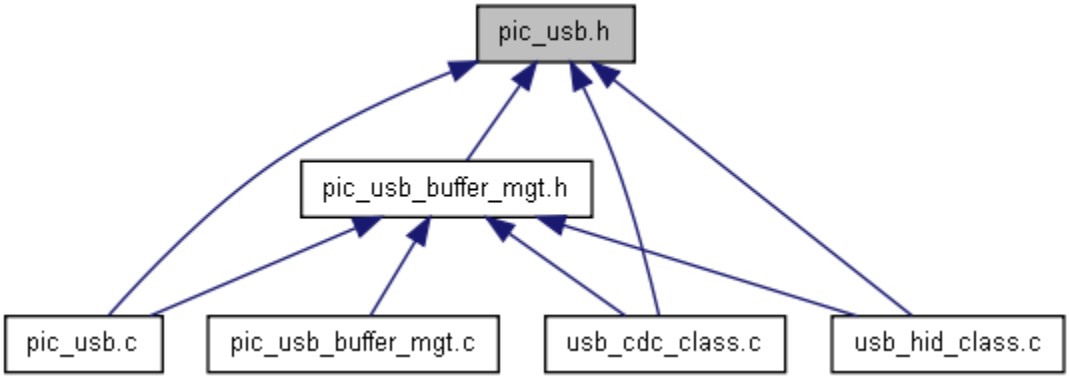
---

pic_usb.h File Reference

Pic USB routines.
Include dependency graph for pic_usb.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct buffer_descriptor
- struct CDC_ACM_functional_descriptor
- struct CDC_call_mgt_functional_descriptor
- struct CDC_header_functional_descriptor
- struct CDC_union_functional_descriptor
- struct configuration_descriptor
- struct device_descriptor
- struct endpoint_descriptor
- struct hid_descriptor

- struct interface_descriptor
- struct setup_data_packet

Defines

- #define BC8 0
- #define BC9 1
- #define BSTALL 2
- #define DATA_STAGE_DIR 7
- #define dt_CONFIGURATION 0x02
- #define dt_CS_INTERFACE 0x24
- #define dt_DEBUG 0x0a
- #define dt_DEVICE 0x01
- #define dt_DEVICE_QUALIFIER 0x06
- #define dt_ENDPOINT 0x05
- #define dt_HID 0x21
- #define dt_HID_REPORT 0x22
- #define dt_INTERFACE 0x04
- #define dt_INTERFACE_ASSOC 0x0b
- #define dt_INTERFACE_POWER 0x08
- #define dt_OTG 0x09
- #define dt_OTHER_SPEED_CONFIG 0x07
- #define dt_STRING 0x03
- #define DTS 6
- #define DTSEN 3
- #define INCDIS 4
- #define KEN 5
- #define PID0 2
- #define PID1 3
- #define PID2 4
- #define PID3 5
- #define pid_ACK 0b00000010
- #define pid_DATA0 0b00000011
- #define pid_DATA1 0b00001011
- #define pid_DATA2 0b00000111
- #define pid_IN 0b00001001
- #define pid_MDATA 0b00001111
- #define pid_NAK 0b00001010
- #define pid_NYET 0b00000110
- #define pid_OUT 0b00000001
- #define pid_SETUP 0b00001101
- #define pid_SOF 0b00000101
- #define pid_STALL 0b00001110
- #define req_Clear_Feature 0x01
- #define req_Get_Configuration 0x08
- #define req_Get_Descriptor 0x06
- #define req_Get_Interface 0x0a
- #define req_Get_Status 0x00
- #define req_Set_Address 0x05
- #define req_Set_Configuration 0x09
- #define req_Set_Descriptor 0x07
- #define req_Set_Feature 0x03
- #define req_Set_Interface 0x0b

- #define req_Synch_Frame 0x0c
- #define REQUEST_TYPE0 5
- #define REQUEST_TYPE1 6
- #define UOWN 7
- #define usb_send_status_ack() usb_send_empty_data_pkt()

Enumerations

- enum control_mode_type { cm_IDLE, cm_CTRL_WRITE_DATA_STAGE, cm_CTRL_WRITE_DATA_STAGE_CLASS, cm_CTRL_READ_DATA_STAGE, cm_CTRL_READ_DATA_STAGE_CLASS, cm_CTRL_READ_AWAITING_STATUS, cm_CTRL_WRITE_SENDING_STATUS }
- enum usb_state_type { st_POWERED, st_DEFAULT, st_ADDRESS, st_CONFIGURED }
- enum usb_status_type { us_IDLE, us_SET_ADDRESS }

Functions

- void turn_usb_ints_on ()
- *Turn on USB interrupts.* void usb_device_configured_callback ()
- *Callback routine triggered when a successful intial USB negotiation has completed.* void usb_enable_module ()
- *Enables the USB hardware and starts USB negotiations.* void usb_ep_data_in_callback (uns8 end_point, uns16 byte_count)
- *Callback routine triggered when data has been sent to the host.* void usb_ep_data_out_callback (uns8 end_point, uns8 *buffer_location, uns16 byte_count)
- *Callback routine triggered when data has been sent to the device.* void usb_get_descriptor_callback (uns8 descriptor_type, uns8 descriptor_num, uns8 **rtn_descriptor_ptr, uns16 *rtn_descriptor_size)
- *Callback routine triggered when the descriptor is requested by the host.* usb_state_type usb_get_state ()
- *Query the current state of the USB connection.* void usb_handle_class_ctrl_read_callback ()
- *Callback routine for a class control read.* void usb_handle_class_ctrl_write_callback (uns8 *data, uns16 count)
- *Callback routine for a class control write.* void usb_handle_class_request_callback (setup_data_packet sdp)
- *Callback routine for a control transfer request that is placed on the class.* void usb_handle_isr ()
- *Handle USB interrupts.* void usb_send_data (uns8 ep, uns8 *data, uns8 send_count, bit first)
- *Send data over an endpoint pipe.* void usb_send_empty_data_pkt ()
- *Send an empty data packet.* void usb_setup ()
- *Setup USB hardware ready for use.* void usb_SOF_callback (uns16 frame)
- *Callback routine triggered each time a start of frame (SOF) has been received.* void usb_stall_ep0 ()

Send a stall on control transfer endpoint. Variables

- control_mode_type control_mode
- uns8 usb_address
- setup_data_packet usb_sdp
- usb_state_type usb_state

Detailed Description

Put the following in your config.h

- - - - - - - - - - - - - - - - - - - pic_usb defines
- - - - - - - - - - - - - - - - - - -

Use this define if you would like to get USB negotiation and data transfer information out the serial (UART) port. You'll also need to include pic_serial.c in your project. define USB_DEBUG

Use this define if you would like a high level (ie, lots) of USB debug information printed out the serial (UART) port. define USB_DEBUG_HIGH

Define the highest numbered endpoint you will use (in this case, we choose 3). define USB_HIGHEST_EP 3

Define either USB_SELF_POWERED or USB_BUS_POWERED define USB_SELF_POWERED define USB_BUS_POWERED

Define your endpoint buffers. These start at 0x500 for a 18f4550. You'll always need endpoint 0, which is the control transfer endpoint, and others as well depending on your use. You don't have to declare the endpoints you don't use, even if they're not sequential.

define USB_EP0_OUT_SIZE 8 define USB_EP0_OUT_ADDR 0x0200

define USB_EP0_IN_SIZE 8 define USB_EP0_IN_ADDR 0x0208

EP1 not used

define USB_EP2_IN_SIZE 8 define USB_EP2_IN_ADDR 0x0210

define USB_EP3_OUT_SIZE 8 define USB_EP3_OUT_ADDR 0x0218 define USB_EP3_IN_SIZE 8 define USB_EP3_IN_ADDR 0x0220

Use this define if you want to get a callback each SOF (Start Of Frame), generally every 1ms define USB_CALLBACK_ON_SOF if you define it, you'll need to include this routine in your code: void usb_SOF_callback(uns16 frame) { }

Use this define if you would like to know when your device has been configured and is ready for use define USB_CALLBACK_ON_DEVICE_CONFIGURED if you define it, you'll need to include this routine in your code: void usb_device_configured_callback() { }

Use this define if your device uses class control transfers, eg, CDC (virtual serial port) is one that does define USB_CALLBACK_ON_CTRL_CLASS if you define it, you'll need to include these routines in your code: void usb_handle_class_ctrl_read_callback(); void usb_handle_class_ctrl_write_callback(uns8 *data, uns16 count); void usb_handle_class_request_callback(setup_data_packet sdp);

Use this define if you would like to get notified when data has arrived or been succesfully sent define USB_EP_DATA_CALLBACK if you define it, you'll need to include these routines in your code: void usb_ep_data_out_callback(uns8 end_point, uns8 *buffer_location, uns16 byte_count); void usb_ep_data_in_callback(uns8 end_point, uns16 byte_count);

Put the following in your ISR

usb_handle_isr();

Put the following in your system setup routine

Setup USB usb_setup();

Turn on USB interrupts void turn_usb_ints_on();

Turn on global interrupts turn_global_ints_on();

When you're ready to start the USB subsystem and negotiate address, send descriptors etc, call

usb_enable_module();

Define Documentation

#define BC8 0

#define BC9 1

#define BSTALL 2

#define DATA_STAGE_DIR 7

#define dt_CONFIGURATION 0x02

#define dt_CS_INTERFACE 0x24

#define dt_DEBUG 0x0a

#define dt_DEVICE 0x01

#define dt_DEVICE_QUALIFIER 0x06

#define dt_ENDPOINT 0x05

#define dt_HID 0x21

#define dt_HID_REPORT 0x22

#define dt_INTERFACE 0x04

#define dt_INTERFACE_ASSOC 0x0b

#define dt_INTERFACE_POWER 0x08

#define dt_OTG 0x09

#define dt_OTHER_SPEED_CONFIG 0x07

#define dt_STRING 0x03

#define DTS 6

#define DTSEN 3

#define INCDIS 4

#define KEN 5

#define PID0 2

#define PID1 3

#define PID2 4

#define PID3 5

#define pid_ACK 0b00000010

#define pid_DATA0 0b00000011

#define pid_DATA1 0b00001011

#define pid_DATA2 0b00000111

#define pid_IN 0b00001001

#define pid_MDATA 0b00001111

#define pid_NAK 0b00001010

#define pid_NYET 0b00000110

#define pid_OUT 0b00000001

#define pid_SETUP 0b00001101

#define pid_SOF 0b00000101

#define pid_STALL 0b00001110

#define req_Clear_Feature 0x01

#define req_Get_Configuration 0x08

#define req_Get_Descriptor 0x06

#define req_Get_Interface 0x0a

#define req_Get_Status 0x00

#define req_Set_Address 0x05

#define req_Set_Configuration 0x09

#define req_Set_Descriptor 0x07

#define req_Set_Feature 0x03

#define req_Set_Interface 0x0b

#define req_Synch_Frame 0x0c

#define REQUEST_TYPE0 5

#define REQUEST_TYPE1 6

#define UOWN 7

#define usb_send_status_ack() usb_send_empty_data_pkt()

Send a status acknowledge by sending an empty data packet

Enumeration Type Documentation

enum control_mode_type

Describe the state of the control transfer

Enumerator:

cm_IDLE No control transfer taking place

cm_CTRL_WRITE_DATA_STAGE Device receiving data during the data stage

cm_CTRL_WRITE_DATA_STAGE_CLASS Device receiving data during the data stage destined for the class

cm_CTRL_READ_DATA_STAGE Device sending data during the data stage

cm_CTRL_READ_DATA_STAGE_CLASS Device class is sending data during the data stage

cm_CTRL_READ_AWAITING_STATUS Device is awaiting reception of status after sending data

cm_CTRL_WRITE_SENDING_STATUS Device is sending status after receiving data

```
211                                 {
213     cm_IDLE,
215     cm_CTRL_WRITE_DATA_STAGE,
217     cm_CTRL_WRITE_DATA_STAGE_CLASS,
219     cm_CTRL_READ_DATA_STAGE,
221     cm_CTRL_READ_DATA_STAGE_CLASS,
223     cm_CTRL_READ_AWAITING_STATUS,
225     cm_CTRL_WRITE_SENDING_STATUS,
226 } control_mode_type;
```

enum usb_state_type

Handle the different states that USB device can be in

Enumerator:

st_POWERED USB device is powered up, ready to start negotiating

st_DEFAULT USB device is now negotiating

st_ADDRESS USB device now has an address

st_CONFIGURED USB device is completely configured and ready to rock and roll

```
141 {
143     st_POWERED,
145     st_DEFAULT,
147     st_ADDRESS,
149     st_CONFIGURED
150 } usb_state_type;
```

enum usb_status_type

Handle the special case of when we send a status ack and THEN change the address. So we need to know that the USB device is in that micro state (ie, received the new address but not yet sent the status

Enumerator:

us_IDLE

us_SET_ADDRESS

```
232                                 {
233     us_IDLE,
234     us_SET_ADDRESS
235 } usb_status_type;
```

Function Documentation

void turn_usb_ints_on ()

If you are using interrupt-driven code (generally the best way of doing things) you can turn on USB interrupts using turn_usb_ints_on(). Don't forget that you will also need to call turn_global_ints_on() as well. Typically this is called in your system setup routine.

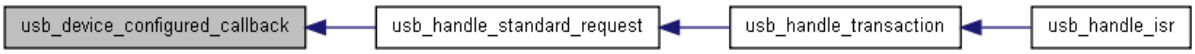
```
962                             {
963
964     set_bit(uie,  STALLIE); // interrupt on stall
965     set_bit(uie,  TRNIE);   //   on transaction complete
966     set_bit(uie,  URSTIE);  //   on reset
967     set_bit(pie2, USBIE);   // general USB interrupts
968     #ifdef USB_CALLBACK_ON_SOF
969         set_bit(uie, SOFIE);
970     #endif
971 }
```

void usb_device_configured_callback ()

Once descriptors have been received by the host and the host has selected a configuration to use, this routine is triggered. Typically this means that negotiations have completed successfully and an appropriate driver has been loaded.

In order for this callback to be triggered, you must define USB_CALLBACK_ON_DEVICE_CONFIGURED in your config.h

Here is the caller graph for this function:



void usb_enable_module ()

After you've called usb_setup(), you can call usb_enable_module() whenever you're ready for USB negotiations to occur. Normally, this would need to occur relatively quickly after power-up if your PIC is powered by USB and it's purpose is to talk over USB. This is normally called from your main() routine once all other configuration is done.

Once the USB module has successfully negotiated a connection with the host, usb_device_configured_callback() will be called if you have requested this in your config.h file. This will indicate a successful connection. Because of the way USB works, there is no way to tell that it *hasn't* worked, except via a timer - if you haven't had a good connection in several seconds, you can assume it has failed (although this may just mean the user is hunting for a driver disk etc).

```
1033                            {
1034     uir = 0;
1035     set_bit(ucon, USBEN);   // enable USB serial interface engine (SIE)
1036     usb_state = st_DEFAULT;
1037 }
```

void usb_ep_data_in_callback (uns8 *end_point*, uns16 *byte_count*)

If you have called usb_send_data to transfer data to the host, this routine will be fired once this data has been transferred. You may send more data by using usb_send_data(). Since the current PicPack USB library supports only single buffering, transfer speed is limited by how quickly you can refill the buffer again. In the future, we may support double buffering (ping poing buffering) which will most likely improve transfer speeds (to be fair, transfer performance has not been a limiting factor in tests so far).

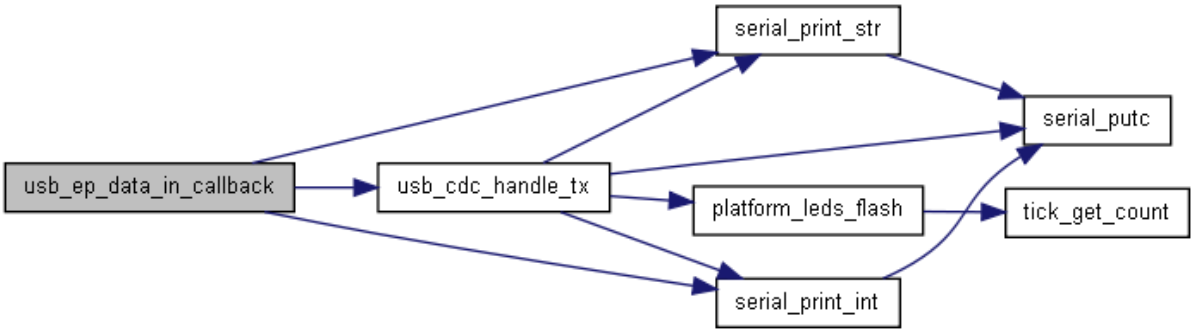
In order for this callback to be triggered, you must define USB_EP_DATA_CALLBACK in your config.h

Parameters:

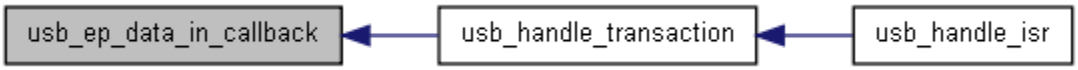
end_point The endpoint on which the data was transferred
byte_count The number of bytes that were actually transferred

```
328                                                                 {
329     #ifdef CDC_DEBUG
330         serial_print_str(" EP data in: ");
331         serial_print_int(byte_count);
332         serial_print_str(" bytes ");
333     #endif
334     // data has been sent, so do we need to send more?
335     if (end_point == USB_CDC_DATA_ENDPOINT) {   // it's the data end point
336         usb_cdc_handle_tx();
337     }
338 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_ep_data_out_callback (uns8 *end_point*, uns8 * *buffer_location*, uns16 *byte_count*)

If data is sent to the device and the endpoint is not endpoint 0 (the control transfer endpoint) then this routine is called. Since the routine is passed the actual hardware buffer location, it is important to pull data out of the buffer as soon as possible in order to free up the buffer to receive more data. The buffer is re-primed only once this routine completes since PicPack only supports single-buffered mode. In the future, we may look at supporting double buffering (ping-pong buffering) in order to be able to receive more data even while this routine is being called.

In order for this callback to be triggered, you must define USB_EP_DATA_CALLBACK in your config.h

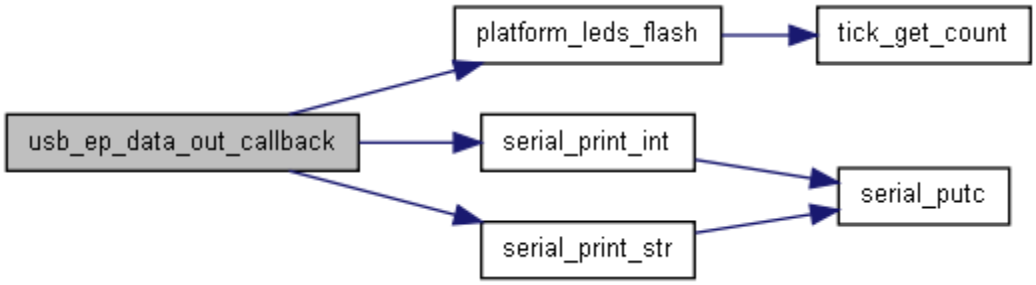
Parameters:

end_point The endpoint the data was sent do
buffer_lcoation The memory location of the USB buffer where the data was received into

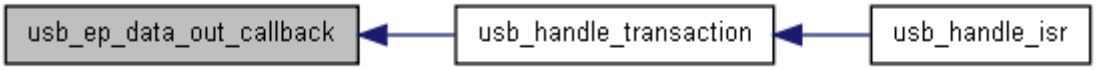
byte_count The number of bytes received

```
289                                                 {
290     uns8 cdc_rx_next;
291     #ifdef CDC_DEBUG
292         serial_print_str(" EP data out: ");
293         serial_print_int(byte_count);
294         serial_print_str(" bytes ");
295     #endif
296 
297     // We have some data!
298 
299     if (end_point == USB_CDC_DATA_ENDPOINT) {   // it's the data end point
300         uns8 count;
301         for (count = 0; count < byte_count; count++) {
302             cdc_rx_next = cdc_rx_end + 1;   // get next buffer position
303             if (cdc_rx_next == USB_CDC_RX_BUFFER_SIZE) {    // if we're at the end
304                 cdc_rx_next = 0;    // then wrap to the beginning
305             }
306             if (cdc_rx_next != cdc_rx_start) { // if space in the fifo
307                 cdc_rx_buffer[cdc_rx_end] = buffer[count]; // put it in
308                 cdc_rx_end = cdc_rx_next;  // and move pointer along
309             } else {
310                 // else... just ignore it, we've lost a byte, no room in the inn
311                 break;
312             }
313         }
314     } else {
315         #ifdef CDC_DEBUG
316             serial_print_str("data for ep ");
317             serial_print_int(end_point);
318         #endif
319     }
320 
321     #ifdef USB_CDC_USE_LEDS
322         platform_leds_flash(3);
323     #endif
324 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_get_descriptor_callback (uns8 *descriptor_type*, uns8 *descriptor_num*, uns8 ** *rtn_descriptor_ptr*, uns16 * *rtn_descriptor_size*)

Once negotiations start, descriptors are quested by the host. The device must be able to respond to these requests. Typically, this routine consists of a switch statement depending on the descriptor_type parameter. The descriptor_num is used to specify which of the descriptor_type descriptors are required, since they may be several (for example, string descriptors).

Since descriptors are specific to a particular device (and project), this callback routine and the associated descriptors are put in a file called usb_config_xxxx.c and placed in the project workspace. This is because while the descriptors could have been provided as part of the PicPack library, you will almost always want to change them to suit your application, even if only for changing the vendor and device IDs and serial numbers.

At present, descriptors are required to be in RAM.

Since descriptor requests are an essential part of the USB protocol, this callback routine is mandatory.

Here is the caller graph for this function:

usb_state_type usb_get_state ()

Returns the USB state, either powered, default, address or connect. This is updated as the negotiation progresses. It may be useful to query this if, after a suitable time-out, the connection has not been made.

```
1039                                    {
1040     return usb_state;
```

1041 }

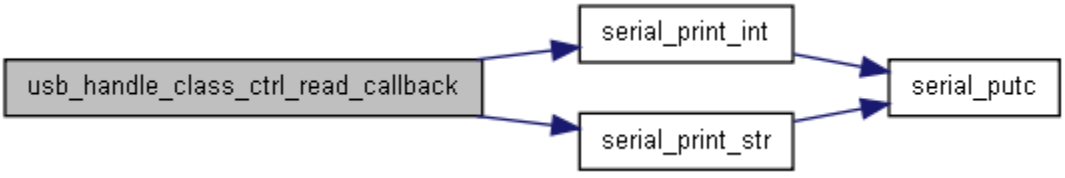
void usb_handle_class_ctrl_read_callback ()

When a control transfer is taking place, this routine is called to indicate that a control read for the class has taken place. Since everything in USB land is all about what has just happened, this callback will occur after data has been transferred to the host. If you wish to send more data to the host, use usb_send_data(), or if your control read has sent all the data required, you will need to indicate that the state has changed by setting the control_mode variable to cm_CTRL_READ_AWAITING_STATUS. This will indicate to the stack that it should now wait for the status packet before completing the control transfer.

To allow this callback to trigger, ensure you define USB_CALLBACK_ON_CLASS_CTRL in your config.h

```
273                                             {
274         switch (usb_sdp.bRequest) {
275             case req_GET_LINE_CODING:
276                 // we know we've already sent everything, so now wait for status
277                 control_mode = cm_CTRL_READ_AWAITING_STATUS;
278                 break;
279             default:
280                 #ifdef CDC_DEBUG
281                     serial_print_str(" cl read ?? ");
282                     serial_print_int(usb_sdp.bRequest);
283                 #endif
284         }
285
286 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void usb_handle_class_ctrl_write_callback (uns8 * *data*, uns16 *count*)

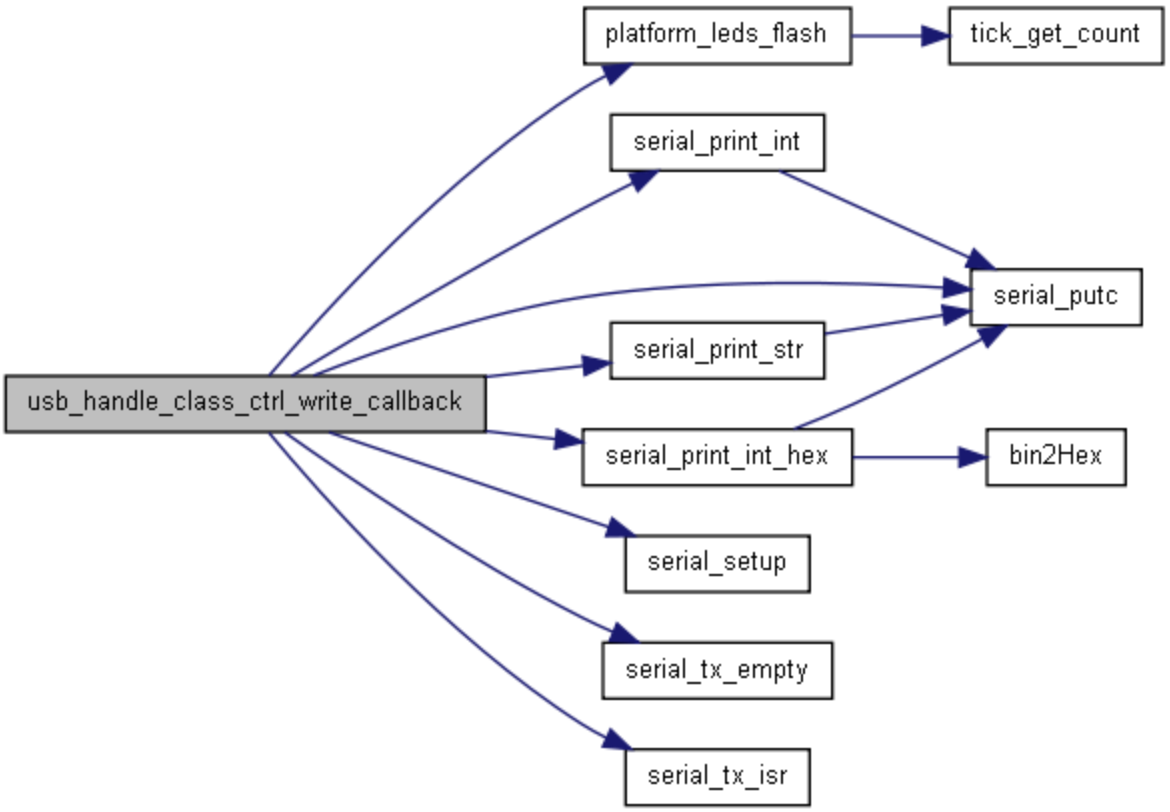
When a control transfer is taking place, this routine is called to indicate that a control write for the class has taken place. Since everything in USB land is all about what has just happened, this callback will occur after data has been received by the device. If you expect more data from the host, it will arrive in due course since endpoint 0 will be primed for more data automatically. If you have received all the data from the host, you will need to set the control_mode state variable to cm_CTRL_WRITE_SENDING_STATUS and then actually send the status by calling usb_send_status_ack(). Once the status has actually been sent, the control_mode state will automatically change to cm_IDLE to indicate the transfer has completed.

To allow this callback to trigger, ensure you define USB_CALLBACK_ON_CLASS_CTRL in your config.h

```
171                                                                 {
172
173     switch (usb_sdp.bRequest) {
174         case req_SET_LINE_CODING:
175             // dump it into class_data
176             memcpy(/* dst */ (void *)&class_data,/* src */ (void *)data, count);
177
178             // Now we need to send an ACK status back
179
180             usb_send_status_ack();
181             control_mode = cm_CTRL_WRITE_SENDING_STATUS;
182
183             line_coding *my_lc;
184             my_lc = (line_coding*) &class_data;
185             #ifdef CDC_DEBUG
186                 serial_print_int_hex(my_lc->dte_rate.as_byte_array[0]);
187                 serial_print_int_hex(my_lc->dte_rate.as_byte_array[1]);
188                 serial_print_int_hex(my_lc->dte_rate.as_byte_array[2]);
189                 serial_print_int_hex(my_lc->dte_rate.as_byte_array[3]);
190                 serial_print_str(" st=");
191                 serial_print_int(my_lc->stop_bits);
192                 serial_print_str(" p=");
193                 serial_print_int(my_lc->parity);
194                 serial_print_str(" db=");
195                 serial_print_int(my_lc->data_bits);
196                 serial_print_str(" bit rate: ");
197             #endif
198
199             dte_rate.as_byte_array[0] = my_lc->dte_rate.as_byte_array[3];
200             dte_rate.as_byte_array[1] = my_lc->dte_rate.as_byte_array[2];
201             dte_rate.as_byte_array[2] = my_lc->dte_rate.as_byte_array[1];
202             dte_rate.as_byte_array[3] = my_lc->dte_rate.as_byte_array[0];
203             parity = my_lc->parity;
204             data_bits = my_lc->data_bits;
205
206             switch (my_lc->dte_rate.as_long) {
207                 case 2400:
208                     current_bit_rate = SPBRG_2400;
209                     #ifdef CDC_DEBUG
210                     serial_print_str("2400 ");
211                     #endif
212                     break;
213                 case 4800:
214                     current_bit_rate = SPBRG_4800;
215                     #ifdef CDC_DEBUG
216                     serial_print_str("4800 ");
217                     #endif
218                     break;
219                 case 9600:
```

```
220                     current_bit_rate = SPBRG_9600;
221                     #ifdef CDC_DEBUG
222                     serial_print_str("9600 ");
223                     #endif
224                     break;
225                 case 19200:
226                     current_bit_rate = SPBRG_19200;
227                     #ifdef CDC_DEBUG
228                     serial_print_str("19200 ");
229                     #endif
230                     break;
231                 case 38400:
232                     current_bit_rate = SPBRG_38400;
233                     #ifdef CDC_DEBUG
234                     serial_print_str("38400 ");
235                     #endif
236                     break;
237                 case 115200:
238                     current_bit_rate = SPBRG_115200;
239                     #ifdef CDC_DEBUG
240                     serial_print_str("115200 ");
241                     #endif
242                     break;
243                 default:
244                     #ifdef CDC_DEBUG
245                         serial_print_str("Don't handle this bit rate");
246                     #endif
247             }
248 
249             clear_bit(rcsta, SPEN);
250             clear_bit(txsta, TXEN);
251             clear_bit(rcsta, CREN);
252 
253 
254             serial_setup(current_bit_rate);
255             if (!serial_tx_empty()) {
256                 #ifdef USB_CDC_USE_LEDS
257                     platform_leds_flash(1);
258                 #endif
259                 set_bit(pie1, TXIE);
260                 serial_tx_isr();
261             }
262             break;
263         default:
264             #ifdef CDC_DEBUG
265                 serial_print_str(" ??cw req=");
266                 serial_print_int_hex(usb_sdp.bRequest);
267                 serial_putc(' ');
268             #endif
269             break;
270     }
271 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

void usb_handle_class_request_callback (setup_data_packet *sdp*)

After receiving a setup packet, where the request is placed on the class, this routine is called. In usb_handle_class_request_callback, you can set up ready for the data stage of the control transfer. The direction of the data stage can be determined by examining test_bit(sdp.bRequest, DATA_STAGE_DIR) although generally it appears to be obvious from the request. The request is stored in sdp.bRequest.

Typically, if it is a control read transfer (that is, it is a request by the host for data), then you will need to move the control_mode state variable to cm_CTRL_READ_DATA_STAGE_CLASS and send data using usb_send_data(). If you only intend to send one packet, you can immediately move the control_mode state variable to cm_CTRL_READ_AWAITING_STATUS to indicate you are waiting for the status to arrive. You could wait for the usb_handle_class_ctrl_read callback and do it (move to cm_CTROL_READ_AWAITING_STATUS) but the PicPack USB stack can handle the control read event for you if you've already switched states.

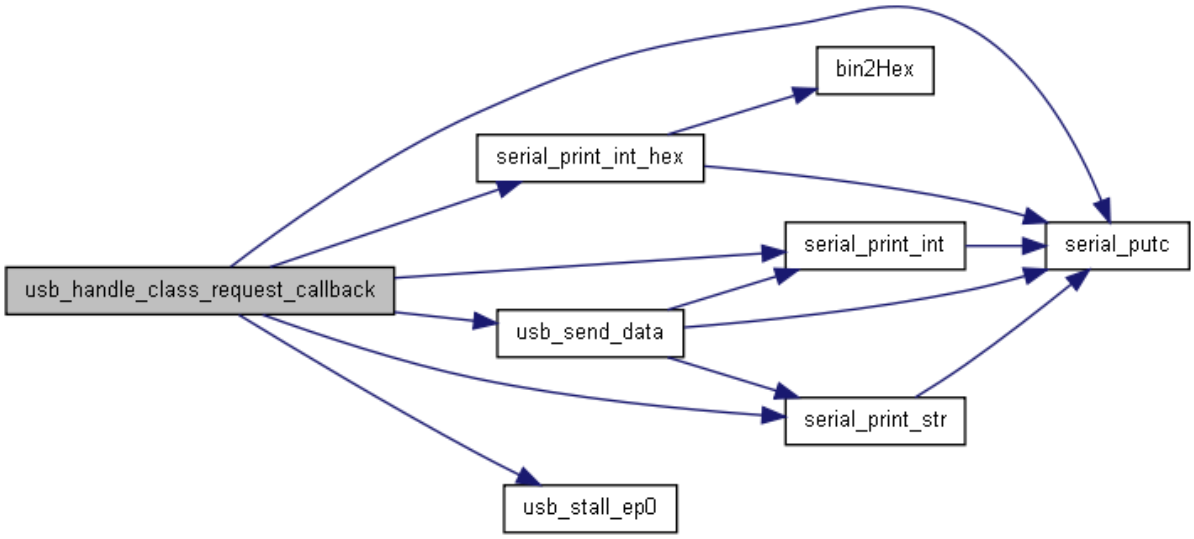
If it is a control write transfer (that is, it is a request by the host to send data to the device), then you will need to move the control_mode state variable to cm_CTRL_WRITE_DATA_STAGE_CLASS. Then, the usb_handle_class_ctrl_write will be fired when data is received by the device in the data stage.

To allow this callback to trigger, ensure you define USB_CALLBACK_ON_CLASS_CTRL in your config.h

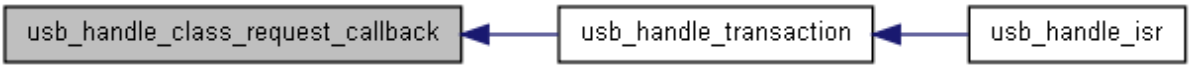
```
115                                                 {
```

```
116
117     switch (sdp.bRequest) {
118         case req SET LINE CODING:
119             // we now expect the line coding to arrive in the data stage
120
121             #ifdef CDC_DEBUG
122                 serial print str("SET_LINE ");
123             #endif
124             control mode = cm CTRL WRITE DATA STAGE CLASS;
125             break;
126         case req GET LINE CODING:
127             #ifdef CDC_DEBUG
128                 serial print str("GET_LINE ");
129                 serial print str(" len=");
130                 serial print int(sdp.wLength);
131                 serial putc(' ');
132             #endif
133             //control_mode = cm_CTRL_READ_DATA_STAGE_CLASS;
134             control mode = cm CTRL READ DATA STAGE CLASS;
135             //  need to prime ep0 IN with some funky data here
136             line coding my_line_coding;
137
138             // We stored dte rate from ealier
139             my_line_coding.dte rate.as byte array[0] = dte rate.as byte array[3];
140             my_line_coding.dte rate.as byte array[1] = dte rate.as byte array[2];
141             my_line_coding.dte rate.as byte array[2] = dte rate.as byte array[1];
142             my_line_coding.dte rate.as byte array[3] = dte rate.as byte array[0];
143             my line coding.stop bits = 0;   // 1 stop bit
144             my_line_coding.data bits = data bits;
145             my_line_coding.parity = parity;
146
147             usb send data(/*ep*/ 0, /*data*/ (uns8 *)&my line coding, /*count*/
sizeof(my_line_coding), /*first*/ 1);
148             // actually we know this will be the last packet, so go straight to waiting for
the status ack
149             control mode = cm CTRL READ AWAITING STATUS;
150
151             break;
152         case req SET CONTROL LINE STATE:
153             #ifdef CDC DEBUG
154                 serial print str("scls=");//dtr = bit 0, rts = bit 1
155                 serial print int hex(sdp.wValue);
156             #endif
157             // no data, so just ack the status
158             // !!! set dtr, rts
159             usb send status ack();
160             control mode = cm CTRL WRITE SENDING STATUS;
161             // Could put a callback here for your own code when DTR or RTS change
162             break;
163         default:
164             #ifdef CDC_DEBUG
165                 serial print str("??r=");
166                 serial print int(sdp.bRequest);
167             #endif
168     }
169 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_handle_isr ()

usb_handle_isr() should be inserted in your interrupt service routine. Alternatively, if you have reason not to want to do interrupt-driven USB, for example, a bootloader, you can poll this routine.

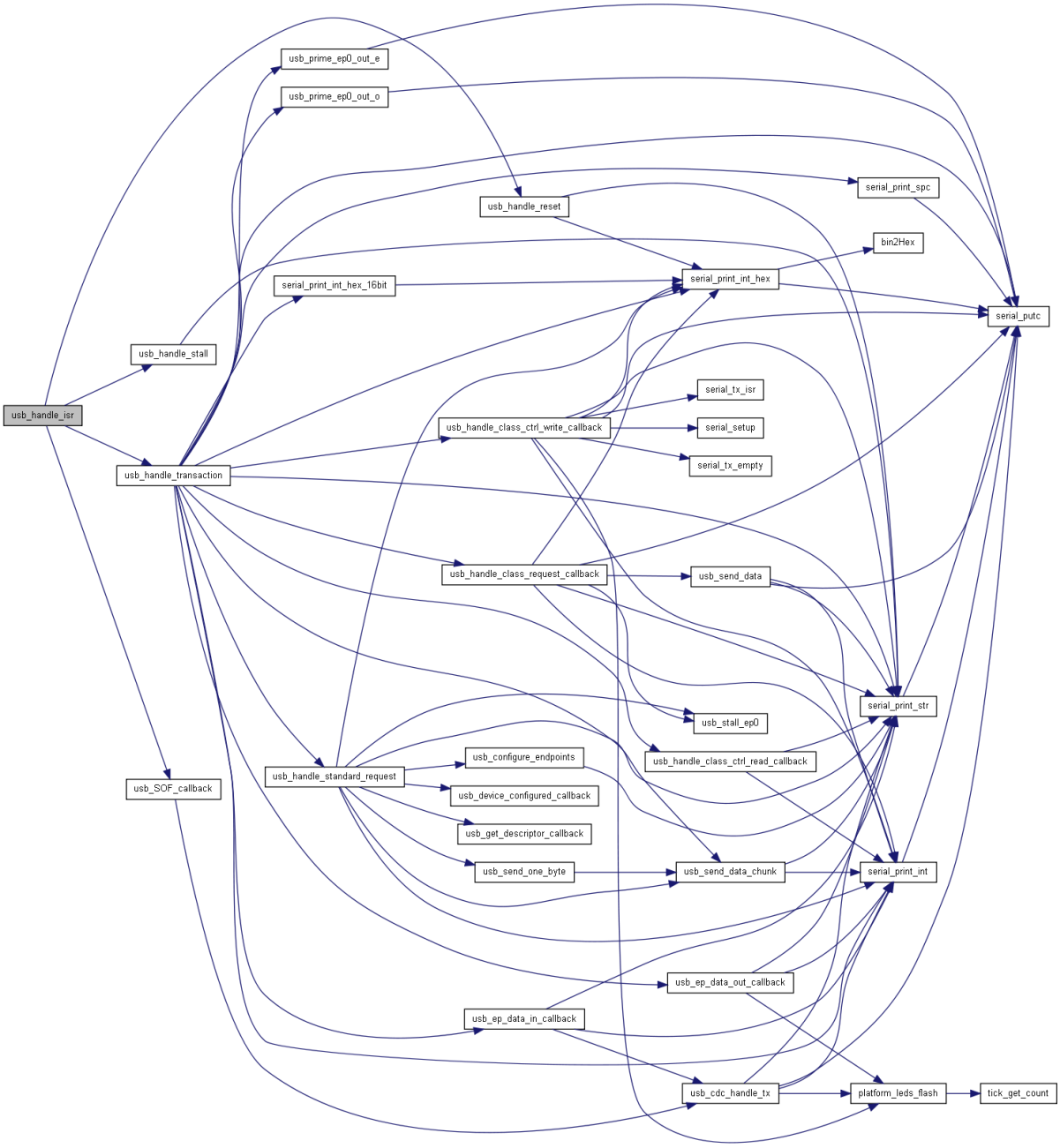
Make sure you call turn_usb_ints() and turn_global_ints_on() to ensure interrupts occur.

It will check for any of the USB interrupt flags and handle: USB transactions, USB reset, USB stall, USB Start Of Frame (including calling usb_SOF_callback() if configured in your config.h and most importantly USB transaction, which is where all the hard work is done.

```
928                     {
929
930     while (test_bit(pir2, USBIF)) {
931
932         while (test_bit(uir, TRNIF)) {
933             uns8 stat = ustat;
934             clear_bit(uir, TRNIF);
935             usb_handle_transaction(stat);
936         }
937
938         if (test_bit(uir, URSTIF)) {
939             usb_handle_reset();
940             clear_bit(uir, URSTIF);
941         }
942
943         if (test_bit(uir, STALLIF)) {
944             usb_handle_stall();
945             clear_bit(uir, STALLIF);
946         }
947         if (test_bit(uir, SOFIF)) {
948             #ifdef USB_CALLBACK_ON_SOF
949                 #ifdef ufrm
950                     usb_SOF_callback(ufrm);
951                 #else
952                     usb_SOF_callback(ufrmh << 8 | ufrml);
953                 #endif
954                 clear_bit(uir, SOFIF);
955             #endif
```

```
956         }
957
958         clear bit(pir2, USBIF);
959     }
960 }
```

Here is the call graph for this function:



void usb_send_data (uns8 *ep*, uns8 * *data*, uns8 *send_count*, bit *first*)

Use this routine to send data across the USB pipe.

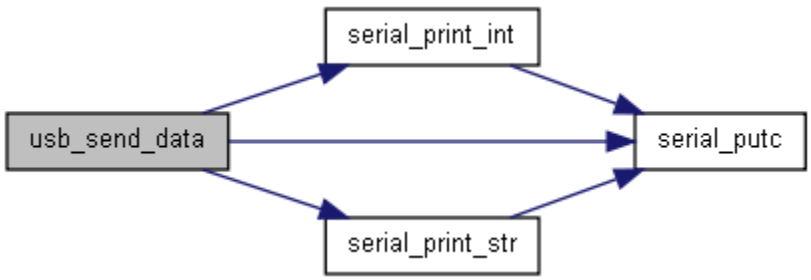
Parameters:

ep Endpoint that the data should be sent from

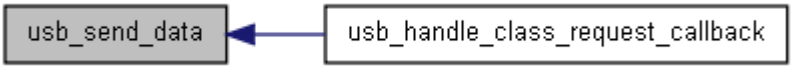
data pointer to the data
send_count the number of bytes to send
first True if this is the first in a series of sends. Generally, this can be set to False, since it will automatically be set to the right value on endpoint creation. However, in the case of control transfers, the data stage needs to have the first parameter set to True to ensure the DTS bit is set correctly.

```
211                                                                     {
212 uns8 count;
213 buffer_descriptor *bd;
214 uns8 *buffer;
215
216     // this is going to be an IN transaction
217     #ifdef USB_DEBUG
218         serial_print_str("Send:EP");
219         serial_print_int(ep);
220         serial_putc(' ');
221     #endif
222     // need to grab buffer descriptor
223     buffer = ep_in_buffer_location[ep];
224
225     bd = ep_in_bd_location[ep];
226
227     if (test_bit(bd->stat, UOWN)) {
228     #ifdef USB_DEBUG
229         serial_print_str(" !Adon't own it! ");
230     #endif
231         return;
232     }
233
234     count = 0;
235     while ((count < send_count)) {
236         buffer[count] = data[count];
237         count++;
238     }
239
240
241     bd->count = count;
242     bd->addr = (uns16)buffer;
243     if (first) {
244         clear_bit(bd->stat, DTS);   // So when it flips, will end up set
245     }
246
247     toggle_bit(bd->stat, DTS);  // flip the DTS bit
248     clear_bit(bd->stat, KEN);   // clear the keep bit
249     clear_bit(bd->stat, INCDIS);    // clear the increment disable
250     set_bit   (bd->stat, DTSEN);
251     clear_bit(bd->stat, BSTALL);    // clear stall bit
252     clear_bit(bd->stat, BC9);
253     clear_bit(bd->stat, BC8);
254
255     set_bit   (bd->stat, UOWN);  // SIE owns the buffer
256 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

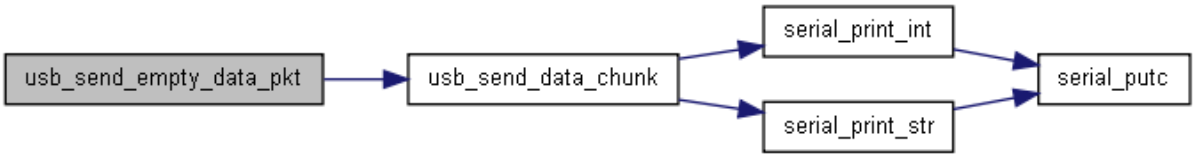


void usb_send_empty_data_pkt ()

Use this routine to send an data across the USB pipe on endpoint 0. This is the equivalent of sending a status acknowledge.

```
341                                 {
342     delivery_buffer_size = USB_EP0_IN_SIZE;
343     delivery_bd = &bd0in;
344     delivery_buffer = &buffer_0_in;
345     delivery_bytes_sent = 0;
346     delivery_bytes_to_send = 0;
347     delivery_bytes_max_send = 0;
348     delivery_ptr = (uns8 *) 0;
349     clear_bit(bd0in.stat, DTS); // ready to get toggled
350     usb_send_data_chunk();
351 }
```

Here is the call graph for this function:



void usb_setup ()

usb_setup() configures the PIC USB hardware ready for use and prepares the internal data structures used to keep track of where the endpoint buffers are.

After calling usb_setup(), you are ready to call usb_enable_module() to actually start USB negotiations. Ensure that you have usb_handle_isr() in your interrupt service routine.

```
973                 {
974
975     usb_state = st_POWERED;
976
977     // init hardware
978     #ifdef UTRDIS
979         clear_bit(ucfg, UTRDIS);    // enable internal tranceiver
980     #endif
981     set_bit  (ucfg, FSEN);  // clear for low speed, set for high speed
982     set_bit  (ucfg, UPUEN); // enable on-chip pull-ups
983
984     clear_bit(ucfg, PPB1);  // double buffering for EP0 OUT
985     set_bit(ucfg, PPB0);
986
987     // if using ping pong buffers, need to do this:
988     set_bit(ucon, PPBRST);  // reset ping pong buffers to even
989     clear_bit(ucon, PPBRST);
990
991     // init endpoint 0
992
993     set_bit(uep0,   EPHSHK);    // EP0 handshaking on
994     set_bit(uep0,   EPOUTEN);   // EP0 OUT enable
995     set_bit(uep0,   EPINEN);    // EP0 IN enable
996     clear_bit(uep0, EPCONDIS);  // EP0 control transfers on (and IN and OUT)
997
998     // init interrupts
999     // Config buffer descriptor table
1000
1001     ep_out_bd_location[0] = &bd0out_e;
```

```
1002     #if USB_HIGHEST_EP >= 1
1003         ep_out_bd_location[1] = &bd1out;
1004     #endif
1005     #if USB_HIGHEST_EP >= 2
1006         ep_out_bd_location[2] = &bd2out;
1007     #endif
1008     #if USB_HIGHEST_EP >= 3
1009         ep_out_bd_location[3] = &bd3out;
1010     #endif
1011     #if USB_HIGHEST_EP >= 4
1012         ep_out_bd_location[4] = &bd4out;
1013     #endif
1014 
1015     ep_in_bd_location[0] = &bd0in;
1016     #if USB_HIGHEST_EP >= 1
1017         ep_in_bd_location[1] = &bd1in;
1018     #endif
1019     #if USB_HIGHEST_EP >= 2
1020         ep_in_bd_location[2] = &bd2in;
1021     #endif
1022     #if USB_HIGHEST_EP >= 3
1023         ep_in_bd_location[3] = &bd3in;
1024     #endif
1025     #if USB_HIGHEST_EP >= 4
1026         ep_in_bd_location[4] = &bd4in;
1027     #endif
1028 
1029 
1030 }
```

void usb_SOF_callback (uns16 *frame*)

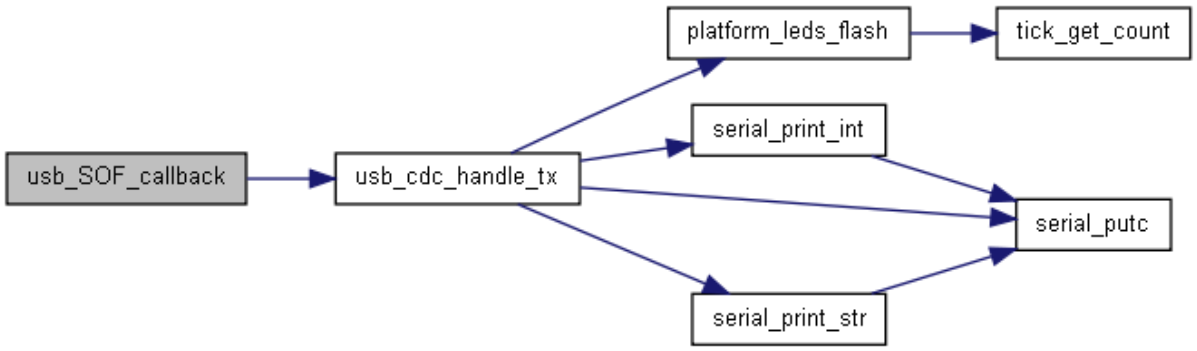
Frames in USB occur each 1ms. A SOF packet is sent to each device at the start of each frame. This is a really neat way of getting a 1ms timer without any further work.

Parameters:

frame The frame number. Frames will wrap at 65535.

```
483                                     {
484     // we don't care about the frame number, we only care if there's something to send...
485     usb_cdc_handle_tx();    // start transmission
486 }
```

Here is the call graph for this function:



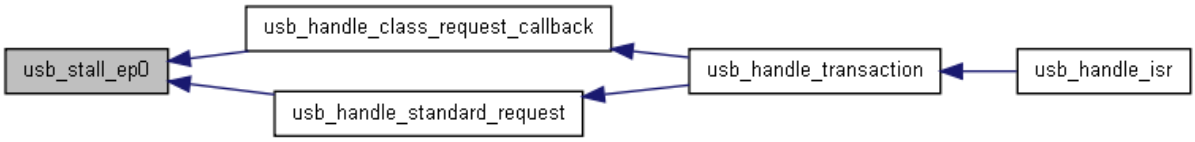
Here is the caller graph for this function:

void usb_stall_ep0 ()

Use this routine to send a stall on the control transfer endpoint - usually used to indicate that the requested function is not available.

```
203                     {
204     set_bit(bd0in.stat, BSTALL);    // stall
205     set_bit(bd0in.stat, UOWN);  // SIE owns the buffer
206 //  set_bit(bd0out.stat, BSTALL);   // stall
207 //  set_bit(bd0out.stat, UOWN); // SIE owns the buffer
208     // ??set_bit(uep0,EPSTALL);
209 }
```

Here is the caller graph for this function:



Variable Documentation

control_mode_type control_mode

Store the control mode state

uns8 usb_address

Store the usb address

setup_data_packet usb_sdp

Store the last setup data packet

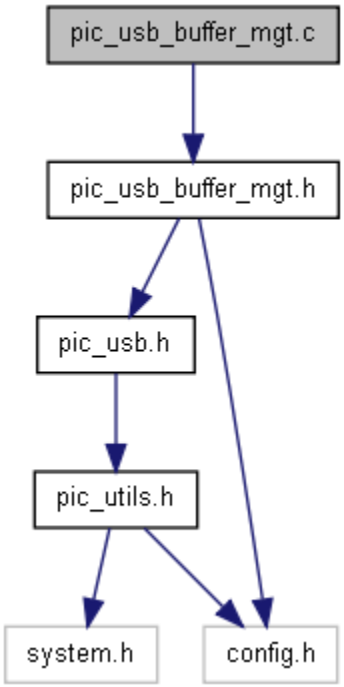
usb_state_type usb_state

Store the current USB device state

---

pic_usb_buffer_mgt.c File Reference

Include dependency graph for pic_usb_buffer_mgt.c:



Variables

- buffer_descriptor bd0in
- buffer_descriptor bd0out_e
- buffer_descriptor bd0out_o
- buffer_descriptor bd1in
- buffer_descriptor bd1out
- buffer_descriptor bd2in
- buffer_descriptor bd2out
- buffer_descriptor bd3in
- buffer_descriptor bd3out
- buffer_descriptor bd4in
- buffer_descriptor bd4out
- buffer_descriptor bd5in
- buffer_descriptor bd5out
- buffer_descriptor bd6in
- buffer_descriptor bd6out
- buffer_descriptor bd7in
- buffer_descriptor bd7out
- buffer_descriptor * ep_in_bd_location [USB_HIGHEST_EP+1]
- uns8 * ep_in_buffer_location [USB_HIGHEST_EP+1]
- uns16 ep_in_buffer_size [USB_HIGHEST_EP+1]
- buffer_descriptor * ep_out_bd_location [USB_HIGHEST_EP+1]
- uns8 * ep_out_buffer_location [USB_HIGHEST_EP+1]
- uns16 ep_out_buffer_size [USB_HIGHEST_EP+1]
- uns8 buffer_0_in[USB_EP0_IN_SIZE] USB_EP0_IN_ADDR
- uns8 buffer_0_out_e[USB_EP0_OUT_E_SIZE] USB_EP0_OUT_E_ADDR
- uns8 buffer_0_out_o[USB_EP0_OUT_O_SIZE] USB_EP0_OUT_O_ADDR

Variable Documentation

buffer_descriptor bd0in

buffer_descriptor bd0out_e

buffer_descriptor bd0out_o

buffer_descriptor bd1in

buffer_descriptor bd1out

buffer_descriptor bd2in

buffer_descriptor bd2out

buffer_descriptor bd3in

buffer_descriptor bd3out

buffer_descriptor bd4in

buffer_descriptor bd4out

buffer_descriptor bd5in

buffer_descriptor bd5out

buffer_descriptor bd6in

buffer_descriptor bd6out

buffer_descriptor bd7in

buffer_descriptor bd7out

buffer_descriptor* ep_in_bd_location[USB_HIGHEST_EP+1]

uns8* ep_in_buffer_location[USB_HIGHEST_EP+1]

uns16 ep_in_buffer_size[USB_HIGHEST_EP+1]

buffer_descriptor* ep_out_bd_location[USB_HIGHEST_EP+1]

uns8* ep_out_buffer_location[USB_HIGHEST_EP+1]

uns16 ep_out_buffer_size[USB_HIGHEST_EP+1]

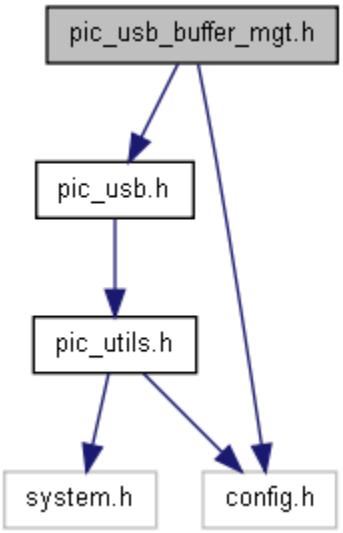
uns8 buffer_0_in [USB_EP0_IN_SIZE] USB_EP0_IN_ADDR

uns8 buffer_0_out_e [USB_EP0_OUT_E_SIZE] USB_EP0_OUT_E_ADDR

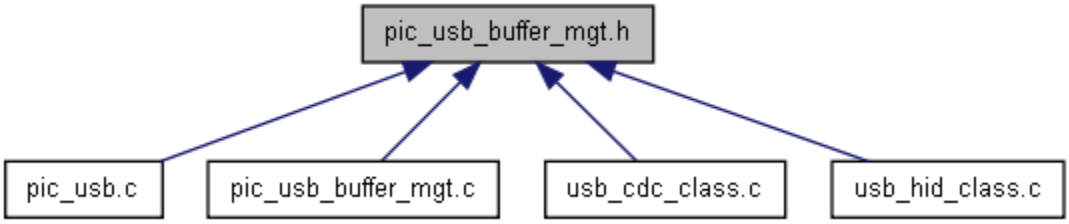
uns8 buffer_0_out_o [USB_EP0_OUT_O_SIZE] USB_EP0_OUT_O_ADDR

pic_usb_buffer_mgt.h File Reference

Pic USB buffer routines.
Include dependency graph for pic_usb_buffer_mgt.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define __pic_ubs_buffer_mgt_h

Variables

- buffer_descriptor bd0in
- buffer_descriptor bd0out_e
- buffer_descriptor bd0out_o
- buffer_descriptor bd1in
- buffer_descriptor bd1out
- buffer_descriptor bd2in
- buffer_descriptor bd2out
- buffer_descriptor bd3in
- buffer_descriptor bd3out
- buffer_descriptor bd4in
- buffer_descriptor bd4out

- buffer_descriptor bd5in
- buffer_descriptor bd5out
- buffer_descriptor bd6in
- buffer_descriptor bd6out
- buffer_descriptor bd7in
- buffer_descriptor bd7out
- uns8 buffer_0_in [USB_EP0_IN_SIZE]
- uns8 buffer_0_out_e [USB_EP0_OUT_E_SIZE]
- uns8 buffer_0_out_o [USB_EP0_OUT_O_SIZE]
- buffer_descriptor * ep_in_bd_location [USB_HIGHEST_EP+1]
- uns8 * ep_in_buffer_location [USB_HIGHEST_EP+1]
- uns16 ep_in_buffer_size [USB_HIGHEST_EP+1]
- buffer_descriptor * ep_out_bd_location [USB_HIGHEST_EP+1]
- uns8 * ep_out_buffer_location [USB_HIGHEST_EP+1]
- uns16 ep_out_buffer_size [USB_HIGHEST_EP+1]

Detailed Description

Buffer data structures for USB transfers

Define Documentation

#define __pic_ubs_buffer_mgt_h

Variable Documentation

buffer_descriptor bd0in

buffer_descriptor bd0out_e

buffer_descriptor bd0out_o

buffer_descriptor bd1in

buffer_descriptor bd1out

buffer_descriptor bd2in

buffer_descriptor bd2out

buffer_descriptor bd3in

buffer_descriptor bd3out

buffer_descriptor bd4in

buffer_descriptor bd4out

buffer_descriptor bd5in

buffer_descriptor bd5out

buffer_descriptor bd6in

buffer_descriptor bd6out

buffer_descriptor bd7in

buffer_descriptor bd7out

uns8 buffer_0_in[USB_EP0_IN_SIZE]

uns8 buffer_0_out_e[USB_EP0_OUT_E_SIZE]

uns8 buffer_0_out_o[USB_EP0_OUT_O_SIZE]

buffer_descriptor* ep_in_bd_location[USB_HIGHEST_EP+1]

uns8* ep_in_buffer_location[USB_HIGHEST_EP+1]

uns16 ep_in_buffer_size[USB_HIGHEST_EP+1]

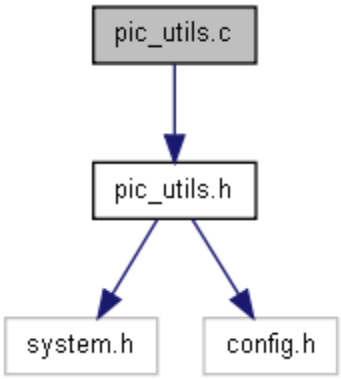
buffer_descriptor* ep_out_bd_location[USB_HIGHEST_EP+1]

uns8* ep_out_buffer_location[USB_HIGHEST_EP+1]

uns16 ep_out_buffer_size[USB_HIGHEST_EP+1]

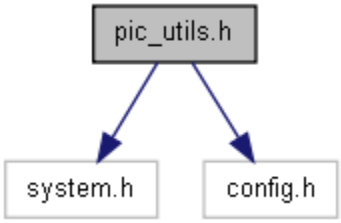
pic_utils.c File Reference

Include dependency graph for pic_utils.c:



pic_utils.h File Reference

Generic PIC helper routines.
Include dependency graph for pic_utils.h:



Defines

- #define change_pin(port, pin, value)
- #define change_pin_var(port, pin, value) change_pin(port, pin, value)
- #define clear_pin(port, pin) clear_bit(port_array[port - PORTA], pin);
- #define clear_pin_var(port, pin) clear_pin(port, pin)
- #define end_crit_sec() intcon.GIE = store_gie
- #define int16 int
- #define int32 long
- #define int8 char
- #define kill_interrupts()
- #define MAGIC_BOOSTBLOADER_REQUEST 4
- #define make_input(port, pin) set_bit(tris_array[port - PORTA], pin)
- #define make_output(port, pin) clear_bit(tris_array[port - PORTA], pin)
- #define NUMBER_PORTS 1
- #define set_pin(port, pin) set_bit(port_array[port - PORTA], pin);

- #define set_pin_var(port, pin) set_pin(port, pin)
- #define start_crit_sec()
- #define test_output_pin(port, pin) ((port_array[port - PORTA] & (1 << pin)) != 0)
- #define test_pin(port, pin) ((port_in_array[port - PORTA] & (1 << pin)) != 0)
- #define test_pin_var(port, pin) test_pin(port, pin)
- #define toggle_pin(port, pin) port_array[port - PORTA] ^= (1 << (pin));
- #define toggle_pin_var(port, pin) toggle_pin(port, pin)
- #define turn_global_ints_off() clear_bit(intcon, GIE)
- #define turn_global_ints_on() set_bit(intcon, GIE)
- #define turn_peripheral_ints_off() clear_bit(intcon, PEIE)
- #define turn_peripheral_ints_on() set_bit(intcon, PEIE)
- #define uns16 unsigned int
- #define uns32 unsigned long
- #define uns8 unsigned char

Detailed Description

Defines datatypes, port/pin access helpers

Define Documentation

#define change_pin(port, pin, value)

```
Value:if (value) { \
        set_pin(port, pin); \
    } else { \
        clear_pin(port, pin); \
    }
```

#define change_pin_var(port, pin, value) change_pin(port, pin, value)

#define clear_pin(port, pin) clear_bit(port_array[port - PORTA], pin);

#define clear_pin_var(port, pin) clear_pin(port, pin)

#define end_crit_sec() intcon.GIE = store_gie

#define int16 int

#define int32 long

#define int8 char

#define kill_interrupts()

```
Value:clear_bit(intcon, GIE); \
 nop(); \
 nop(); \
 nop(); \
 nop()
```

#define MAGIC_BOOSTBLOADER_REQUEST 4

#define make_input(port, pin) set_bit(tris_array[port - PORTA], pin)

#define make_output(port, pin) clear_bit(tris_array[port - PORTA], pin)

#define NUMBER_PORTS 1

#define set_pin(port, pin) set_bit(port_array[port - PORTA], pin);

#define set_pin_var(port, pin) set_pin(port, pin)

#define start_crit_sec()

```
Value:bit store_gie = intcon.GIE; \
    kill interrupts()
```

#define test_output_pin(port, pin) ((port_array[port - PORTA] & (1 << pin)) != 0)

#define test_pin(port, pin) ((port_in_array[port - PORTA] & (1 << pin)) != 0)

#define test_pin_var(port, pin) test_pin(port, pin)

#define toggle_pin(port, pin) port_array[port - PORTA] ^= (1 << (pin));

#define toggle_pin_var(port, pin) toggle_pin(port, pin)

#define turn_global_ints_off() clear_bit(intcon, GIE)

#define turn_global_ints_on() set_bit(intcon, GIE)

#define turn_peripheral_ints_off() clear_bit(intcon, PEIE)

#define turn_peripheral_ints_on() set_bit(intcon, PEIE)

#define uns16 unsigned int

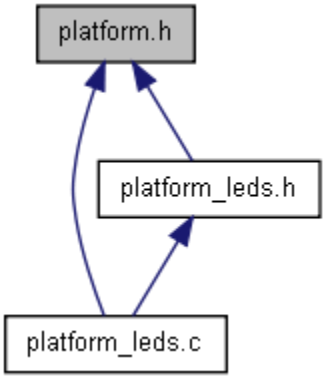
#define uns32 unsigned long

#define uns8 unsigned char

---

platform.h File Reference

Platform definitions.
This graph shows which files directly or indirectly include this file:



Defines

- #define EA_DSP0801 11
- #define EA_DSP_0801 11
- #define EA_LED_PANEL_DRIVER 7
- #define EA_PLT1001 7
- #define EA_PLT1002 8
- #define EA_PLT1003 10
- #define EA_PLT_1001 7
- #define EA_PLT_1002 8
- #define EA_PLT_1003 10
- #define EA_USB2SERIAL 10
- #define EA_WEATHER_STATION 9
- #define EA_WIRELESS_TEMP_SENSOR 8
- #define OLIMEX_BOARD 2
- #define OLIMEX_PIC_LCD3310 5
- #define SCHMARTBOARD 6
- #define SFE_TDN_V1 3
- #define SURE_PICDEM_2 1
- #define TECH_TOYS_PIC18F4550 4
- #define TIRTLE_BOARD 12

Detailed Description

Define Documentation

#define EA_DSP0801 11

#define EA_DSP_0801 11

#define EA_LED_PANEL_DRIVER 7

EA Clock board

#define EA_PLT1001 7

#define EA_PLT1002 8

#define EA_PLT1003 10

#define EA_PLT_1001 7

#define EA_PLT_1002 8

#define EA_PLT_1003 10

#define EA_USB2SERIAL 10
EA USB2TTL board

#define EA_WEATHER_STATION 9
EA weather station

#define EA_WIRELESS_TEMP_SENSOR 8
EA wireless temp sensor board

#define OLIMEX_BOARD 2
Standard Olimex board

#define OLIMEX_PIC_LCD3310 5
Olimex PIC LCD with Nokia 3310 LCD dispaly

#define SCHMARTBOARD 6
Schmartboard

#define SFE_TDN_V1 3
SparkFun RF Terminal Development Node

#define SURE_PICDEM_2 1
Sure Electronics PicDem 2 demo board

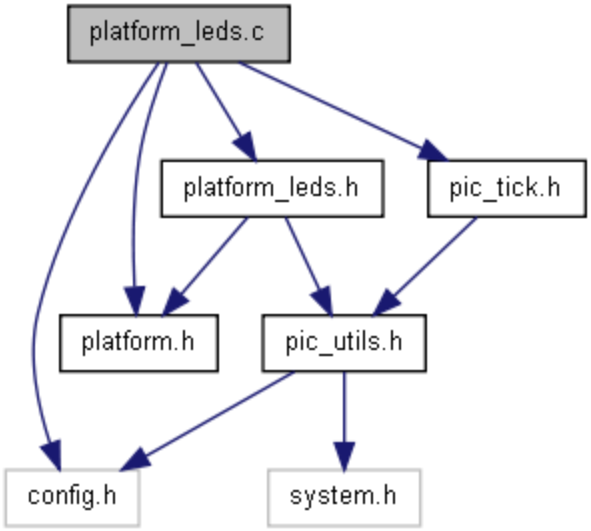
#define TECH_TOYS_PIC18F4550 4
Tech Toys HK Pic 18f4550 Eval Dev 4a

#define TIRTLE_BOARD 12

---

platform_leds.c File Reference

Include dependency graph for platform_leds.c:



Defines

- #define PLATFORM_LEDS_FLASH_TICKS 250

Functions

- void platform_leds_flash (uns8 led)
- void platform_leds_flashing (uns8 led, uns8 enable)
- void platform_leds_off (uns8 led)
- void platform_leds_on (uns8 led)
- void platform_leds_process ()
- void platform_leds_setup_io ()

Define Documentation

#define PLATFORM_LEDS_FLASH_TICKS 250

Function Documentation

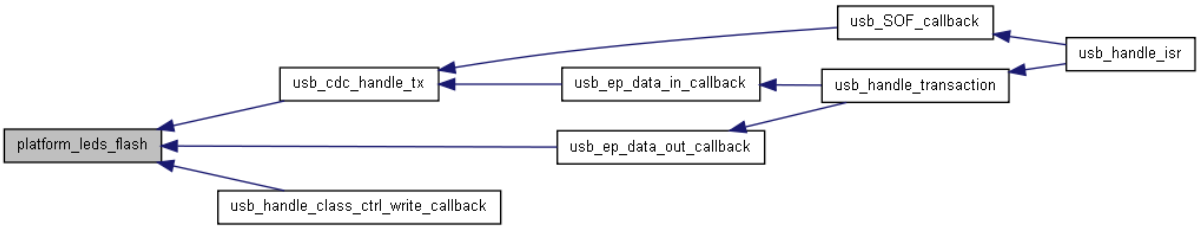
void platform_leds_flash (uns8 *led*)

```
116                                     {
117
118     switch (led) {
119         case 1:
120             platform_led1_on();
121             led1_start_tick = tick_get_count();
122             led1_on = 1;
123             break;
124         #ifdef led2_port
125             case 2:
126                 platform_led2_on();
127                 led2_start_tick = tick_get_count();
128                 led2_on = 1;
129                 break;
130         #endif
```

```
131         #ifdef led3_port
132             case 3:
133                 platform_led3_on();
134                 led3_start_tick = tick_get_count();
135                 led3_on = 1;
136                 break;
137         #endif
138     }
139 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void platform_leds_flashing (uns8 *led*, uns8 *enable*)

```
141                                                 {
142     switch (led) {
143         case 1:
144             led1_flashing = enable.0;
145             break;
146         #ifdef led2_port
147             case 2:
148                 led2_flashing = enable.0;
149                 break;
150         #endif
151         #ifdef led3 port
152             case 3:
153                 led3_flashing = enable.0;
154                 break;
155         #endif
156     }
157 }
```

void platform_leds_off (uns8 *led*)

```
98                                  {
99         switch (led) {
100         case 1:
101             platform_led1_off();
102             break;
103         #ifdef led2_port
104             case 2:
105                 platform_led2_off();
106                 break;
107         #endif
108         #ifdef led3 port
109             case 3:
110                 platform_led3_off();
111                 break;
```

```
112         #endif
113     }
114 }
```

void platform_leds_on (uns8 *led*)

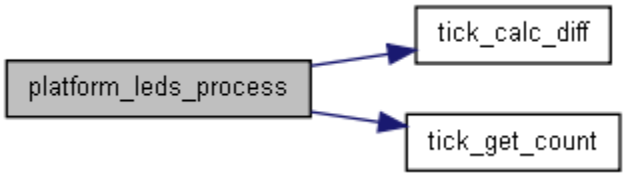
```
80                                  {
81         switch (led) {
82         case 1:
83             platform_led1_on();
84             break;
85         #ifdef led2_port
86             case 2:
87                 platform led2 on();
88                 break;
89         #endif
90         #ifdef led3_port
91             case 3:
92                 platform_led3_on();
93                 break;
94         #endif
95     }
96 }
```

void platform_leds_process ()

```
159                                 {
160
161 uns16 current tick;
162
163     current_tick = tick get count();
164
165     #ifdef led1_port
166     if (led1 on || led1 flashing) {
167         if (tick calc diff(led1_start_tick, current_tick) > PLATFORM LEDS FLASH TICKS) {
168             if (led1_on) {
169                 led1_on = 0;
170                 platform led1 off();
171             } else {
172                 led1_on = 1;
173                 platform_led1_on();
174             }
175             if (led1_flashing) {
176                 led1_start_tick = tick get count();
177             }
178         }
179     }
180     #endif
181     #ifdef led2_port
182     if (led2 on || led2 flashing) {
183         if (tick calc diff(led2_start_tick, current_tick) > PLATFORM LEDS FLASH TICKS) {
184             if (led2_on) {
185                 led2_on = 0;
186                 platform_led2_off();
187             } else {
188                 led2_on = 1;
189                 platform_led2_on();
190             }
191             if (led2_flashing) {
192                 led2_start_tick = tick get count();
193             }
194         }
195     }
196     #endif
197     #ifdef led3_port
```

```
198     if (led3_on || led1_flashing) {
199         if (tick_calc_diff(led3_start_tick, current_tick) > PLATFORM_LEDS_FLASH_TICKS) {
200             if (led3_on) {
201                 led3_on = 0;
202                 platform_led3_off();
203             } else {
204                 led3_on = 1;
205                 platform_led3_on();
206             }
207             if (led3_flashing) {
208                 led3_start_tick = tick_get_count();
209             }
210         }
211     }
212     #endif
213 }
```

Here is the call graph for this function:



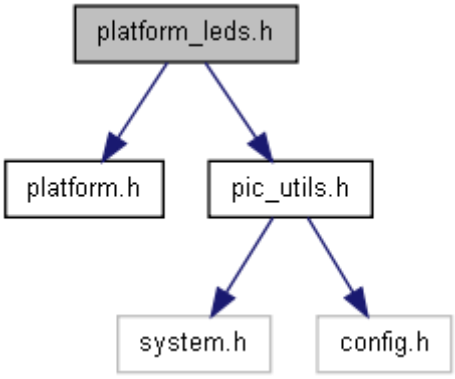
void platform_leds_setup_io ()

```
64                                  {
65
66     #ifdef led1_port
67         clear_pin(led1_port, led1_pin);
68         make_output(led1_port, led1_pin);
69     #endif
70     #ifdef led2_port
71         clear_pin(led2_port, led2_pin);
72         make_output(led2_port, led2_pin);
73     #endif
74     #ifdef led3_port
75         clear_pin(led3_port, led3_pin);
76         make_output(led3_port, led3_pin);
77     #endif
78 }
```

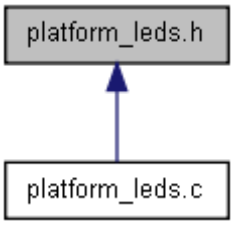
---

platform_leds.h File Reference

Easy access to the LEDs on your platform.
Include dependency graph for platform_leds.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define __platform_heds_h

Functions

- void platform_leds_flash (uns8 led)
- void platform_leds_flashing (uns8 led, uns8 enable)
- void platform_leds_off (uns8 led)
- void platform_leds_on (uns8 led)
- void platform_leds_process ()
- void platform_leds_setup_io ()

Detailed Description

Define Documentation

#define __platform_heds_h

Function Documentation

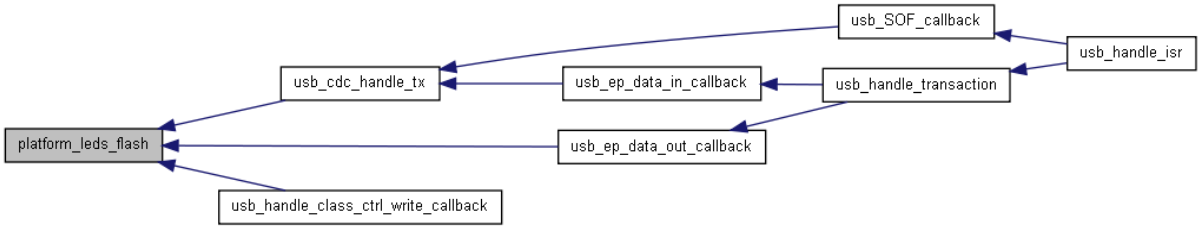
void platform_leds_flash (uns8 *led*)

```
116                                     {
117
```

```
118     switch (led) {
119         case 1:
120             platform_led1_on();
121             led1_start_tick = tick_get_count();
122             led1_on = 1;
123             break;
124         #ifdef led2_port
125             case 2:
126                 platform_led2_on();
127                 led2_start_tick = tick_get_count();
128                 led2_on = 1;
129                 break;
130         #endif
131         #ifdef led3_port
132             case 3:
133                 platform_led3_on();
134                 led3_start_tick = tick_get_count();
135                 led3_on = 1;
136                 break;
137         #endif
138     }
139 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



void platform_leds_flashing (uns8 *led*, uns8 *enable*)

```
141                                                 {
142     switch (led) {
143         case 1:
144             led1_flashing = enable.0;
145             break;
146         #ifdef led2_port
147             case 2:
148                 led2_flashing = enable.0;
149                 break;
150         #endif
151         #ifdef led3_port
152             case 3:
153                 led3_flashing = enable.0;
154                 break;
155         #endif
156     }
157 }
```

void platform_leds_off (uns8 *led*)

```
98                                  {
```

```
99          switch (led) {
100          case 1:
101              platform_led1_off();
102              break;
103          #ifdef led2_port
104              case 2:
105                  platform_led2_off();
106                  break;
107          #endif
108          #ifdef led3_port
109              case 3:
110                  platform_led3_off();
111                  break;
112          #endif
113      }
114 }
```

void platform_leds_on (uns8 *led*)

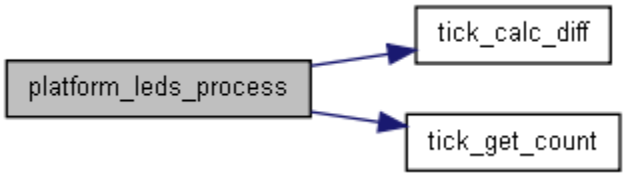
```
80                                  {
81          switch (led) {
82          case 1:
83              platform_led1_on();
84              break;
85          #ifdef led2_port
86              case 2:
87                  platform_led2_on();
88                  break;
89          #endif
90          #ifdef led3_port
91              case 3:
92                  platform_led3_on();
93                  break;
94          #endif
95      }
96 }
```

void platform_leds_process ()

```
159                               {
160
161 uns16 current_tick;
162
163     current_tick = tick_get_count();
164
165     #ifdef led1_port
166     if (led1_on || led1_flashing) {
167         if (tick_calc_diff(led1_start_tick, current_tick) > PLATFORM_LEDS_FLASH_TICKS) {
168             if (led1_on) {
169                 led1_on = 0;
170                 platform_led1_off();
171             } else {
172                 led1_on = 1;
173                 platform_led1_on();
174             }
175             if (led1_flashing) {
176                 led1_start_tick = tick_get_count();
177             }
178         }
179     }
180     #endif
181     #ifdef led2_port
182     if (led2_on || led2_flashing) {
183         if (tick_calc_diff(led2_start_tick, current_tick) > PLATFORM_LEDS_FLASH_TICKS) {
184             if (led2_on) {
```

```
185                 led2_on = 0;
186                 platform_led2_off();
187             } else {
188                 led2_on = 1;
189                 platform_led2_on();
190             }
191             if (led2_flashing) {
192                 led2_start_tick = tick_get_count();
193             }
194         }
195     }
196     #endif
197     #ifdef led3_port
198     if (led3_on || led1_flashing) {
199         if (tick_calc_diff(led3_start_tick, current_tick) > PLATFORM_LEDS_FLASH_TICKS) {
200             if (led3_on) {
201                 led3_on = 0;
202                 platform_led3_off();
203             } else {
204                 led3_on = 1;
205                 platform_led3_on();
206             }
207             if (led3_flashing) {
208                 led3_start_tick = tick_get_count();
209             }
210         }
211     }
212     #endif
213 }
```

Here is the call graph for this function:



void platform_leds_setup_io ()

```
64                                  {
65
66     #ifdef led1_port
67         clear_pin(led1_port, led1_pin);
68         make_output(led1_port, led1_pin);
69     #endif
70     #ifdef led2_port
71         clear_pin(led2_port, led2_pin);
72         make_output(led2_port, led2_pin);
73     #endif
74     #ifdef led3_port
75         clear_pin(led3_port, led3_pin);
76         make_output(led3_port, led3_pin);
77     #endif
78 }
```

protocol.h File Reference

Protocol definitions for Pkt mesh network.

Defines

- #define CAPS_INFO1_DATE 1
- #define CAPS_INFO1_TIME 0
- #define CAPS_INPUTS_SWITCH1 0
- #define CAPS_INPUTS_SWITCH2 1
- #define CAPS_INPUTS_SWITCH3 2
- #define CAPS_INPUTS_SWITCH4 3
- #define CAPS_INPUTS_SWITCH5 4
- #define CAPS_INPUTS_SWITCH6 5
- #define CAPS_INPUTS_SWITCH7 6
- #define CAPS_INPUTS_SWITCH8 7
- #define CAPS_OUTPUTS_DIMMER1 4
- #define CAPS_OUTPUTS_DIMMER2 5
- #define CAPS_OUTPUTS_DIMMER3 6
- #define CAPS_OUTPUTS_DIMMER4 7
- #define CAPS_OUTPUTS_RELAY1 0
- #define CAPS_OUTPUTS_RELAY2 1
- #define CAPS_OUTPUTS_RELAY3 2
- #define CAPS_OUTPUTS_RELAY4 3
- #define CAPS_SENSOR_AIR_PRESSURE 2
- #define CAPS_SENSOR_HUMIDITY 1
- #define CAPS_SENSOR_LIGHT 3
- #define CAPS_SENSOR_PRESENCE 4
- #define CAPS_SENSOR_TEMP 0
- #define EE_MY_ADDR_H 0x00
- #define EE_MY_ADDR_L 0x01
- #define EE_MY_INFO1 0x07
- #define EE_MY_INPUTS 0x05
- #define EE_MY_LAST_PKT_ID_H 0x02
- #define EE_MY_LAST_PKT_ID_L 0x03
- #define EE_MY_OUTPUTS 0x06
- #define EE_MY_SENSORS 0x04
- #define PL_ADDR_RESPONSE 5
- #define PL_CAPS_RESPONSE 4
- #define PL_CHANGE_RESPONSE 9
- #define PL_OTHER_RESPONSE 11
- #define PL_REQ_ADDR 2
- #define PL_REQ_CAPS 3
- #define PL_REQ_INFO1 10
- #define PL_REQ_INFORM_ON_CHANGE 8
- #define PL_REQ_SENSOR 6
- #define PL_SENSOR_RESPONSE 7
- #define PL_SET_ADDR 1
- #define PL_SET_OUTPUT 9

Detailed Description

Define Documentation

#define CAPS_INFO1_DATE 1

#define CAPS_INFO1_TIME 0

#define CAPS_INPUTS_SWITCH1 0

#define CAPS_INPUTS_SWITCH2 1

#define CAPS_INPUTS_SWITCH3 2

#define CAPS_INPUTS_SWITCH4 3

#define CAPS_INPUTS_SWITCH5 4

#define CAPS_INPUTS_SWITCH6 5

#define CAPS_INPUTS_SWITCH7 6

#define CAPS_INPUTS_SWITCH8 7

#define CAPS_OUTPUTS_DIMMER1 4

#define CAPS_OUTPUTS_DIMMER2 5

#define CAPS_OUTPUTS_DIMMER3 6

#define CAPS_OUTPUTS_DIMMER4 7

#define CAPS_OUTPUTS_RELAY1 0

#define CAPS_OUTPUTS_RELAY2 1

#define CAPS_OUTPUTS_RELAY3 2

#define CAPS_OUTPUTS_RELAY4 3

#define CAPS_SENSOR_AIR_PRESSURE 2

#define CAPS_SENSOR_HUMIDITY 1

#define CAPS_SENSOR_LIGHT 3

#define CAPS_SENSOR_PRESENCE 4

#define CAPS_SENSOR_TEMP 0

#define EE_MY_ADDR_H 0x00

#define EE_MY_ADDR_L 0x01

#define EE_MY_INFO1 0x07

#define EE_MY_INPUTS 0x05

#define EE_MY_LAST_PKT_ID_H 0x02

#define EE_MY_LAST_PKT_ID_L 0x03

#define EE_MY_OUTPUTS 0x06

#define EE_MY_SENSORS 0x04

#define PL_ADDR_RESPONSE 5

#define PL_CAPS_RESPONSE 4

#define PL_CHANGE_RESPONSE 9

#define PL_OTHER_RESPONSE 11

#define PL_REQ_ADDR 2

#define PL_REQ_CAPS 3

#define PL_REQ_INFO1 10

#define PL_REQ_INFORM_ON_CHANGE 8

#define PL_REQ_SENSOR 6

#define PL_SENSOR_RESPONSE 7

#define PL_SET_ADDR 1

#define PL_SET_OUTPUT 9

---

sfe_tdn_v1.h File Reference

Defines

- #define __sfe_tdn_v1_h defined
- #define stat1 1
- #define stat2 3
- #define stat3 4

Define Documentation

#define __sfe_tdn_v1_h defined

#define stat1 1

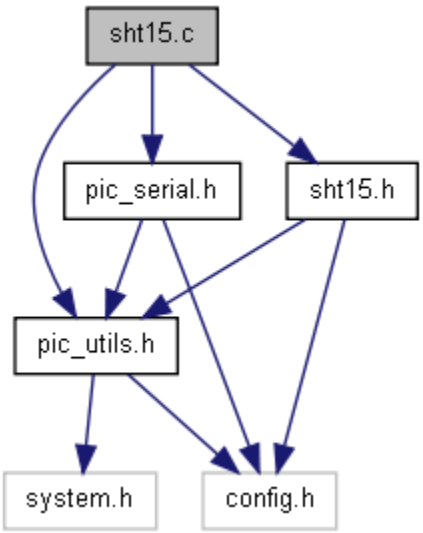
#define stat2 3

#define stat3 4

---

sht15.c File Reference

Include dependency graph for sht15.c:



Defines

- #define CHECK_HUMD 0b00000101
- #define CHECK_STAT 0b00000111
- #define CHECK_TEMP 0b00000011
- #define sht15_read_sda() set_bit(tris_array[sht15_sda_port - PORTA], sht15_sda_pin);
- #define sht15_write_sda() clear_bit(tris_array[sht15_sda_port - PORTA], sht15_sda_pin);
- #define WRITE_STAT 0b00000110

Functions

- uns16 sht15_fix_humidity (uns16 sensor_out)
- uns16 sht15_fix_humidity_l (uns8 sensor_out)
- uns16 sht15_fix_humidity_r (uns16 sensor_out)
- int16 sht15_fix_temperature_h (uns16 sensor_out)
- void sht15_read (void)
- uns16 sht15_read_byte16 (void)
- uns16 sht15_read_humidity (void)
- uns16 sht15_read_temperature (void)

- void sht15_send_byte (uns8 sht15_command)
- void sht15_setup_io (void)
- void sht15_start (void)

Define Documentation

#define CHECK_HUMD 0b00000101

#define CHECK_STAT 0b00000111

#define CHECK_TEMP 0b00000011

#define sht15_read_sda() set_bit(tris_array[sht15_sda_port - PORTA], sht15_sda_pin);

#define sht15_write_sda() clear_bit(tris_array[sht15_sda_port - PORTA], sht15_sda_pin);

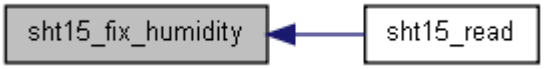
#define WRITE_STAT 0b00000110

Function Documentation

uns16 sht15_fix_humidity (uns16 *sensor_out*)

```
328 {
329
330 long result; // 16Bit unsigned for the result
331
332     if ( sensor_out <= 1712 ) {
333         result = 143 * sensor_out;
334         result = result - 8192; // result = a * sensor_out
335         //result < 512 ? result = 512; // check for underflow
336         //result = result - 8192; // result = result + b
337     } else {
338         result = 111 *sensor_out + 46288; // result = a * sensor_out
339         //result = result + 46288; // result = result + b
340         //result > 25600 ? result = 25600; // check for overflow (optional)
341     }
342     //8 MSB's are 0-100%RH integers, 8 LSB's are remainder
343     result = result / 4096;
344     return result;
345 }
```

Here is the caller graph for this function:



uns16 sht15_fix_humidity_l (uns8 *sensor_out*)

```
348 {
349
350 uns16 result; // 16Bit unsigned for the result
351
352     if ( sensor_out <= 107 ) {
353         result = 143 * sensor_out; // result = a * sensor_out
354         if (result < 512) { result = 512; } // check for underflow
```

```
355         result -= 512;
356     } else {
357         result = 111 * sensor out; // result = a * sensor out
358         result += 2893; // result = result + b
359         if (result > 25600) { result = 25600; } // check for overflow (optional)
360     }
361     //8 MSB's are 0-100%RH integers, 8 LSB's are remainder
362     result = result >> 8; // result = result / 256
363     return result;
364 }
```

Here is the caller graph for this function:

uns16 sht15_fix_humidity_r (uns16 *sensor_out*)

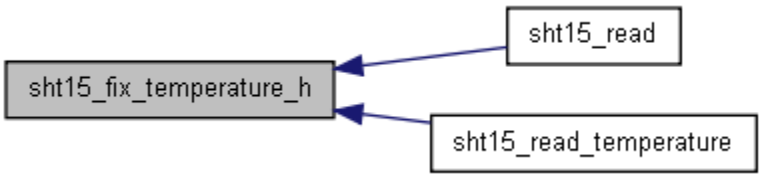
```
305                                                 {
306
307 int32 c1 = -40000000;
308 int32 c2 = 405000;
309 int32 c3 = 28;
310 int32 s;
311 int32 final;
312
313
314 s = sensor_out;
315
316 s = s * 10000000;
317
318 final = c1 + c2 * s + c3 * s * s;
319
320 final = final / 10000;
321
322 return final;
323 }
```

int16 sht15_fix_temperature_h (uns16 *sensor_out*)

```
367 {
368     int16 result;
369
370 //  result = -3963 + sensor_out;
371     result = -4001 + sensor_out;
372     return result;
```

373 }

Here is the caller graph for this function:

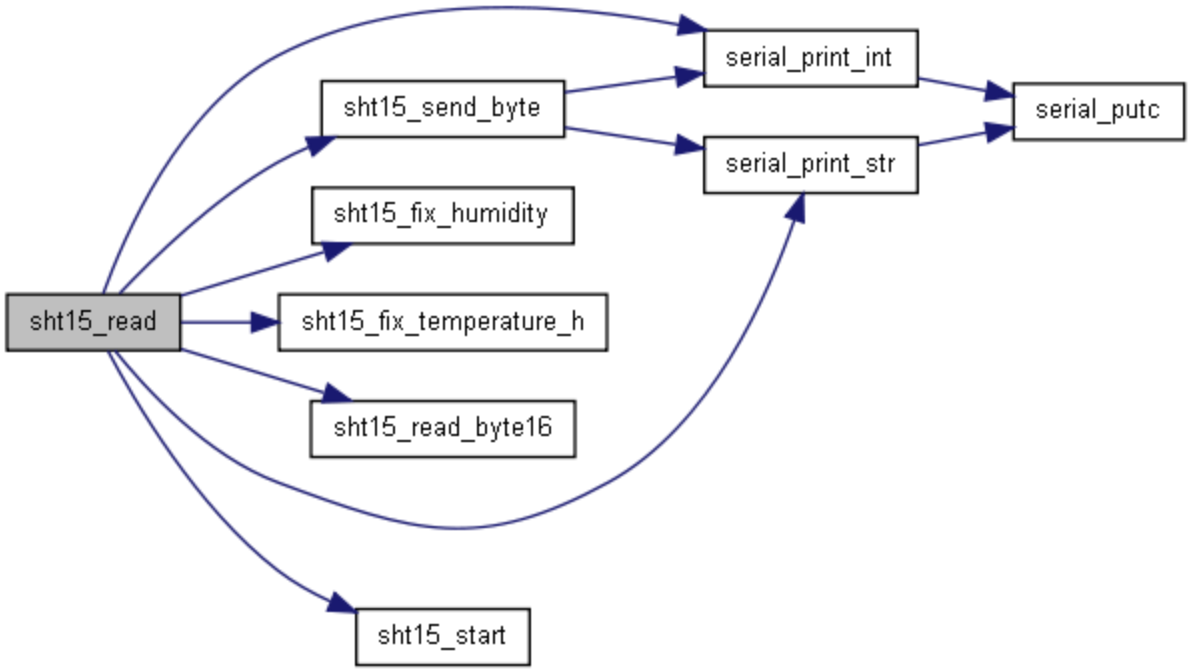


void sht15_read (void)

```
88 {
```

```
89      uns16 response;
90
91      //Issue command start
92      sht15_start();
93
94      //Now send command code
95      sht15_send_byte(CHECK_HUMD);
96
97      response = sht15_read_byte16();
98
99
100      serial_print_str("\nH=");
101      serial_print_int(response);
102      //response = response >> 4;
103      response = sht15_fix_humidity(response);
104      serial_print_str("\nH=");
105      serial_print_int(response);
106
107      sht15_start();
108      sht15_send_byte(CHECK_TEMP);
109      response = sht15_read_byte16(); //Listen for response from SHT15
110      serial_print_str("\nT=");
111      serial_print_int(response);//
112      response = sht15_fix_temperature_h(response);
113      serial_print_str("\nT=");
114      serial_print_int(response);
115
116  }
```

Here is the call graph for this function:



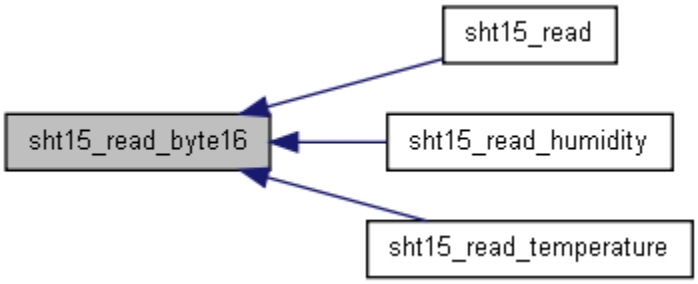
uns16 sht15_read_byte16 (void)

```
217  {
218      uns8 j;
219      uns16 in_byte;
220      uns8 crc;
221      //bit  my_store_gie = intcon.GIE;
222      //kill_interrupts();
```

```
223
224     clear pin(sht15_sck_port, sht15_sck_pin);
225
226     sht15 read sda();
227
228     for(j = 0 ; j < 8 ; j++)
229     {
230         set pin(sht15_sck_port, sht15_sck_pin);
231 delay_us(10);
232         in_byte = in_byte << 1;
233         in byte.0 = test pin(sht15 sda port, sht15 sda pin);
234
235         clear pin(sht15_sck_port, sht15_sck_pin);
236 delay_us(10);
237     }
238     delay_us(100);
239
240     // send ack
241     sht15 write sda();
242     clear pin(sht15_sda_port, sht15_sda_pin);
243
244         set pin(sht15_sck_port, sht15_sck_pin);
245 delay us(10);
246         clear pin(sht15_sck_port, sht15_sck_pin);
247 delay_us(10);
248
249     sht15 read sda();
250     delay us(100);
251
252
253     for(j = 0 ; j < 8 ; j++)
254     {
255         set pin(sht15_sck_port, sht15_sck_pin);
256 delay_us(10);
257         in_byte = in_byte << 1;
258         in_byte.0 = test pin(sht15_sda_port, sht15_sda_pin);
259
260         clear pin(sht15_sck_port, sht15_sck_pin);
261 delay_us(10);
262     }
263
264     // send ack
265     sht15 write sda();
266     clear pin(sht15 sda port, sht15 sda pin);
267
268         set pin(sht15_sck_port, sht15_sck_pin);
269 delay_us(10);
270         clear pin(sht15_sck_port, sht15_sck_pin);
271 delay us(10);
272
273     sht15 read sda();
274     delay_us(100);
275
276
277     for(j = 0 ; j < 8 ; j++)
278     {
279         set pin(sht15 sck port, sht15 sck pin);
280 delay_us(10);
281         crc = crc << 1;
282         crc.0 = test pin(sht15_sda_port, sht15_sda_pin);
283
284         clear pin(sht15_sck_port, sht15_sck_pin);
285 delay_us(10);
286     }
287
288     // send ack
289     sht15 write sda();
290     clear pin(sht15_sda_port, sht15_sda_pin);
291
292         set pin(sht15_sck_port, sht15_sck_pin);
293 delay_us(10);
```

```
294         clear_pin(sht15_sck_port, sht15_sck_pin);
295 delay_us(10);
296
297     sht15_read_sda();
298     delay_us(100);
299
300     //intcon.GIE = my_store_gie;
301     return(in_byte);
302 }
```

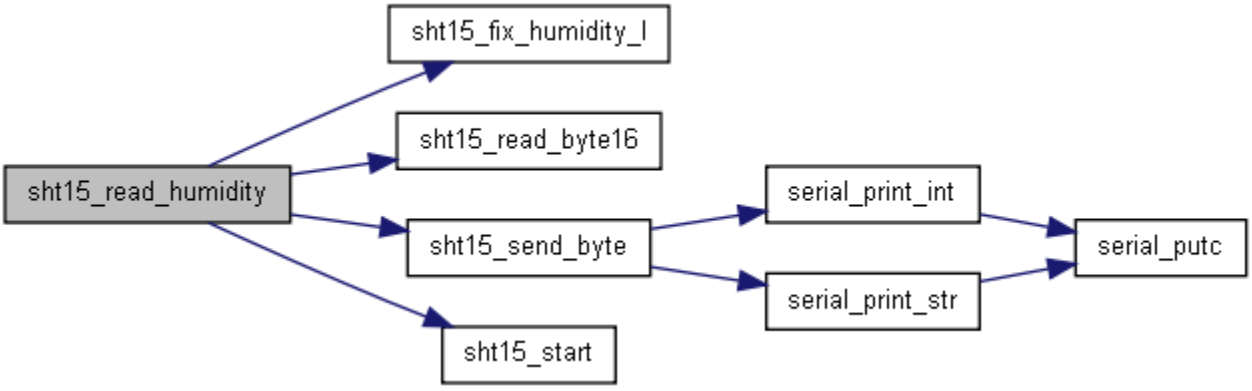
Here is the caller graph for this function:



uns16 sht15_read_humidity (void)

```
59                                  {
60
61     uns16 response;
62
63     sht15_start();
64     sht15_send_byte(CHECK_HUMD);
65     response = sht15_read_byte16();
66     response = response >> 4;
67     response = sht15_fix_humidity_l(response);
68     return response;
69 }
```

Here is the call graph for this function:

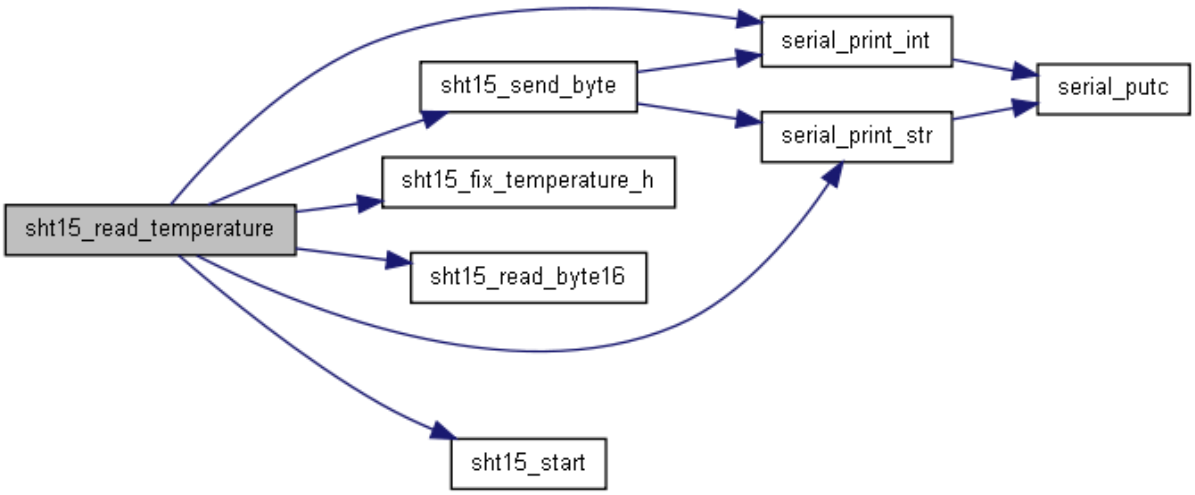


uns16 sht15_read_temperature (void)

```
71                                      {
72
73 uns16 response;
74
```

```
75      sht15_start();
76      sht15_send_byte(CHECK_TEMP);
77      response = sht15_read_byte16(); //Listen for response from SHT15
78      serial_print_str("\nT=");
79      serial_print_int(response);
80      response = sht15_fix_temperature_h(response);
81      serial_print_str("\nT=");
82      serial_print_int(response);
83
84  }
```

Here is the call graph for this function:

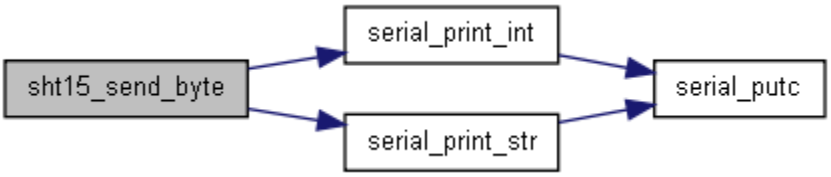


void sht15_send_byte (uns8 *sht15_command*)

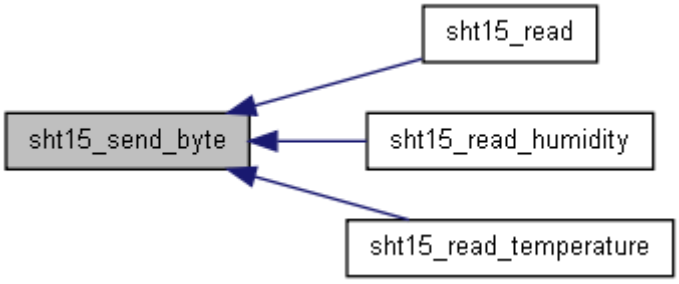
```
119 {
120     uns8 i;
121
122
123     sht15_write_sda();
124
125
126     clear_pin(sht15_sck_port, sht15_sck_pin);
127     for(i = 0 ; i < 8 ; i++)
128     {
129         delay_us(10);
130         change_pin(sht15_sda_port, sht15_sda_pin, sht15_command.7);
131         sht15_command = sht15_command << 1;
132         delay_us(10);
133         set_pin(sht15_sck_port, sht15_sck_pin);
134         delay_us(10);
135         clear_pin(sht15_sck_port, sht15_sck_pin);
136     }
137     delay_us(100);
138     //Wait for SHT15 to acknowledge.
139 //    clear_pin(sht15_sck_port, sht15_sck_pin);
140     sht15_read_sda();
141     // <<!!>> Fix following
142 //    while (test_pin(sht15_sda_port, sht15_sda_pin) == 1); //Wait for SHT to pull line low
143
144     set_pin(sht15_sck_port, sht15_sck_pin);
145     // read ack here
146
147     delay_us(10);
148     clear_pin(sht15_sck_port, sht15_sck_pin); //Falling edge of 9th clock
149
```

```
150     while (test pin(sht15_sda_port, sht15_sda_pin) == 0); //Wait for SHT to release line
151
152     //Wait for SHT15 to pull SDA low to signal measurement completion.
153     //This can take up to 210ms for 14 bit measurements
154     i = 0;
155     while ((test pin(sht15_sda_port, sht15_sda_pin) == 1)) //Wait for SHT to pull line low
156     {
157         i++;
158         if (i == 255) break;
159
160         delay ms(10);
161     }
162
163     //Debug info
164     i *= 10; //Convert to ms
165     serial print str("\nRt=");
166     serial print int(i);
167     serial print str("ms\n"); //Debug to see how long response took
168
169 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void sht15_setup_io (void)

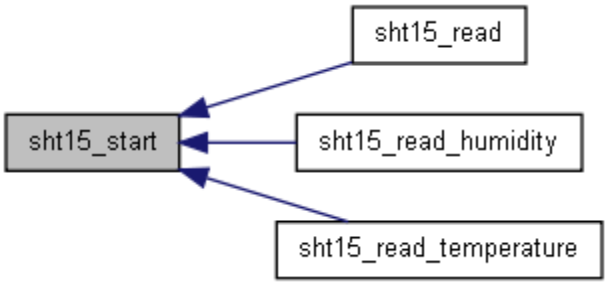
```
52 {
53     clear pin(sht15_sck_port, sht15_sck_pin);
54     clear pin(sht15_sda_pin, sht15_sda_pin);
55     make output(sht15_sda_port, sht15_sda_pin);
56     make output(sht15_sck_port, sht15_sck_pin);
57 }
```

void sht15_start (void)

```
173 {
174 /*    sht15 write sda();
175     set_pin(sht15_sda_port, sht15_sda_pin);     //SDA = 1;
176     set_pin(sht15_sck_port, sht15_sck_pin); //SCK = 1;
177
178     clear_pin(sht15_sda_port, sht15_sda_pin);   //SDA = 0;
179     clear_pin(sht15_sck_port, sht15_sck_pin); //SCK = 0;
```

```
180     set_pin(sht15_sck_port, sht15_sck_pin); // SCK = 1;
181     set_pin(sht15_sda_port, sht15_sda_pin);     //SDA = 1;
182     clear pin(sht15 sck port, sht15 sck pin); //SCK = 0;
183 */
184     sht15 write sda();
185
186     // initial state
187
188     clear pin(sht15_sck_port, sht15_sck_pin); //SCK = 0;
189     set pin(sht15_sda_port, sht15_sda_pin);     //SDA = 1;
190         delay us(10);
191
192     set pin(sht15_sck_port, sht15_sck_pin); // SCK = 1;
193         delay_us(10);
194
195     // sck is high, so lower the data ling
196     clear pin(sht15_sda_port, sht15_sda_pin);   //SDA = 0;
197         delay_us(10);
198
199     // low pulse SCK
200     clear pin(sht15_sck_port, sht15_sck_pin); //SCK = 0;
201         delay_us(10);
202     set pin(sht15 sck port, sht15 sck pin); // SCK = 1;
203         delay_us(10);
204
205     // now raise SDA while SCK is high
206     set pin(sht15_sda_port, sht15_sda_pin);     //SDA = 1;
207         delay us(10);
208
209     // return sck to normal state
210     clear pin(sht15_sck_port, sht15_sck_pin); //SCK = 0;
211         delay us(10);
212
213 }
```

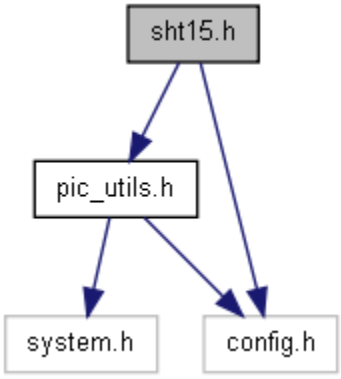
Here is the caller graph for this function:



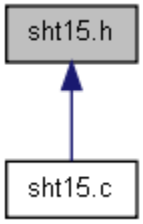
---

sht15.h File Reference

Support for SHT15 and SHT11 digital humidity sensors.
Include dependency graph for sht15.h:



This graph shows which files directly or indirectly include this file:



Functions

- uns16 sht15_fix_humidity (uns16 sensor_out)
- uns16 sht15_fix_humidity_l (uns8 sensor_out)
- uns16 sht15_fix_humidity_r (uns16 sensor_out)
- int16 sht15_fix_temperature_h (uns16 sensor_out)
- void sht15_read (void)
- uns16 sht15_read_byte16 (void)
- uns16 sht15_read_humidity (void)
- uns16 sht15_read_temperature (void)
- void sht15_send_byte (uns8 sht15_command)
- void sht15_setup_io (void)
- void sht15_start (void)

Detailed Description

Include the following in your config.h:

- - - - - - - - - - sht15 defines
- - - - - - - - - -

define sht15_sck_port PORTA define sht15_sck_pin 1 define sht15_sda_port PORTA define sht15_sda_pin 0

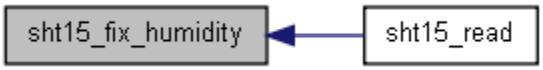
- - - - - - - - - -

Function Documentation

uns16 sht15_fix_humidity (uns16 *sensor_out*)

```
328 {
329
330 long result; // 16Bit unsigned for the result
331
332     if ( sensor_out <= 1712 ) {
333         result = 143 * sensor_out;
334         result = result - 8192; // result = a * sensor_out
335         //result < 512 ? result = 512; // check for underflow
336         //result = result - 8192; // result = result + b
337     } else {
338         result = 111 *sensor_out + 46288; // result = a * sensor_out
339         //result = result + 46288; // result = result + b
340         //result > 25600 ? result = 25600; // check for overflow (optional)
341     }
342     //8 MSB's are 0-100%RH integers, 8 LSB's are remainder
343     result = result / 4096;
344     return result;
345 }
```

Here is the caller graph for this function:



uns16 sht15_fix_humidity_l (uns8 *sensor_out*)

```
348 {
349
350 uns16 result; // 16Bit unsigned for the result
351
352     if ( sensor_out <= 107 ) {
353         result = 143 * sensor_out; // result = a * sensor_out
354         if (result < 512) { result = 512; } // check for underflow
355         result -= 512;
356     } else {
357         result = 111 * sensor_out; // result = a * sensor_out
358         result += 2893; // result = result + b
359         if (result > 25600) { result = 25600; } // check for overflow (optional)
360     }
361     //8 MSB's are 0-100%RH integers, 8 LSB's are remainder
362     result = result >> 8; // result = result / 256
363     return result;
364 }
```

Here is the caller graph for this function:

uns16 sht15_fix_humidity_r (uns16 *sensor_out*)

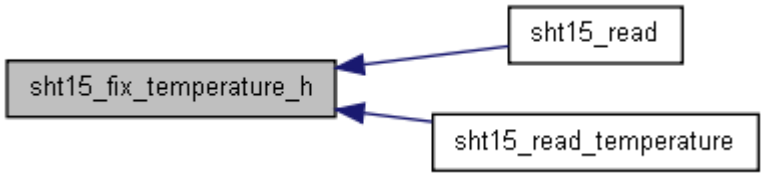
```
305                                                 {
306
307 int32 c1 = -40000000;
308 int32 c2 = 405000;
309 int32 c3 = 28;
```

```
310 int32 s;
311 int32 final;
312
313
314 s = sensor_out;
315
316 s = s * 10000000;
317
318 final = c1 + c2 * s + c3 * s * s;
319
320 final = final / 10000;
321
322 return final;
323 }
```

int16 sht15_fix_temperature_h (uns16 *sensor_out*)

```
367 {
368     int16 result;
369
370 //  result = -3963 + sensor_out;
371     result = -4001 + sensor_out;
372     return result;
373 }
```

Here is the caller graph for this function:

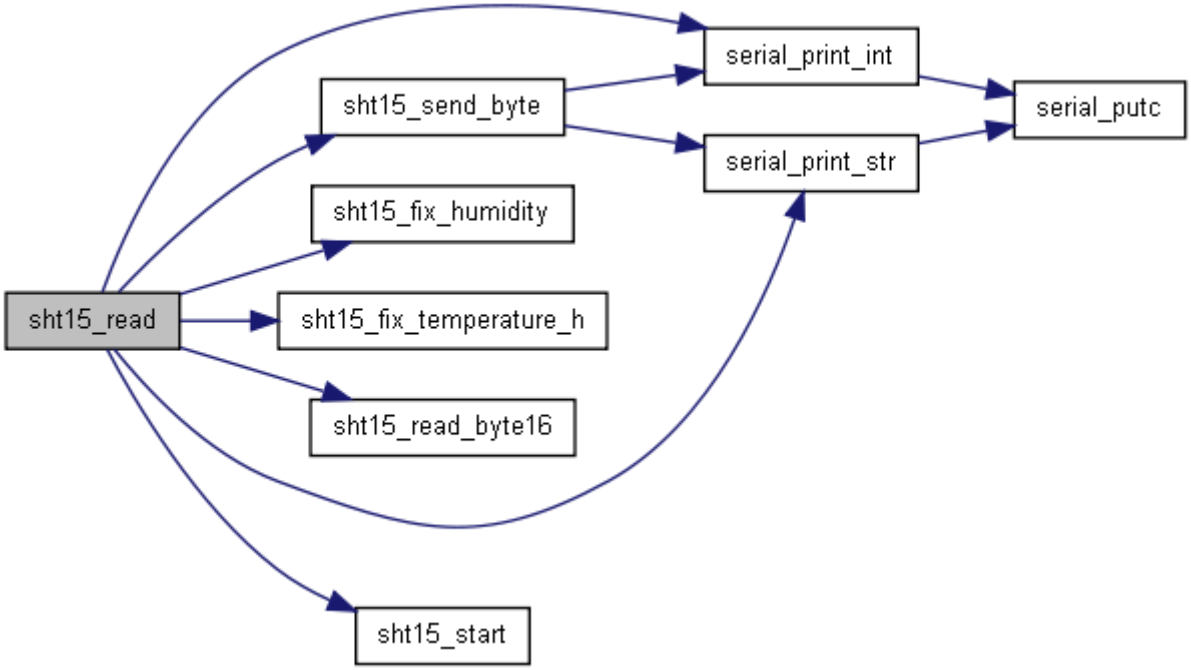


void sht15_read (void)

```
88 {
89     uns16 response;
90
91     //Issue command start
92     sht15_start();
93
94     //Now send command code
95     sht15_send_byte(CHECK_HUMD);
96
97     response = sht15_read_byte16();
98
99
100     serial_print_str("\nH=");
101     serial_print_int(response);
102     //response = response >> 4;
103     response = sht15_fix_humidity(response);
104     serial_print_str("\nH=");
105     serial_print_int(response);
106
107     sht15_start();
108     sht15_send_byte(CHECK_TEMP);
109     response = sht15_read_byte16(); //Listen for response from SHT15
110     serial_print_str("\nT=");
111     serial_print_int(response);//
112     response = sht15_fix_temperature_h(response);
113     serial_print_str("\nT=");
114     serial_print_int(response);
115
```

```
116 }
```

Here is the call graph for this function:

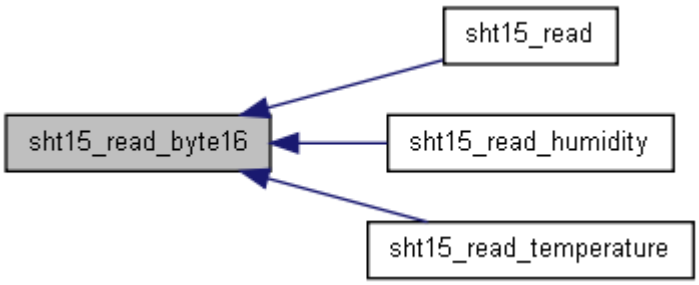


uns16 sht15_read_byte16 (void)

```
217 {
218     uns8 j;
219     uns16 in_byte;
220     uns8 crc;
221     //bit  my_store_gie = intcon.GIE;
222     //kill_interrupts();
223 
224     clear_pin(sht15_sck_port, sht15_sck_pin);
225 
226     sht15_read_sda();
227 
228     for(j = 0 ; j < 8 ; j++)
229     {
230         set_pin(sht15_sck_port, sht15_sck_pin);
231 delay_us(10);
232         in_byte = in_byte << 1;
233         in_byte.0 = test_pin(sht15_sda_port, sht15_sda_pin);
234 
235         clear_pin(sht15_sck_port, sht15_sck_pin);
236 delay_us(10);
237     }
238     delay_us(100);
239 
240     // send ack
241     sht15_write_sda();
242     clear_pin(sht15_sda_port, sht15_sda_pin);
243 
244         set_pin(sht15_sck_port, sht15_sck_pin);
245 delay_us(10);
246         clear_pin(sht15_sck_port, sht15_sck_pin);
247 delay_us(10);
248 
249     sht15_read_sda();
```

```
250     delay_us(100);
251
252
253     for(j = 0 ; j < 8 ; j++)
254     {
255         set_pin(sht15_sck_port, sht15_sck_pin);
256 delay_us(10);
257         in_byte = in_byte << 1;
258         in_byte.0 = test_pin(sht15_sda_port, sht15_sda_pin);
259
260         clear_pin(sht15 sck port, sht15 sck pin);
261 delay_us(10);
262     }
263
264     // send ack
265     sht15_write_sda();
266     clear_pin(sht15_sda_port, sht15_sda_pin);
267
268         set_pin(sht15_sck_port, sht15_sck_pin);
269 delay_us(10);
270         clear_pin(sht15_sck_port, sht15_sck_pin);
271 delay_us(10);
272
273     sht15_read_sda();
274     delay_us(100);
275
276
277     for(j = 0 ; j < 8 ; j++)
278     {
279         set_pin(sht15_sck_port, sht15_sck_pin);
280 delay_us(10);
281         crc = crc << 1;
282         crc.0 = test_pin(sht15_sda_port, sht15_sda_pin);
283
284         clear_pin(sht15_sck_port, sht15_sck_pin);
285 delay_us(10);
286     }
287
288     // send ack
289     sht15_write_sda();
290     clear_pin(sht15_sda_port, sht15_sda_pin);
291
292         set_pin(sht15_sck_port, sht15_sck_pin);
293 delay_us(10);
294         clear_pin(sht15_sck_port, sht15_sck_pin);
295 delay_us(10);
296
297     sht15_read_sda();
298     delay_us(100);
299
300     //intcon.GIE = my_store_gie;
301     return(in_byte);
302 }
```

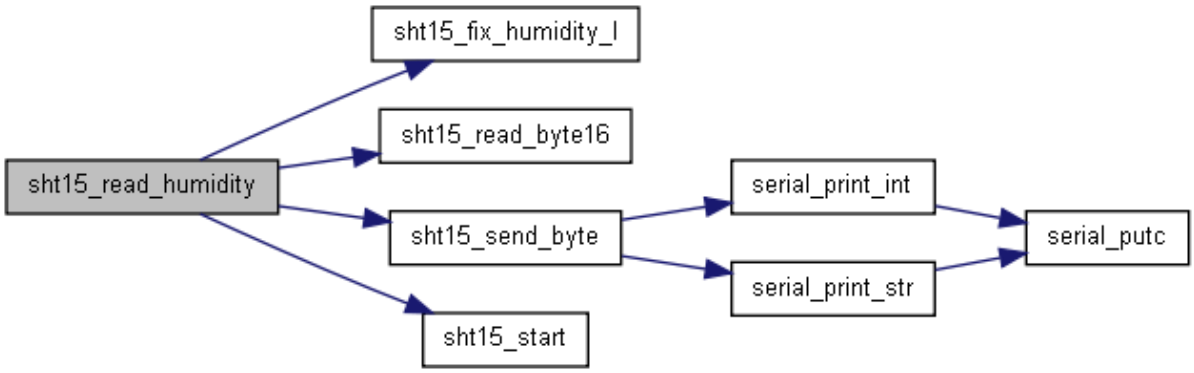
Here is the caller graph for this function:



uns16 sht15_read_humidity (void)

```
59                                  {
60
61      uns16 response;
62
63      sht15_start();
64      sht15_send_byte(CHECK_HUMD);
65      response = sht15_read_byte16();
66      response = response >> 4;
67      response = sht15_fix_humidity_l(response);
68      return response;
69  }
```

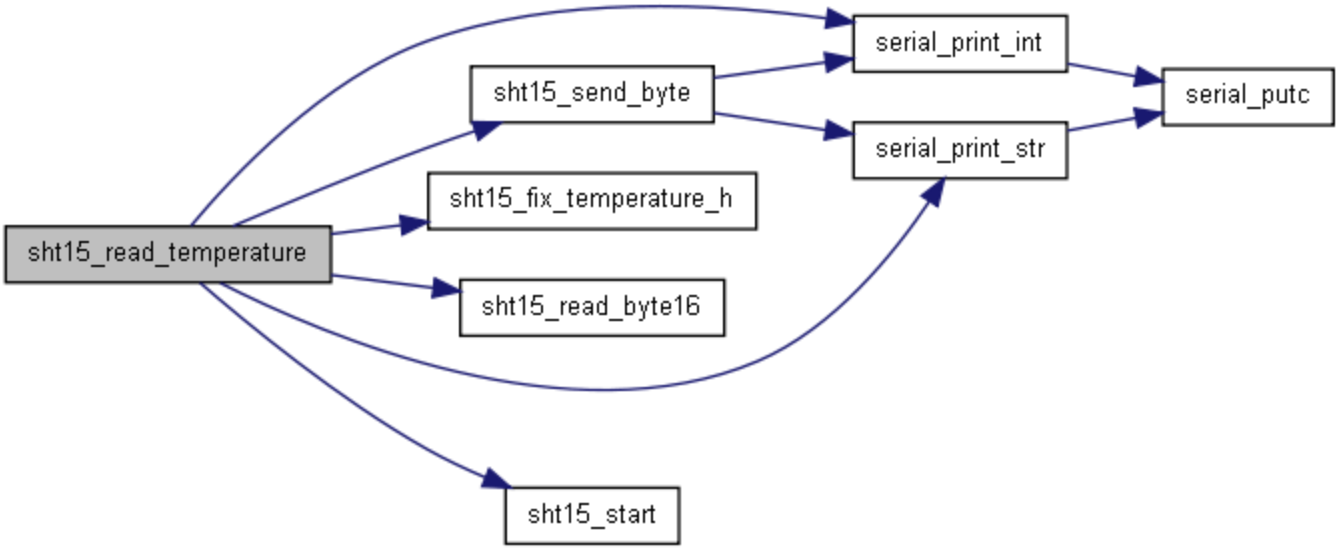
Here is the call graph for this function:



uns16 sht15_read_temperature (void)

```
71                                      {
72
73  uns16 response;
74
75      sht15_start();
76      sht15_send_byte(CHECK_TEMP);
77      response = sht15_read_byte16(); //Listen for response from SHT15
78      serial_print_str("\nT=");
79      serial_print_int(response);
80      response = sht15_fix_temperature_h(response);
81      serial_print_str("\nT=");
82      serial_print_int(response);
83
84  }
```

Here is the call graph for this function:

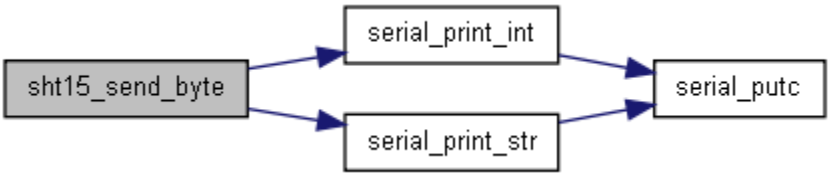


void sht15_send_byte (uns8 *sht15_command*)

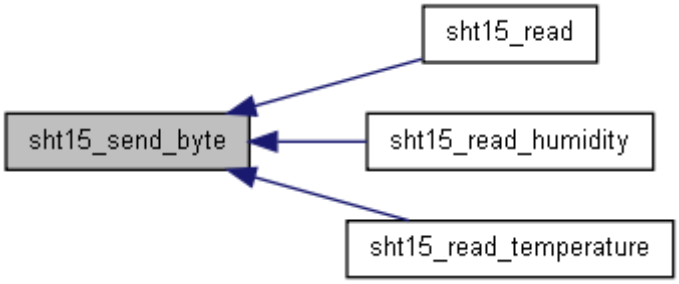
```
119 {
120     uns8 i;
121 
122 
123     sht15_write_sda();
124 
125 
126     clear_pin(sht15_sck_port, sht15_sck_pin);
127     for(i = 0 ; i < 8 ; i++)
128     {
129         delay_us(10);
130         change_pin(sht15_sda_port, sht15_sda_pin, sht15_command.7);
131         sht15_command = sht15_command << 1;
132         delay_us(10);
133         set_pin(sht15_sck_port, sht15_sck_pin);
134         delay_us(10);
135         clear_pin(sht15_sck_port, sht15_sck_pin);
136     }
137     delay_us(100);
138     //Wait for SHT15 to acknowledge.
139 //    clear_pin(sht15_sck_port, sht15_sck_pin);
140     sht15_read_sda();
141     // <<!!>> Fix following
142 //    while (test_pin(sht15_sda_port, sht15_sda_pin) == 1); //Wait for SHT to pull line low
143 
144     set_pin(sht15_sck_port, sht15_sck_pin);
145     // read ack here
146 
147     delay_us(10);
148     clear_pin(sht15_sck_port, sht15_sck_pin); //Falling edge of 9th clock
149 
150     while (test_pin(sht15_sda_port, sht15_sda_pin) == 0); //Wait for SHT to release line
151 
152     //Wait for SHT15 to pull SDA low to signal measurement completion.
153     //This can take up to 210ms for 14 bit measurements
154     i = 0;
155     while ((test_pin(sht15_sda_port, sht15_sda_pin) == 1)) //Wait for SHT to pull line low
156     {
157         i++;
158         if (i == 255) break;
159 
160         delay_ms(10);
```

```
161     }
162
163     //Debug info
164     i *= 10; //Convert to ms
165     serial_print_str("\nRt=");
166     serial_print_int(i);
167     serial_print_str("ms\n"); //Debug to see how long response took
168
169 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void sht15_setup_io (void)

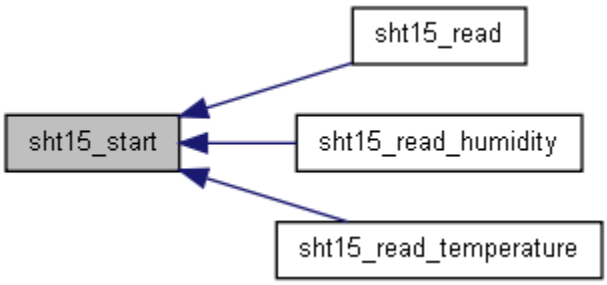
```
52 {
53     clear_pin(sht15_sck_port, sht15_sck_pin);
54     clear_pin(sht15_sda_pin, sht15_sda_pin);
55     make_output(sht15_sda_port, sht15_sda_pin);
56     make_output(sht15_sck_port, sht15_sck_pin);
57 }
```

void sht15_start (void)

```
173 {
174 /*    sht15_write_sda();
175     set_pin(sht15_sda_port, sht15_sda_pin);     //SDA = 1;
176     set_pin(sht15_sck_port, sht15_sck_pin); //SCK = 1;
177
178     clear_pin(sht15_sda_port, sht15_sda_pin);   //SDA = 0;
179     clear_pin(sht15_sck_port, sht15_sck_pin); //SCK = 0;
180     set_pin(sht15_sck_port, sht15_sck_pin); // SCK = 1;
181     set_pin(sht15_sda_port, sht15_sda_pin);     //SDA = 1;
182     clear_pin(sht15_sck_port, sht15_sck_pin); //SCK = 0;
183 */
184     sht15_write_sda();
185
186     // initial state
187
188     clear_pin(sht15_sck_port, sht15_sck_pin); //SCK = 0;
189     set_pin(sht15_sda_port, sht15_sda_pin);     //SDA = 1;
190         delay_us(10);
```

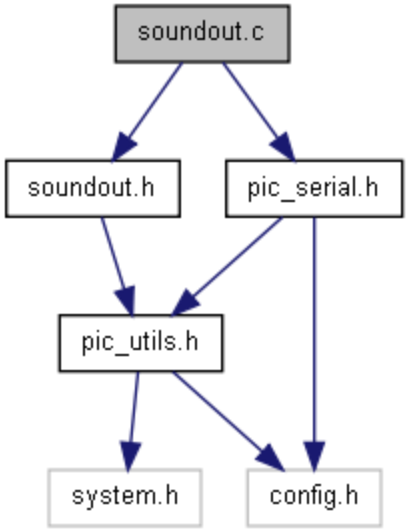
```
191
192     set_pin(sht15_sck_port, sht15_sck_pin); // SCK = 1;
193         delay_us(10);
194
195     // sck is high, so lower the data ling
196     clear_pin(sht15_sda_port, sht15_sda_pin);   //SDA = 0;
197         delay_us(10);
198
199     // low pulse SCK
200     clear_pin(sht15_sck_port, sht15_sck_pin); //SCK = 0;
201         delay_us(10);
202     set_pin(sht15_sck_port, sht15_sck_pin); // SCK = 1;
203         delay_us(10);
204
205     // now raise SDA while SCK is high
206     set_pin(sht15_sda_port, sht15_sda_pin);     //SDA = 1;
207         delay_us(10);
208
209     // return sck to normal state
210     clear_pin(sht15_sck_port, sht15_sck_pin); //SCK = 0;
211         delay_us(10);
212
213 }
```

Here is the caller graph for this function:



soundout.c File Reference

Include dependency graph for soundout.c:



Functions

- uns8 soundout_is_busy ()
- void soundout_play_pause ()
- void soundout_reset ()
- void soundout_send_data (uns16 data)
- void soundout_set_file_id (uns16 file_id)
- void soundout_set_volume (uns8 level)
- void soundout_setup_io ()
- void soundout_standby ()
- void soundout_stop ()
- void soundout_wake ()

Function Documentation

uns8 soundout_is_busy ()

```
123                             {
124     #ifdef soundout_busy_port
125         return test_pin(soundout_busy_port, soundout_busy_pin);
126     #else
127         return 0;
128     #endif
129 }
```

void soundout_play_pause ()

```
113                                 {
114     soundout_send_data(soundout_PLAY_PAUSE_CMD);
115 }
```

Here is the call graph for this function:

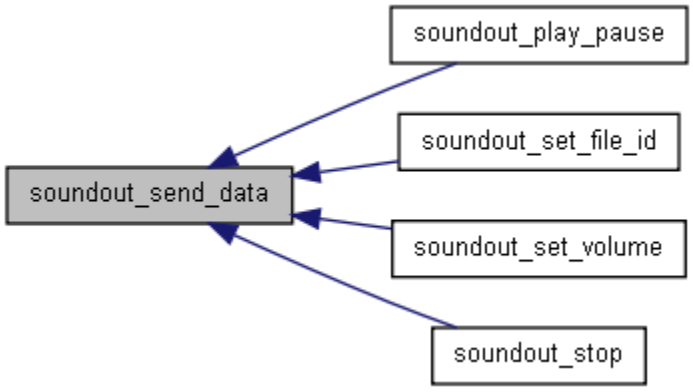
void soundout_reset ()

```
57                          {
58      #ifdef soundout_reset_port
59          clear_pin(soundout_reset_port, soundout_reset_pin);
60          delay_ms(10);   // minimum 5ms
61          set_pin(soundout_reset_port, soundout_reset_pin);
62      #endif
63  }
```

void soundout_send_data (uns16 *data*)

```
79                                      {
80
81      // Signal start
82      clear_pin(soundout_clk_port, soundout_clk_pin);
83      delay_ms(3);    // tSTART = 2ms
84
85      for (uns8 count = 0; count < 16; count++) {
86          if (data.15) {
87              set_pin(soundout_data_port, soundout_data_pin);
88          } else {
89              clear_pin(soundout_data_port, soundout_data_pin);
90          }
91          delay_us(2);    // tDS = 1us
92          set_pin(soundout_clk_port, soundout_clk_pin);
93          delay_us(220);  // tCH = 200us
94          clear_pin(soundout_clk_port, soundout_clk_pin);
95          delay_us(220); // tCL = 200us
96          data = data << 1;
97      }
98      // Signal end
99      set_pin(soundout_clk_port, soundout_clk_pin);
100      delay_ms(3);    // tSTOP = 2ms
101 }
```

Here is the caller graph for this function:



void soundout_set_file_id (uns16 *file_id*)

```
103                                         {
104
105     soundout_send_data(file_id);
106 }
```

Here is the call graph for this function:

void soundout_set_volume (uns8 *level*)

```
108                                         {
109     level = level & 0x07;
110     soundout_send_data(soundout_VOLUME_CMD + level);
111 }
```

Here is the call graph for this function:

void soundout_setup_io ()

```
41                           {
42
43     make_output(soundout_clk_port, soundout_clk_pin);
44     set_pin(soundout_clk_port, soundout_clk_pin);   // idle high
45
46     make_output(soundout_data_port, soundout_data_pin);
47     // don't care about data pin state
48     #ifdef soundout_reset_port
49         make_output(soundout_reset_port, soundout_reset_pin);
50         set_pin(soundout_reset_port, soundout_reset_pin);
51     #endif
52     #ifdef soundout_busy_port
53         make_input(soundout_busy_port, soundout_busy_pin);
54     #endif
55 }
```

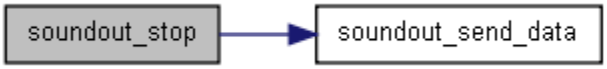
void soundout_standby ()

```
65                           {
66     #ifdef soundout_reset_port
67         clear_pin(soundout_reset_port, soundout_reset_pin);
68         // Need to wait for 1 second to go into standby
69         // seems a shame to hang around here waiting though...
70         // delay_s(1);
71     #endif
72 }
```

void soundout_stop ()

```
117                          {
118
119     soundout_send_data(soundout_STOP_CMD);
120
121 }
```

Here is the call graph for this function:



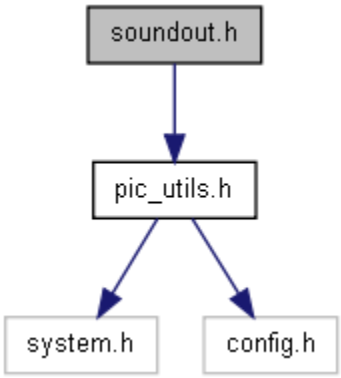
void soundout_wake ()

```
73                      {
74     #ifdef soundout_reset_port
75         set_pin(soundout_reset_port, soundout_reset_pin);
76     #endif
77 }
```

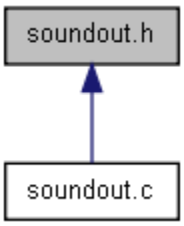
---

soundout.h File Reference

SoundOut MOD-1007 (Somo-14D) audio player interface.
Include dependency graph for soundout.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define soundout_PLAY_PAUSE_CMD 0xfffe
- #define soundout_STOP_CMD 0xffff
- #define soundout_VOLUME_CMD 0xfff0

Functions

- uns8 soundout_is_busy ()

- void soundout_play_pause ()
- void soundout_reset ()
- void soundout_set_file_id (uns16 file_id)
- void soundout_set_volume (uns8 level)
- void soundout_setup_io ()
- void soundout_standby ()
- void soundout_stop ()
- void soundout_wake ()

Detailed Description

Library for accessing the functionality of the WTV .ad4 audio player
Put the following into your config.h

- - - - - - - - - - - - - - - - - - - SoundOut defines
- - - - - - - - - - - - - - - - - - -

define soundout_clk_port PORTA define soundout_clk_pin 1
define soundout_data_port PORTA define soundout_data_pin 2
don't define these if you don't want to use them
define soundout_reset_port PORTA define soundout_reset_pin 3
define soundout_busy_port PORTA define soundout_busy_pin 4

- - - - - - - - - - - - - - - - - - -

Define Documentation

#define soundout_PLAY_PAUSE_CMD 0xfffe

#define soundout_STOP_CMD 0xffff

#define soundout_VOLUME_CMD 0xfff0

Function Documentation

uns8 soundout_is_busy ()

```
123                             {
124     #ifdef soundout_busy_port
125         return test_pin(soundout_busy_port, soundout_busy_pin);
126     #else
127         return 0;
128     #endif
129 }
```

void soundout_play_pause ()

```
113                             {
114     soundout_send_data(soundout_PLAY_PAUSE_CMD);
```

```
115 }
```

Here is the call graph for this function:

void soundout_reset ()

```
57                          {
58     #ifdef soundout_reset_port
59         clear_pin(soundout_reset_port, soundout_reset_pin);
60         delay_ms(10);   // minimum 5ms
61         set_pin(soundout_reset_port, soundout_reset_pin);
62     #endif
63 }
```

void soundout_set_file_id (uns16 *file_id*)

```
103                                             {
104
105     soundout_send_data(file_id);
106 }
```

Here is the call graph for this function:

void soundout_set_volume (uns8 *level*)

```
108                                         {
109     level = level & 0x07;
110     soundout_send_data(soundout_VOLUME_CMD + level);
111 }
```

Here is the call graph for this function:

void soundout_setup_io ()

```
41                          {
42
43     make_output(soundout_clk_port, soundout_clk_pin);
44     set_pin(soundout_clk_port, soundout_clk_pin);   // idle high
45
46     make_output(soundout_data_port, soundout_data_pin);
47     // don't care about data pin state
48     #ifdef soundout_reset_port
49         make_output(soundout_reset_port, soundout_reset_pin);
50         set_pin(soundout_reset_port, soundout_reset_pin);
51     #endif
52     #ifdef soundout_busy_port
```

```
53          make_input(soundout_busy_port, soundout_busy_pin);
54      #endif
55  }
```

void soundout_standby ()

```
65                          {
66      #ifdef soundout_reset_port
67          clear_pin(soundout_reset_port, soundout_reset_pin);
68          // Need to wait for 1 second to go into standby
69          // seems a shame to hang around here waiting though...
70          // delay_s(1);
71      #endif
72  }
```

void soundout_stop ()

```
117                         {
118
119      soundout_send_data(soundout_STOP_CMD);
120
121 }
```

Here is the call graph for this function:

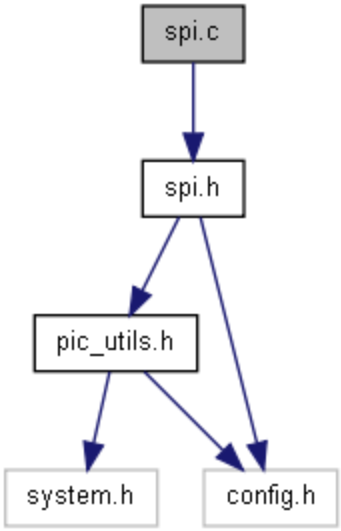
void soundout_wake ()

```
73                          {
74      #ifdef soundout_reset_port
75          set_pin(soundout_reset_port, soundout_reset_pin);
76      #endif
77  }
```

---

spi.c File Reference

Include dependency graph for spi.c:



Functions

- void spi_pulse_0 ()
- *SPI test routine.* void spi_pulse_1 ()
- *SPI test routine.* void spi_setup ()
- *Setup ports and pins for SPI output.* void spi_write (uns8 data)
- *Send a byte of data using software spi.* void spi_write_lsb (uns8 data)
- *Send a byte of data using software spi.* void spi_write_sure (uns8 data)

SPI write for Sure devices.

Function Documentation

void spi_pulse_0 ()

```
90                      {
91          change_pin(spi_data_port, spi_data_pin, 0);
92          clear_pin(spi_clk_port, spi_clk_pin);
93          set_pin(spi_clk_port, spi_clk_pin);
94  }
```

void spi_pulse_1 ()

```
96                      {
97          change_pin(spi_data_port, spi_data_pin, 1);
98          clear_pin(spi_clk_port, spi_clk_pin);
99          set_pin(spi_clk_port, spi_clk_pin); /
```

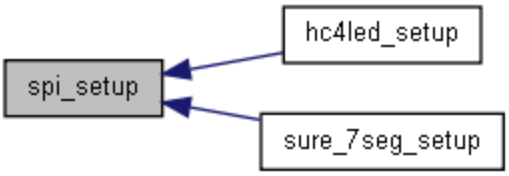
100 }

void spi_setup ()

Setup ports and pins for SPI output

```
69                  {
70      clear_bit(tris_array[spi_clk_port - PORTA], spi_clk_pin);   // output
71      clear_bit(tris_array[spi_data_port - PORTA], spi_data_pin);   // output
72
73  }
```

Here is the caller graph for this function:



void spi_write (uns8 *data*)

Sends a byte of data MSB first, data only changes on clock low

```
41                              {
42
43 uns8 count;
44     for (count = 0; count < 8; count++) {
45         clear_pin(spi_clk_port, spi_clk_pin);   // set to low
46         change_pin(spi_data_port, spi_data_pin, data.7);
47         data = data << 1;
48         set_pin(spi_clk_port, spi_clk_pin); // set to low
49     }
50
51 }
```

Here is the caller graph for this function:

void spi_write_lsb (uns8 *data*)

Sends a byte of data LSB first, data only changes on clock low

```
55                                  {
56
57 uns8 count;
58
59
60     for (count = 0; count < 8; count++) {
61         clear_pin(spi_clk_port, spi_clk_pin);   // set to low
62         change_pin(spi_data_port, spi_data_pin, data.0);
63         data = data >> 1;
64         set_pin(spi_clk_port, spi_clk_pin); // set to low
65     }
66
67 }
```

void spi_write_sure (uns8 *data*)

SPI write byte for Sure devices. Sure devices do things a little differently. Data goes LSB first but data changes on clock high.

```
75                                   {
76
77 uns8 count;
78 uns8 data_in;
79
80     data_in = data;
81     for (count = 0; count < 8; count++) {
82         change_pin(spi_data_port, spi_data_pin, data_in.0);
83         clear_pin(spi_clk_port, spi_clk_pin);
84         set_pin(spi_clk_port, spi_clk_pin);
```

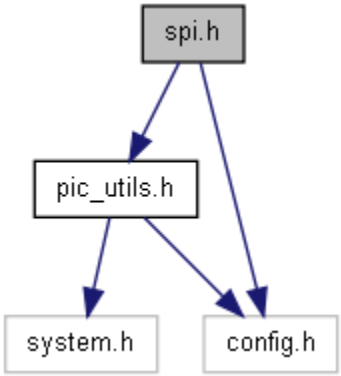
```
85          data_in = data_in >> 1;
86      }
87  }
```

Here is the caller graph for this function:

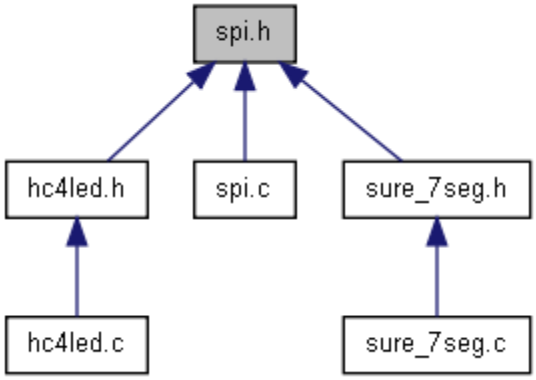
---

spi.h File Reference

Outputs SPI-like interfaces (clock+data).
Include dependency graph for spi.h:



This graph shows which files directly or indirectly include this file:



Functions

- void spi_pulse_0 ()
- *SPI test routine.* void spi_pulse_1 ()
- *SPI test routine.* void spi_setup ()
- *Setup ports and pins for SPI output.* void spi_write (uns8 data)
- *Send a byte of data using software spi.* void spi_write_lsb (uns8 data)
- *Send a byte of data using software spi.* void spi_write_sure (uns8 data)

Detailed Description

Covers standard SPI-like interfaces (clock + data) and Sure Electronics displays which are a little different

---

Function Documentation

void spi_pulse_0 ()

```
90                      {
91          change pin(spi_data_port, spi_data_pin, 0);
92          clear pin(spi_clk_port, spi_clk_pin);
93          set pin(spi_clk_port, spi_clk_pin);
94  }
```

void spi_pulse_1 ()

```
96                      {
97          change pin(spi_data_port, spi_data_pin, 1);
98          clear pin(spi_clk_port, spi_clk_pin);
99          set pin(spi_clk_port, spi_clk_pin); /
```

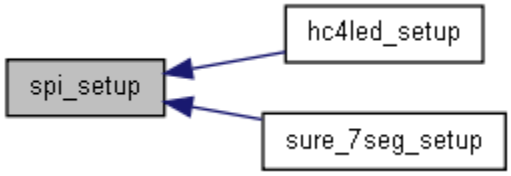
100 }

void spi_setup ()

Setup ports and pins for SPI output

```
69                  {
70      clear bit(tris array[spi clk port - PORTA], spi clk pin);   // output
71      clear_bit(tris_array[spi_data_port - PORTA], spi_data_pin);   // output
72
73  }
```

Here is the caller graph for this function:



void spi_write (uns8 *data*)

Sends a byte of data MSB first, data only changes on clock low

```
41                              {
42
43  uns8 count;
44      for (count = 0; count < 8; count++) {
45          clear pin(spi_clk_port, spi_clk_pin);    // set to low
46          change pin(spi_data_port, spi_data_pin, data.7);
47          data = data << 1;
48          set pin(spi_clk_port, spi_clk_pin); // set to low
49      }
50
51  }
```

Here is the caller graph for this function:

void spi_write_lsb (uns8 *data*)

Sends a byte of data LSB first, data only changes on clock low

```
55                                  {
56
57 uns8 count;
58
59
60     for (count = 0; count < 8; count++) {
61         clear pin(spi_clk_port, spi_clk_pin);   // set to low
62         change pin(spi data port, spi data pin, data.0);
63         data = data >> 1;
64         set pin(spi_clk_port, spi_clk_pin); // set to low
65     }
66
67 }
```

void spi_write_sure (uns8 *data*)

SPI write byte for Sure devices. Sure devices do things a little differently. Data goes LSB first but data changes on clock high.

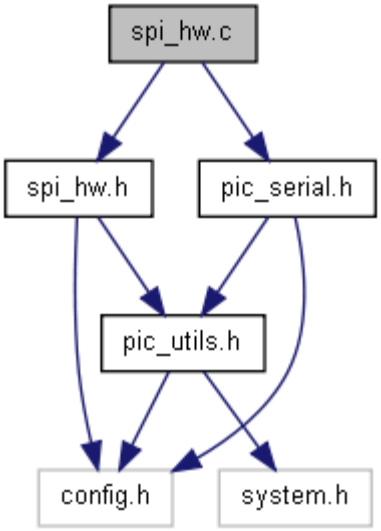
```
75                                  {
76
77 uns8 count;
78 uns8 data_in;
79
80     data_in = data;
81     for (count = 0; count < 8; count++) {
82         change pin(spi_data_port, spi_data_pin, data_in.0);
83         clear pin(spi_clk_port, spi_clk_pin);
84         set pin(spi_clk_port, spi_clk_pin);
85         data_in = data_in >> 1;
86     }
87 }
```

Here is the caller graph for this function:

---

spi_hw.c File Reference

Include dependency graph for spi_hw.c:



Functions

- void spi_hw_init ()
- uns8 spi_hw_master_receive ()
- void spi_hw_master_transmit (uns8 data)
- void spi_hw_setup_io ()

Function Documentation

void spi_hw_init ()

```
99                      {
100
101     #ifdef SPI_HW_CLK_IDLE_IS_HIGH
102         set_bit(sspcon1, CKP);
103     #else
104         // for 0,0 mode - mrf24j40 needs this
105         clear_bit(sspcon1, CKP);
106     #endif
107     #ifdef SPI_HW_MASTER_MODE
108
109         set_bit(sspstat, SMP);  // from tim
110     #else
111         clear_bit(sspstat, SMP);    // smp must be cleared in slave mode
112     #endif
113
114     #ifdef SPI_HW_TRANSMIT_ON_IDLE_TO_ACT
115         clear_bit(sspstat, CKE);
116     #else
117         // need this for mrf24j40
118         // for 0,0 mode
119         set_bit(sspstat, CKE);
120     #endif
121     set_bit(sspcon1, SSPEN);    // enable mssp serial port
122 }
```

uns8 spi_hw_master_receive ()

```
141                                 {
142
143     spi_hw_master_transmit(0x00);   // dummy transmit to get something back
144     return sspbuf;
145
146 }
```

Here is the call graph for this function:

void spi_hw_master_transmit (uns8 *data*)

```
126                                         {
127
128     clear_bit(pir1, SSPIF);
129     // Try and send, if we have something already sending, try again
130     do {
131         clear bit(sspcon1, WCOL);
132         sspbuf = data;
133     } while (test_bit(sspcon1, WCOL));
134
135     // Wait here while we transmit
136     // Note that we will hang here if we don't have SPI setup properly
137     // or we're running under the simulator
138     while (!test_bit(pir1, SSPIF)) {}
139 }
```

Here is the caller graph for this function:

void spi_hw_setup_io ()

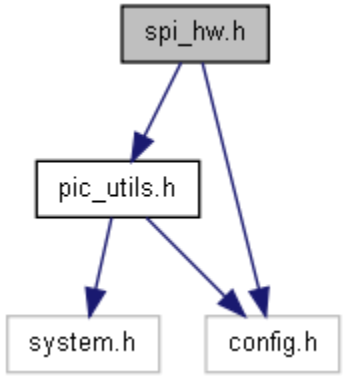
```
40                          {
41
42 #ifndef _PIC18F14K50
43     make_output(PORTC, 5);
44     make_input(PORTC, 4);
45 #else
46 #endif
47
48 #ifdef SPI_HW_MASTER_MODE
49
50     #ifndef _PIC18F14K50
51     #else
52         make_output(PORTC, 3);
53     #endif
54
55     clear_bit(sspcon1, SSPM3);
56     clear_bit(sspcon1, SSPM2);
57     #ifdef SPI_HW_MASTER_CLOCK_TMR2_DIV_2
58         set_bit(sspcon1, SSPM1);
59         set_bit(sspcon1, SSPM0);
60     #endif
61     #ifdef SPI_HW_MASTER_CLOCK_FOSC_DIV_64
62         set_bit  (sspcon1, SSPM1);
63         clear_bit(sspcon1, SSPM0);
64     #endif
65     #ifdef SPI_HW_MASTER_CLOCK_FOSC_DIV_16
```

```
66          clear_bit(sspcon1, SSPM1);
67          set_bit  (sspcon1, SSPM0);
68      #endif
69      #ifdef SPI_HW_MASTER_CLOCK_FOSC_DIV_4
70          clear_bit(sspcon1, SSPM1);
71          clear_bit(sspcon1, SSPM0);
72      #endif
73  #else
74      // Slave mode
75      #ifndef _PIC18F14K50
76          make_input(PORTC, 3);   // sck
77      #else
78          make_input(PORTB, 6);   // sck
79          make_output(PORTC, 7);  // sdo
80      #endif
81      clear_bit(sspcon1, SSPM3);
82      set_bit  (sspcon1, SSPM2);
83      clear_bit(sspcon1, SSPM1);
84      #ifdef SPI_HW_USE_SS
85          #ifndef _PIC18F14K50
86              make_input(PORTA, 5);   // SS
87          #else
88              make_input(PORTC, 6);   // SS
89          #endif
90          clear_bit(sspcon1, SSPM0);
91      #else
92          set_bit(sspcon1, SSPM0);
93      #endif
94  #endif
95
96  }
```

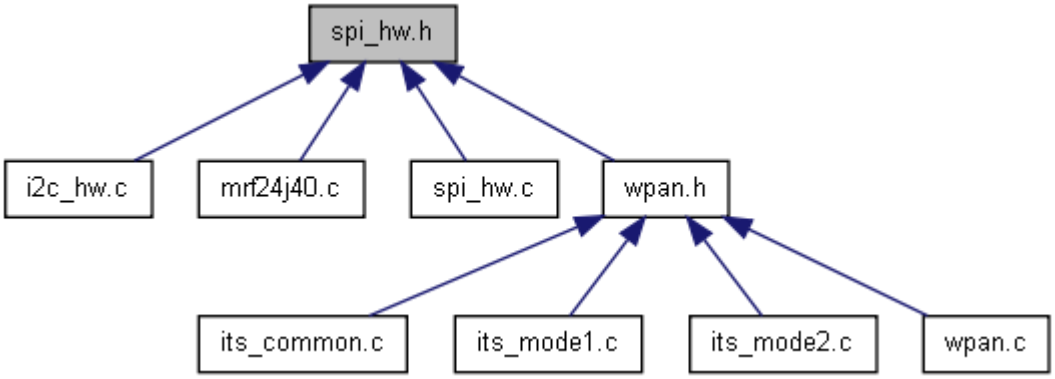
---

spi_hw.h File Reference

Include dependency graph for spi_hw.h:



This graph shows which files directly or indirectly include this file:



Functions

- void spi_hw_init ()
- uns8 spi_hw_master_receive ()
- void spi_hw_master_transmit (uns8 data)
- void spi_hw_setup_io ()

Detailed Description

Serial Peripheral Interface (HW) routines

Put the following into your config.h

- - - - - - - - - - - - - - - - - - - spi_hw defines
- - - - - - - - - - - - - - - - - - -

define SPI_HW_MASTER_MODE or define SPI_HW_SLAVE_MODE

In slave mode, we can use ss: define SPI_HW_USE_SS

In master mode, we need to define clock define SPI_HW_MASTER_CLOCK_TMR2_DIV_2 define SPI_HW_MASTER_CLOCK_FOSC_DIV_64 define SPI_HW_MASTER_CLOCK_FOSC_DIV_16 define SPI_HW_MASTER_CLOCK_FOSC_DIV_4

define SPI_HW_TRANSMIT_ON_ACT_TO_IDLE or define SPI_HW_TRANSMIT_ON_IDLE_TO_ACT

define SPI_HW_CLK_IDLE_IS_HIGH or define SPI_HW_CLK_IDLE_IS_LOW

- - - - - - - - - - - - - - - - - - -

Function Documentation

void spi_hw_init ()

```
82                      {
83     // for 0,0 mode
84     clear bit(sspcon1, CKP);
85
86     set_bit(sspstat, CKE);
87     set_bit(sspstat, SMP);  // from tim
88     set_bit(sspcon1, SSPEN);    // enable mssp serial port
```

```
89 }
```

uns8 spi_hw_master_receive ()

```
141                                 {
142
143     spi_hw_master_transmit(0x00);   // dummy transmit to get something back
144     return sspbuf;
145
146 }
```

Here is the call graph for this function:

void spi_hw_master_transmit (uns8 *data*)

```
126                                         {
127
128     clear_bit(pir1, SSPIF);
129     // Try and send, if we have something already sending, try again
130     do {
131         clear_bit(sspcon1, WCOL);
132         sspbuf = data;
133     } while (test_bit(sspcon1, WCOL));
134
135     // Wait here while we transmit
136     // Note that we will hang here if we don't have SPI setup properly
137     // or we're running under the simulator
138     while (!test_bit(pir1, SSPIF)) {}
139 }
```

Here is the caller graph for this function:

void spi_hw_setup_io ()

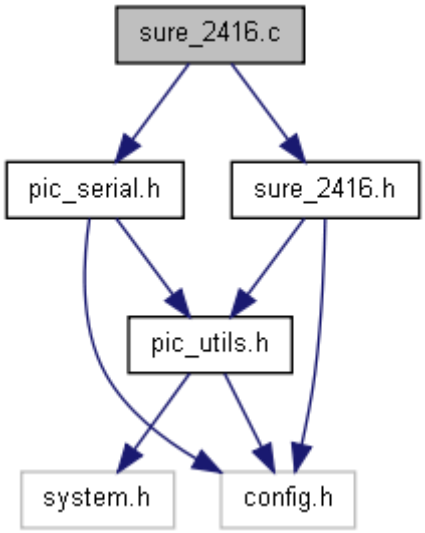
```
40                          {
41
42 make_output(PORTC, 5);
43 make_input(PORTC, 4);
44
45 #ifdef SPI_HW_MASTER_MODE
46     make_output(PORTC, 3);
47     clear_bit(sspcon1, 3);
48     clear_bit(sspcon1, 2);
49     #ifdef SPI_HW_MASTER_CLOCK_TMR2_DIV_2
50         set_bit(sspcon1, 1);
51         set_bit(sspcon1, 0);
52     #endif
53     #ifdef SPI_HW_MASTER_CLOCK_FOSC_DIV_64
54         set_bit  (sspcon1, 1);
55         clear_bit(sspcon1, 0);
56     #endif
57     #ifdef SPI_HW_MASTER_CLOCK_FOSC_DIV_16
58         clear_bit(sspcon1, 1);
```

```
59          set_bit  (sspcon1, 0);
60      #endif
61      #ifdef SPI HW MASTER CLOCK FOSC DIV 4
62          clear_bit(sspcon1, 1);
63          clear_bit(sspcon1, 0);
64      #endif
65  #else
66      // Slave mode
67      make input(PORTC, 3);   // sck
68      clear_bit(sspcon1, 3);
69      set bit  (sspcon1, 2);
70      clear_bit(sspcon1, 1);
71      #ifdef SPI_HW_USE_SS
72          make input(PORTA, 5);   // SS
73          clear bit(sspcon1, 0);
74      #else
75          set_bit(sspcon1, 0);
76      #endif
77  #endif
78
79  }
```

---

sure_2416.c File Reference

Sure 2416 led matrix display routines.
Include dependency graph for sure_2416.c:



Functions

- void sure_2416_fill (uns8 colour)
- void sure_2416_fill2 (uns8 colour)
- void sure_2416_init ()
- void sure_2416_send_command (uns8 command)
- void sure_2416_set_brightness (uns8 brightness)
- void sure_2416_set_pixel (uns8 x, uns8 y, uns8 colour)
- void sure_2416_setup ()

- void sure_2416_write (uns8 mem_addr, uns8 data)

Detailed Description

Function Documentation

void sure_2416_fill (uns8 *colour*)

```
308                                     {
309     uns8 mem_address;
310     uns8 fill;
311 
312     if (colour) {
313         fill = 0b00001111;
314     } else {
315         fill = 0b00000000;
316     }
317 
318     for(mem_address = 0 ; mem_address < 96 ; mem_address++) {
319         sure_2416_write(mem_address, fill);
320     }
321 }
```

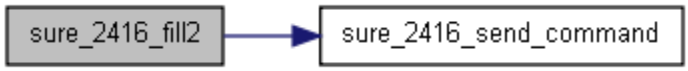
Here is the call graph for this function:

void sure_2416_fill2 (uns8 *colour*)

```
323                                     {
324 
325 uns16 count;
326     sure_2416_send_command(SURE_2416_CMD_LEDS_OFF);
327     clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
328 
329     // send WR command
330 
331     // send 1
332     set_pin   (sure_2416_data_port, sure_2416_data_pin);
333     // pulse wr
334     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
335     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
336 
337     // send 0
338     clear_pin   (sure_2416_data_port, sure_2416_data_pin);
339     // pulse wr
340     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
341     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
342 
343     // send 1
344     set_pin   (sure_2416_data_port, sure_2416_data_pin);
345     // pulse wr
346     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
347     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
348 
349     // send mem address of zero
350     clear_pin(sure_2416_data_port, sure_2416_data_pin);
```

```
351
352     // write mem addr, bits 6 -> 0
353     for(count = 0 ; count < 7 ; count++) {
354
355         //change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(mem_addr, 6));
356         // pulse wr
357         clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
358         set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
359         // shift mem addr along
360
361     }
362     if (colour) {
363         set_pin(sure_2416_data_port, sure_2416_data_pin);
364     } else {
365         clear_pin(sure_2416_data_port, sure_2416_data_pin);
366     }
367     // we need to toggle 384 times
368
369     for(count = 0 ; count < 384 ; count++) {
370         // pulse wr
371         clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
372         set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
373         // shift mem addr along
374
375     }
376     // reset CS
377
378     set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
379     sure_2416_send_command(SURE_2416_CMD_LEDS_ON);
380
381
382 }
```

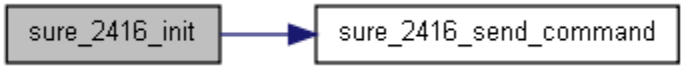
Here is the call graph for this function:



void sure_2416_init ()

```
58                      {
59
60     sure_2416_send_command(SURE_2416_CMD_SYS_DISABLE);
61     sure_2416_send_command(SURE_2416_CMD_PMOS_16_COMMON);   // Correct hardware layout for
the board
62
63     sure_2416_send_command(SURE_2416_CMD_CLK_MASTER_MODE);  // We are the master
64     sure_2416_send_command(SURE_2416_CMD_SYS_ENABLE);
65     sure_2416_send_command(SURE_2416_CMD_LEDS_ON); //led on
66 }
```

Here is the call graph for this function:

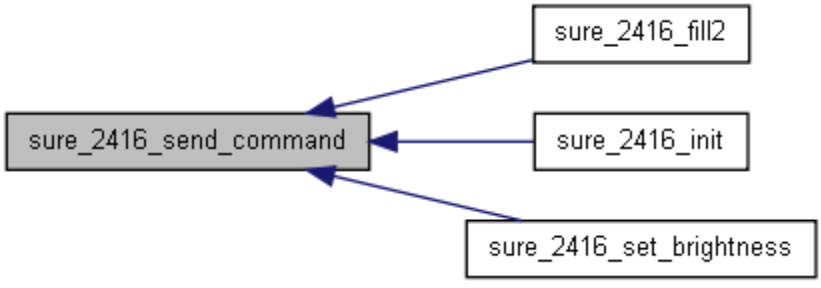


void sure_2416_send_command (uns8 *command*)

```
68                                          {
69
70 uns8 count;
71
```

```
72      clear pin(sure_2416_cs1_port, sure_2416_cs1_pin);
73
74      // send  command
75      // send 1
76      set pin   (sure_2416_data_port, sure_2416_data_pin);
77      // pulse wr
78      clear pin(sure_2416_wr_port, sure_2416_wr_pin);
79      set pin   (sure_2416_wr_port, sure_2416_wr_pin);
80
81      // send 0
82      clear pin   (sure 2416 data port, sure 2416 data pin);
83      // pulse wr
84      clear pin(sure_2416_wr_port, sure_2416_wr_pin);
85      set pin   (sure_2416_wr_port, sure_2416_wr_pin);
86
87      // send 0
88      clear pin   (sure_2416_data_port, sure_2416_data_pin);
89      // pulse wr
90      clear pin(sure_2416_wr_port, sure_2416_wr_pin);
91      set pin   (sure_2416_wr_port, sure_2416_wr_pin);
92
93      // command bits 7 - 0
94      for(count = 0 ; count < 8 ; count++) {
95
96          if (test_bit(command, 7)) {
97              set pin(sure_2416_data_port, sure_2416_data_pin);
98          } else {
99              clear pin(sure 2416 data port, sure 2416 data pin);
100          }
101          //change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(command, 7));
102          // pulse wr
103          clear pin(sure 2416 wr port, sure 2416 wr pin);
104          set pin   (sure_2416_wr_port, sure_2416_wr_pin);
105          // shift mem addr along
106          command = command << 1;
107      }
108
109      // the don't care pulse
110
111          clear pin(sure 2416 wr port, sure 2416 wr pin);
112          set pin   (sure_2416_wr_port, sure_2416_wr_pin);
113
114      // reset CS??
115
116 // don't think we need this
117 //   set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
118 //   delay_ms(1);
119 //   clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
120
121     set pin(sure_2416_cs1_port, sure_2416_cs1_pin);
122 }
```

Here is the caller graph for this function:



void sure_2416_set_brightness (uns8 *brightness*)

```
188                                                 {
189     // allows level 0 - 15
190     sure_2416_send_command(0b10100000 | (brightness & 0b00001111));
191 }
```

Here is the call graph for this function:

void sure_2416_set_pixel (uns8 *x*, uns8 *y*, uns8 *colour*)

```
193                                                         {
194
195 uns8 common, panel, led_in_panel, inverted_x, out, mem_addr, bit_in_mem_addr, count, data;
196
197     // first calculate memory address
198
199     // y location on panels is top left based
200
201     common = 15 - y;
202
203 /* Previous calculations:
204     panel = x / 8;  // which panel of the three is it that we need to change?
205     led_in_panel = x - (panel * 8);
206     inverted_x = 7 - led_in_panel;
207     out = panel * 8 + led_in_panel; //inverted_x;
208     mem_addr = out * 4 + common / 4;
209 */
210     bit_in_mem_addr = common & 0b00000011;
211
212
213     mem_addr = x * 4 + common / 4;
214
215
216     clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
217
218     // send WR command
219
220     // send 1
221     set_pin   (sure_2416_data_port, sure_2416_data_pin);
222     // pulse wr
223     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
224     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
225
226     // send 0
227     clear_pin  (sure_2416_data_port, sure_2416_data_pin);
228     // pulse wr
229     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
230     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
231
232     // send 1
233     set_pin   (sure_2416_data_port, sure_2416_data_pin);
234     // pulse wr
235     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
236     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
237
238     // write mem addr, bits 6 -> 0
239     for(count = 0 ; count < 7 ; count++) {
240         if (test_bit(mem_addr, 6)) {
241             set_pin(sure_2416_data_port, sure_2416_data_pin);
242         } else {
243             clear_pin(sure_2416_data_port, sure_2416_data_pin);
```

```
244         }
245
246         //change pin var(sure 2416 data port, sure 2416 data pin, test bit(mem addr, 6));
247         // pulse rd
248         clear pin(sure_2416_wr_port, sure_2416_wr_pin);
249         set pin  (sure_2416_wr_port, sure_2416_wr_pin);
250
251         // shift mem addr along
252         mem_addr = mem_addr << 1;
253     }
254
255     // Retrieve 4 bits
256     // read clocked out on falling edge of RD
257
258     make input(sure 2416 data port, sure 2416 data pin);
259
260     for(count = 0 ; count < 4 ; count++) {
261         // pulse rd
262         clear pin(sure_2416_rd_port, sure_2416_rd_pin);
263
264         data = data >> 1;
265         data.3 = test pin(sure_2416_data_port, sure_2416_data_pin);
266
267         set pin  (sure_2416_rd_port, sure_2416_rd_pin);
268
269     }
270
271     make output(sure 2416 data port, sure 2416 data pin);
272
273     // now we have the data, we need to change the bit
274     if (colour) {
275         set bit(data, bit in mem addr);
276     } else {
277         clear_bit(data, bit_in_mem_addr);
278     }
279
280     // Now write it back out again
281
282     // write data, bits 0 -> 3 (different from mem addr format)
283     for(count = 0 ; count < 4 ; count++) {
284         //change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(data, 0));
285         if (test_bit(data, 0)) {
286             set pin(sure_2416_data_port, sure_2416_data_pin);
287         } else {
288             clear pin(sure_2416_data_port, sure_2416_data_pin);
289         }
290
291         // pulse wr
292         clear pin(sure 2416 wr port, sure 2416 wr pin);
293         set pin  (sure_2416_wr_port, sure_2416_wr_pin);
294         // shift data along
295         data = data >> 1;
296     }
297
298
299     // reset CS
300     // don't think this is necessary
301 //  set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
302 //  clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
303     set pin  (sure_2416_cs1_port, sure_2416_cs1_pin);
304
305
306 }
```

void sure_2416_setup ()

```
43                      {
44
```

```
45      //serial_print_str("Setting up 2416\n");
46
47      make_output(sure_2416_cs1_port, sure_2416_cs1_pin);
48      make_output(sure_2416_data_port, sure_2416_data_pin);
49      make_output(sure_2416_wr_port, sure_2416_wr_pin);
50      make_output(sure_2416_rd_port, sure_2416_rd_pin);
51
52      set_pin(sure_2416_wr_port, sure_2416_wr_pin);
53      set_pin(sure_2416_rd_port, sure_2416_rd_pin);
54      set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
55
56  }
```

void sure_2416_write (uns8 *mem_addr*, uns8 *data*)

```
124                                                 {
125
126 uns8 count;
127
128     clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
129
130     // send WR command
131
132     // send 1
133     set_pin   (sure_2416_data_port, sure_2416_data_pin);
134     // pulse wr
135     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
136     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
137
138     // send 0
139     clear_pin   (sure_2416_data_port, sure_2416_data_pin);
140     // pulse wr
141     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
142     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
143
144     // send 1
145     set_pin   (sure_2416_data_port, sure_2416_data_pin);
146     // pulse wr
147     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
148     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
149
150     // write mem addr, bits 6 -> 0
151     for(count = 0 ; count < 7 ; count++) {
152         if (test_bit(mem_addr, 6)) {
153             set_pin(sure_2416_data_port, sure_2416_data_pin);
154         } else {
155             clear_pin(sure_2416_data_port, sure_2416_data_pin);
156         }
157
158         //change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(mem_addr, 6));
159         // pulse wr
160         clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
161         set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
162         // shift mem addr along
163         mem_addr = mem_addr << 1;
164     }
165
166     // write data, bits 0 -> 3 (different from mem addr format)
167     for(count = 0 ; count < 4 ; count++) {
168         change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(data, 0));
169         // pulse wr
170         clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
171         set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
172         // shift mem addr along
173         data = data >> 1;
174     }
175     // reset CS
176
```

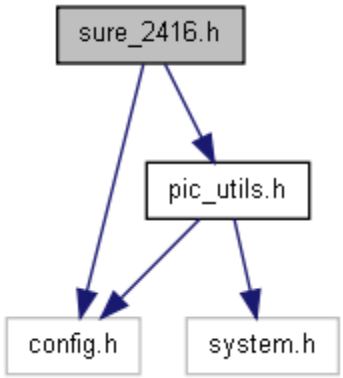
```
177 //  set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
178 //  delay_ms(1);
179 //  clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
180     set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
181
182
183 }
```

Here is the caller graph for this function:

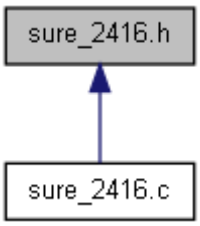
---

sure_2416.h File Reference

Sure 2416 led matrix display routines.
Include dependency graph for sure_2416.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define SURE_2416_CMD_BLINK_OFF 0b00001000
- #define SURE_2416_CMD_BLINK_ON 0b00001001
- #define SURE_2416_CMD_CLK_MASTER_MODE 0b00010100
- #define SURE_2416_CMD_CLK_SLAVE_MODE 0b00010000
- #define SURE_2416_CMD_CLK_SOURCE_EXT 0b00011100
- #define SURE_2416_CMD_CLK_SOURCE_INT_RC 0b00011000
- #define SURE_2416_CMD_LEDS_OFF 0b00000010

- #define SURE_2416_CMD_LEDS_ON 0b00000011
- #define SURE_2416_CMD_NMOS_16_COMMON 0b00100100
- #define SURE_2416_CMD_NMOS_8_COMMON 0b00100000
- #define SURE_2416_CMD_PMOS_16_COMMON 0b00101100
- #define SURE_2416_CMD_PMOS_8_COMMON 0b00101000
- #define SURE_2416_CMD_SYS_DISABLE 0b00000000
- #define SURE_2416_CMD_SYS_ENABLE 0b00000001

Functions

- void sure_2416_clear ()
- void sure_2416_fill (uns8 colour)
- void sure_2416_fill2 (uns8 colour)
- uns8 sure_2416_get_pixel (uns8 x, uns8 y)
- void sure_2416_horizontal_line (uns8 x, uns8 y, uns8 length, uns8 colour)
- void sure_2416_init ()
- void sure_2416_send_command (uns8 command)
- void sure_2416_set_brightness (uns8 brightness)
- void sure_2416_set_pixel (uns8 x, uns8 y, uns8 colour)
- void sure_2416_setup ()
- void sure_2416_vertical_line (uns8 x, uns8 y, uns8 length, uns8 colour)
- void sure_2416_write (uns8 mem_addr, uns8 data)

Detailed Description

Define Documentation

#define SURE_2416_CMD_BLINK_OFF 0b00001000

#define SURE_2416_CMD_BLINK_ON 0b00001001

#define SURE_2416_CMD_CLK_MASTER_MODE 0b00010100

#define SURE_2416_CMD_CLK_SLAVE_MODE 0b00010000

#define SURE_2416_CMD_CLK_SOURCE_EXT 0b00011100

#define SURE_2416_CMD_CLK_SOURCE_INT_RC 0b00011000

#define SURE_2416_CMD_LEDS_OFF 0b00000010

#define SURE_2416_CMD_LEDS_ON 0b00000011

#define SURE_2416_CMD_NMOS_16_COMMON 0b00100100

#define SURE_2416_CMD_NMOS_8_COMMON 0b00100000

#define SURE_2416_CMD_PMOS_16_COMMON 0b00101100

#define SURE_2416_CMD_PMOS_8_COMMON 0b00101000

#define SURE_2416_CMD_SYS_DISABLE 0b00000000

#define SURE_2416_CMD_SYS_ENABLE 0b00000001

Function Documentation

void sure_2416_clear ()

void sure_2416_fill (uns8 *colour*)

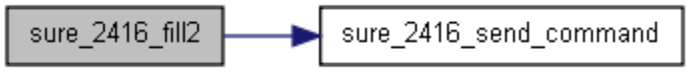
```
308                                 {
309     uns8 mem_address;
310     uns8 fill;
311 
312     if (colour) {
313         fill = 0b00001111;
314     } else {
315         fill = 0b00000000;
316     }
317 
318     for(mem_address = 0 ; mem_address < 96 ; mem_address++) {
319         sure_2416_write(mem_address, fill);
320     }
321 }
```

Here is the call graph for this function:

void sure_2416_fill2 (uns8 *colour*)

```
323                                     {
324
325 uns16 count;
326     sure_2416_send_command(SURE_2416_CMD_LEDS_OFF);
327     clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
328
329     // send WR command
330
331     // send 1
332     set_pin   (sure_2416_data_port, sure_2416_data_pin);
333     // pulse wr
334     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
335     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
336
337     // send 0
338     clear_pin   (sure_2416_data_port, sure_2416_data_pin);
339     // pulse wr
340     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
341     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
342
343     // send 1
344     set_pin   (sure_2416_data_port, sure_2416_data_pin);
345     // pulse wr
346     clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
347     set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
348
349     // send mem address of zero
350     clear_pin(sure_2416_data_port, sure_2416_data_pin);
351
352     // write mem addr, bits 6 -> 0
353     for(count = 0 ; count < 7 ; count++) {
354
355         //change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(mem_addr, 6));
356         // pulse wr
357         clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
358         set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
359         // shift mem addr along
360
361     }
362     if (colour) {
363         set_pin(sure_2416_data_port, sure_2416_data_pin);
364     } else {
365         clear_pin(sure_2416_data_port, sure_2416_data_pin);
366     }
367     // we need to toggle 384 times
368
369     for(count = 0 ; count < 384 ; count++) {
370         // pulse wr
371         clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
372         set_pin   (sure_2416_wr_port, sure_2416_wr_pin);
373         // shift mem addr along
374
375     }
376     // reset CS
377
378     set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
379     sure_2416_send_command(SURE_2416_CMD_LEDS_ON);
380
381
382 }
```

Here is the call graph for this function:



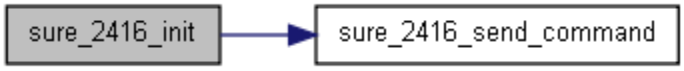
uns8 sure_2416_get_pixel (uns8 *x*, uns8 *y*)

void sure_2416_horizontal_line (uns8 *x*, uns8 *y*, uns8 *length*, uns8 *colour*)

void sure_2416_init ()

```
58                      {
59
60      sure_2416_send_command(SURE_2416_CMD_SYS_DISABLE);
61      sure_2416_send_command(SURE_2416_CMD_PMOS_16_COMMON);   // Correct hardware layout for
the board
62
63      sure_2416_send_command(SURE_2416_CMD_CLK_MASTER_MODE);  // We are the master
64      sure_2416_send_command(SURE_2416_CMD_SYS_ENABLE);
65      sure_2416_send_command(SURE_2416_CMD_LEDS_ON); //led on
66  }
```

Here is the call graph for this function:

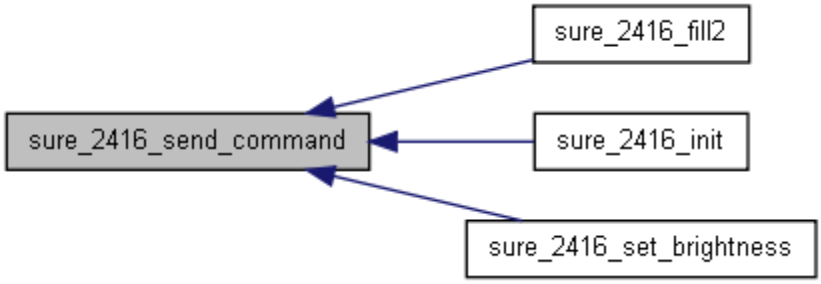


void sure_2416_send_command (uns8 *command*)

```
68                                          {
69
70  uns8 count;
71
72      clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
73
74      // send  command
75      // send 1
76      set_pin  (sure_2416_data_port, sure_2416_data_pin);
77      // pulse wr
78      clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
79      set_pin  (sure_2416_wr_port, sure_2416_wr_pin);
80
81      // send 0
82      clear_pin  (sure_2416_data_port, sure_2416_data_pin);
83      // pulse wr
84      clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
85      set_pin  (sure_2416_wr_port, sure_2416_wr_pin);
86
87      // send 0
88      clear_pin  (sure_2416_data_port, sure_2416_data_pin);
89      // pulse wr
90      clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
91      set_pin  (sure_2416_wr_port, sure_2416_wr_pin);
92
93      // command bits 7 - 0
94      for(count = 0 ; count < 8 ; count++) {
95
96          if (test_bit(command, 7)) {
97              set_pin(sure_2416_data_port, sure_2416_data_pin);
98          } else {
99              clear_pin(sure_2416_data_port, sure_2416_data_pin);
100         }
101         //change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(command, 7));
102         // pulse wr
103         clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
104         set_pin  (sure_2416_wr_port, sure_2416_wr_pin);
105         // shift mem addr along
106         command = command << 1;
107     }
```

```
108
109     // the don't care pulse
110
111         clear_pin(sure_2416_wr_port, sure_2416_wr_pin);
112         set_pin  (sure_2416_wr_port, sure_2416_wr_pin);
113
114     // reset CS??
115
116 // don't think we need this
117 //  set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
118 //  delay_ms(1);
119 //  clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
120
121     set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
122 }
```

Here is the caller graph for this function:



void sure_2416_set_brightness (uns8 *brightness*)

```
188                                                 {
189     // allows level 0 - 15
190     sure_2416_send_command(0b10100000 | (brightness & 0b00001111));
191 }
```

Here is the call graph for this function:

void sure_2416_set_pixel (uns8 *x*, uns8 *y*, uns8 *colour*)

```
193                                                         {
194
195 uns8 common, panel, led_in_panel, inverted_x, out, mem_addr, bit_in_mem_addr, count, data;
196
197     // first calculate memory address
198
199     // y location on panels is top left based
200
201     common = 15 - y;
202
203 /* Previous calculations:
204     panel = x / 8;  // which panel of the three is it that we need to change?
205     led_in_panel = x - (panel * 8);
206     inverted_x = 7 - led_in_panel;
207     out = panel * 8 + led_in_panel; //inverted_x;
208     mem addr = out * 4 + common / 4;
209 */
210     bit_in_mem_addr = common & 0b00000011;
211
```

```
212
213     mem_addr = x * 4 + common / 4;
214
215
216     clear pin(sure_2416_cs1_port, sure_2416_cs1_pin);
217
218     // send WR command
219
220     // send 1
221     set pin   (sure_2416_data_port, sure_2416_data_pin);
222     // pulse wr
223     clear pin(sure_2416_wr_port, sure_2416_wr_pin);
224     set pin   (sure_2416_wr_port, sure_2416_wr_pin);
225
226     // send 0
227     clear pin   (sure_2416_data_port, sure_2416_data_pin);
228     // pulse wr
229     clear pin(sure_2416_wr_port, sure_2416_wr_pin);
230     set pin   (sure_2416_wr_port, sure_2416_wr_pin);
231
232     // send 1
233     set pin   (sure_2416_data_port, sure_2416_data_pin);
234     // pulse wr
235     clear pin(sure_2416_wr_port, sure_2416_wr_pin);
236     set pin   (sure_2416_wr_port, sure_2416_wr_pin);
237
238     // write mem addr, bits 6 -> 0
239     for(count = 0 ; count < 7 ; count++) {
240         if (test_bit(mem_addr, 6)) {
241             set pin(sure_2416_data_port, sure_2416_data_pin);
242         } else {
243             clear pin(sure 2416 data port, sure 2416 data pin);
244         }
245
246         //change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(mem_addr, 6));
247         // pulse rd
248         clear pin(sure_2416_wr_port, sure_2416_wr_pin);
249         set pin   (sure_2416_wr_port, sure_2416_wr_pin);
250
251         // shift mem addr along
252         mem_addr = mem_addr << 1;
253     }
254
255     // Retrieve 4 bits
256     // read clocked out on falling edge of RD
257
258     make input(sure_2416_data_port, sure_2416_data_pin);
259
260     for(count = 0 ; count < 4 ; count++) {
261         // pulse rd
262         clear pin(sure_2416_rd_port, sure_2416_rd_pin);
263
264         data = data >> 1;
265         data.3 = test pin(sure_2416_data_port, sure_2416_data_pin);
266
267         set pin   (sure_2416_rd_port, sure_2416_rd_pin);
268
269     }
270
271     make output(sure_2416_data_port, sure_2416_data_pin);
272
273     // now we have the data, we need to change the bit
274     if (colour) {
275         set_bit(data, bit_in_mem_addr);
276     } else {
277         clear_bit(data, bit_in_mem_addr);
278     }
279
280     // Now write it back out again
281
282     // write data, bits 0 -> 3 (different from mem addr format)
```

```
283     for(count = 0 ; count < 4 ; count++) {
284         //change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(data, 0));
285         if (test bit(data, 0)) {
286             set pin(sure_2416_data_port, sure_2416_data_pin);
287         } else {
288             clear pin(sure_2416_data_port, sure_2416_data_pin);
289         }
290 
291         // pulse wr
292         clear pin(sure_2416_wr_port, sure_2416_wr_pin);
293         set pin  (sure 2416 wr port, sure 2416 wr pin);
294         // shift data along
295         data = data >> 1;
296     }
297 
298 
299     // reset CS
300     // don't think this is necessary
301 //  set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
302 //  clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
303     set pin  (sure_2416_cs1_port, sure_2416_cs1_pin);
304 
305 
306 }
```

void sure_2416_setup ()

```
43                          {
44 
45     //serial_print_str("Setting up 2416\n");
46 
47     make output(sure_2416_cs1_port, sure_2416_cs1_pin);
48     make output(sure_2416_data_port, sure_2416_data_pin);
49     make output(sure_2416_wr_port, sure_2416_wr_pin);
50     make output(sure 2416 rd port, sure 2416 rd pin);
51 
52     set pin(sure_2416_wr_port, sure_2416_wr_pin);
53     set pin(sure_2416_rd_port, sure_2416_rd_pin);
54     set pin(sure_2416_cs1_port, sure_2416_cs1_pin);
55 
56 }
```

void sure_2416_vertical_line (uns8 *x*, uns8 *y*, uns8 *length*, uns8 *colour*)

void sure_2416_write (uns8 *mem_addr*, uns8 *data*)

```
124                                                 {
125 
126 uns8 count;
127 
128     clear pin(sure_2416_cs1_port, sure_2416_cs1_pin);
129 
130     // send WR command
131 
132     // send 1
133     set pin  (sure_2416_data_port, sure_2416_data_pin);
134     // pulse wr
135     clear pin(sure_2416_wr_port, sure_2416_wr_pin);
136     set pin  (sure_2416_wr_port, sure_2416_wr_pin);
137 
138     // send 0
139     clear pin  (sure_2416_data_port, sure_2416_data_pin);
140     // pulse wr
141     clear pin(sure_2416_wr_port, sure_2416_wr_pin);
```

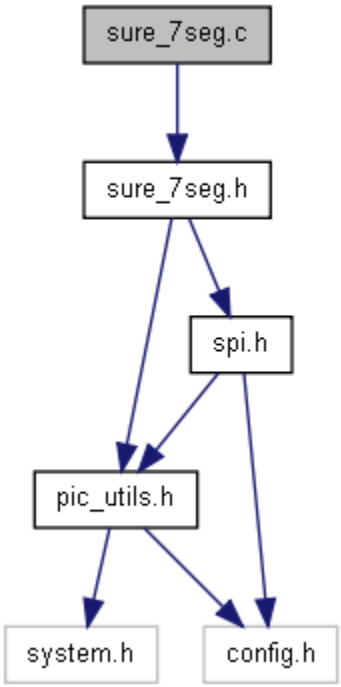
```
142     set pin   (sure_2416_wr_port, sure_2416_wr_pin);
143
144     // send 1
145     set pin   (sure_2416_data_port, sure_2416_data_pin);
146     // pulse wr
147     clear pin(sure_2416_wr_port, sure_2416_wr_pin);
148     set pin   (sure_2416_wr_port, sure_2416_wr_pin);
149
150     // write mem addr, bits 6 -> 0
151     for(count = 0 ; count < 7 ; count++) {
152         if (test bit(mem addr, 6)) {
153             set pin(sure_2416_data_port, sure_2416_data_pin);
154         } else {
155             clear pin(sure_2416_data_port, sure_2416_data_pin);
156         }
157
158         //change_pin_var(sure_2416_data_port, sure_2416_data_pin, test_bit(mem_addr, 6));
159         // pulse wr
160         clear pin(sure_2416_wr_port, sure_2416_wr_pin);
161         set pin   (sure_2416_wr_port, sure_2416_wr_pin);
162         // shift mem addr along
163         mem_addr = mem_addr << 1;
164     }
165
166     // write data, bits 0 -> 3 (different from mem addr format)
167     for(count = 0 ; count < 4 ; count++) {
168         change pin var(sure_2416_data_port, sure_2416_data_pin, test_bit(data, 0));
169         // pulse wr
170         clear pin(sure_2416_wr_port, sure_2416_wr_pin);
171         set pin   (sure_2416_wr_port, sure_2416_wr_pin);
172         // shift mem addr along
173         data = data >> 1;
174     }
175     // reset CS
176
177 //  set_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
178 //  delay_ms(1);
179 //  clear_pin(sure_2416_cs1_port, sure_2416_cs1_pin);
180     set pin(sure_2416_cs1_port, sure_2416_cs1_pin);
181
182
183 }
```

Here is the caller graph for this function:

---

sure_7seg.c File Reference

Routines to talk to Sure electronics seven segment displays.
Include dependency graph for sure_7seg.c:



Functions

- uns8 sure_7seg_convert (uns8 digit)
- void sure_7seg_setup ()
- *Setup ports and pins to communicate to Sure display.* void sure_7seg_write_str (char *data)

Display ASCII string to 7 segment displays.

Detailed Description

Function Documentation

uns8 sure_7seg_convert (uns8 *digit*)

```
48                                      {
49     switch (digit) {
50     case ' ': return 0;
51     case '0': return 0xfc;
52     case '1': return 0x60;
53     case '2': return 0xda;
54     case '3': return 0xf2;
55     case '4': return 0x66;
56     case '5': return 0xb6;
57     case '6': return 0xbe;
58     case '7': return 0xe0;
59     case '8': return 0xfe;
60     case '9': return 0xf6;
61     }
62 }
```

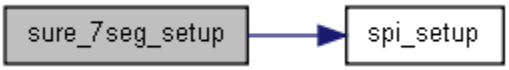
Here is the caller graph for this function:

void sure_7seg_setup ()

Set up ports and pins as appropriate to communicate via SPI to Sure 7 segment displays

```
43                          {
44     spi_setup();
45 }
```

Here is the call graph for this function:



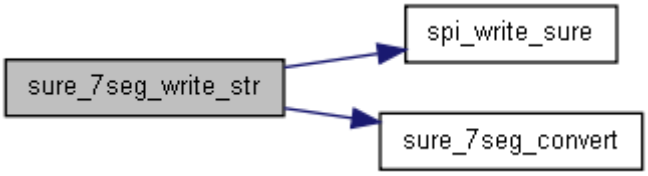
void sure_7seg_write_str (char * *data*)

Converts ASCII to the appropriate magic characters to display on a Sure 7 segment display. Only numbers 0-9 and space are implented for now.

To create your own characters, add together: 0x40 -------- | | 0x02 | | 0x20 | 0x01 | |------| | | 0x04 | | 0x10 | | -------- 0x08

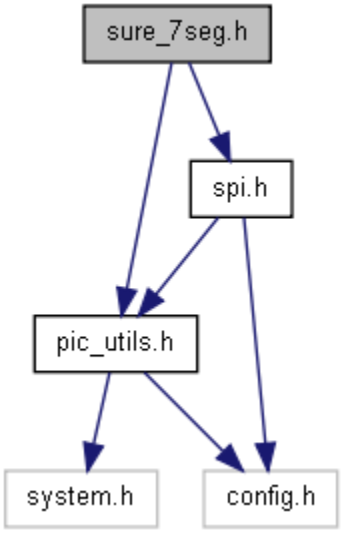
```
64                                      {
65
66 uns8 count, digit;
67 char converted[5];
68
69     count = 0;
70     do {
71         digit = data[count];
72         converted[count] = sure_7seg_convert(digit);
73         count++;
74        } while (count < 4);
75     spi_write_sure(converted[3]);
76     spi_write_sure(converted[2]);
77     spi_write_sure(converted[1]);
78     spi_write_sure(converted[0]);
79
80     clear_pin(spi_clk_port, spi_clk_pin);   // set to low
81     set_pin(spi_clk_port, spi_clk_pin); // set to high
82
83 }
```

Here is the call graph for this function:

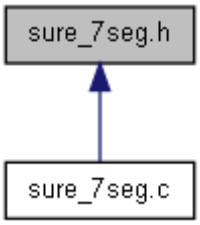


sure_7seg.h File Reference

Routines to talk to Sure electronics seven segment displays.
Include dependency graph for sure_7seg.h:



This graph shows which files directly or indirectly include this file:



Functions

- void sure_7seg_setup ()
- *Setup ports and pins to communicate to Sure display.* void sure_7seg_write_str (char *data)

Display ASCII string to 7 segment displays.

Detailed Description

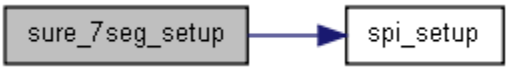
Function Documentation

void sure_7seg_setup ()

Set up ports and pins as appropriate to communicate via SPI to Sure 7 segment displays

```
43                          {
44      spi_setup();
45  }
```

Here is the call graph for this function:



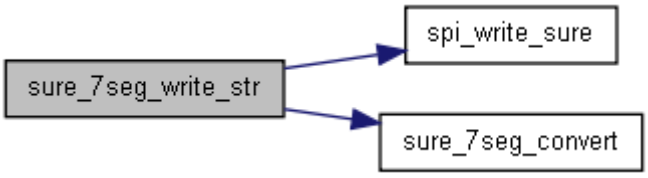
void sure_7seg_write_str (char * *data*)

Converts ASCII to the appropriate magic characters to display on a Sure 7 segment display. Only numbers 0-9 and space are implented for now.

To create your own characters, add together: 0x40 -------- | | 0x02 | | 0x20 | 0x01 | |------| | | 0x04 | | 0x10 | | -------- 0x08

```
64                                      {
65
66 uns8 count, digit;
67 char converted[5];
68
69     count = 0;
70     do {
71         digit = data[count];
72         converted[count] = sure_7seg_convert(digit);
73         count++;
74        } while (count < 4);
75     spi_write_sure(converted[3]);
76     spi_write_sure(converted[2]);
77     spi_write_sure(converted[1]);
78     spi_write_sure(converted[0]);
79
80     clear_pin(spi_clk_port, spi_clk_pin);   // set to low
81     set_pin(spi_clk_port, spi_clk_pin); // set to high
82
83 }
```

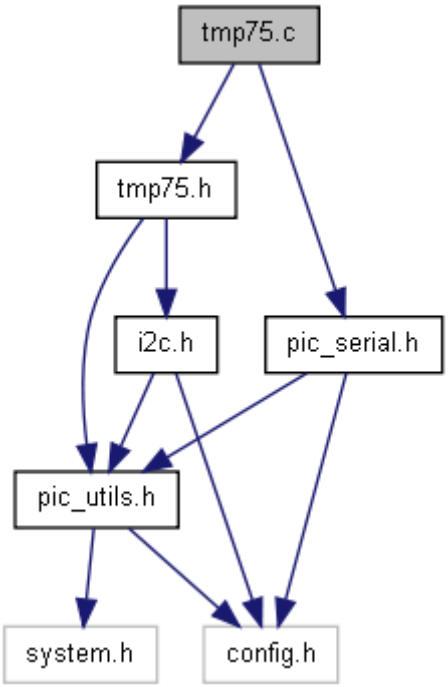
Here is the call graph for this function:



---

tmp75.c File Reference

Routines to access TMP75 temperature sensor.
Include dependency graph for tmp75.c:



Functions

- void tmp75_convert_temp (uns8 addr)
- *Start temperature conversion on tmp75.* uns8 tmp75_get_config (uns8 addr)
- *Get tmp75 config register.* uns16 tmp75_get_temp (uns8 addr)
- *Read temperature from tmp75.* uns8 tmp75_read (uns8 addr, uns8 pointer)
- uns16 tmp75_read_16bit (uns8 addr, uns8 pointer)
- void tmp75_set_config (uns8 addr, uns8 config)
- *Set tmp75 config register.* void tmp75_setup ()
- uns8 tmp75_write (uns8 addr, uns8 pointer, uns8 data)

Detailed Description

Function Documentation

void tmp75_convert_temp (uns8 *addr*)

This routine starts the temperature conversion in the tmp75. Issue this command before actually reading the temperature.

needs fixing

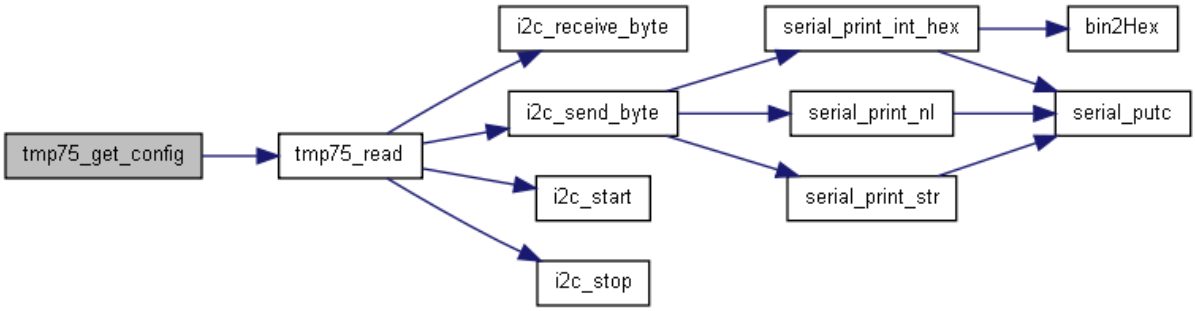
```
107                                     {
109 //  i2c_start();
110 //  i2c_send_byte(0x90 + addr);
111 //  i2c_send_byte(ds1631_start_convert);
112 //  i2c_stop();
113 }
```

uns8 tmp75_get_config (uns8 *addr*)

Gets the tmp75 config register (memory location 0x01)

```
103 {
104     return tmp75_read(addr, TMP75_CONFIG_REGISTER);
105 }
```

Here is the call graph for this function:

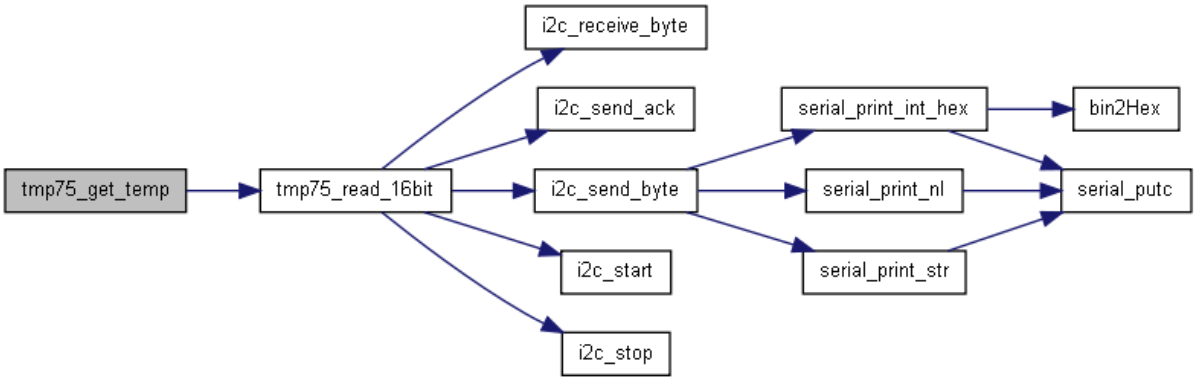


uns16 tmp75_get_temp (uns8 *addr*)

Returns 16bit raw temperature register from tmp75.

```
116 {
117     return tmp75_read_16bit(addr, TMP75_TEMP_REGISTER);
118 
119 }
```

Here is the call graph for this function:

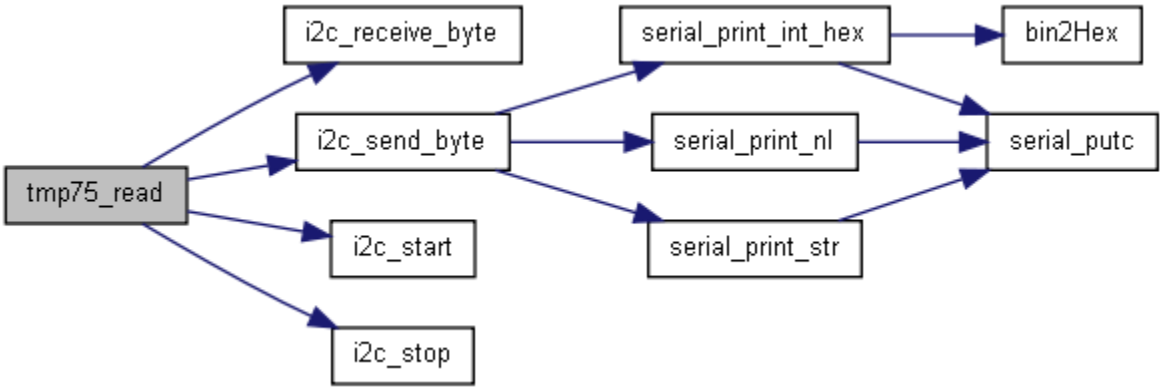


uns8 tmp75_read (uns8 *addr*, uns8 *pointer*)

```
42                                          {
43 uns8 data;
44 
45     i2c_start();
46     i2c_send_byte(0x90 + addr); //  w=0, write pointer
47 
48     i2c_send_byte(pointer); // includes reading ack
49 
50     i2c_start();
51     i2c_send_byte(0x90 + addr + 1); // read
52 
```

```
53      data = i2c_receive_byte();
54
55
56      i2c_stop();
57
58      return data;
59  }
```

Here is the call graph for this function:

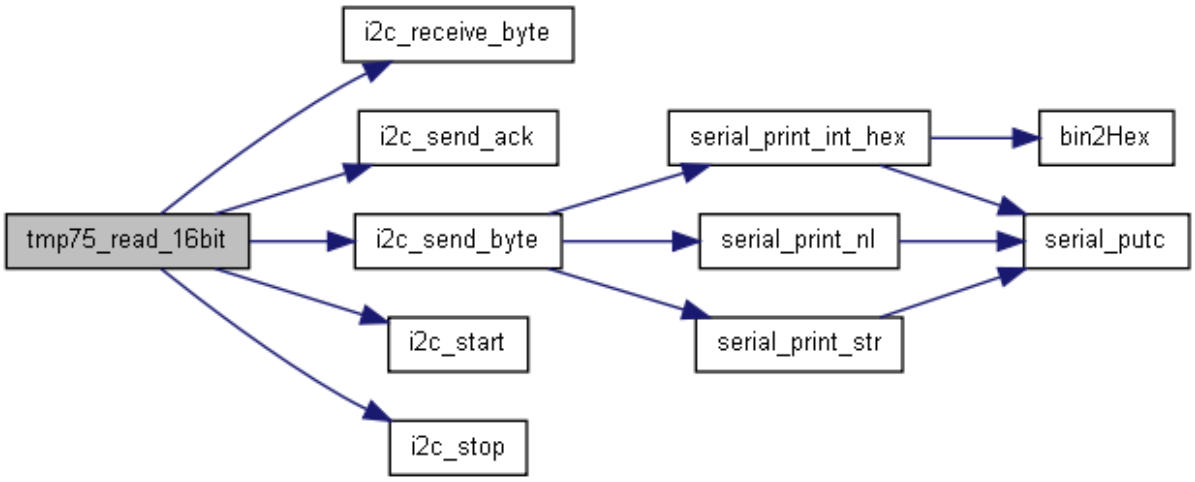


Here is the caller graph for this function:

uns16 tmp75_read_16bit (uns8 *addr*, uns8 *pointer*)

```
73                                                  {
74  uns16 data;
75
76      i2c_start();
77      i2c_send_byte(0x90 + addr); // LSB=0 == write
78      i2c_send_byte(pointer);
79
80      i2c_start();
81      i2c_send_byte(0x90 + addr + 1); // LSB=0 == read
82      data = i2c_receive_byte();
83
84      i2c_send_ack();
85
86      data = data << 8 | i2c_receive_byte(); // reuse local variable
87
88      i2c_send_ack();
89
90      i2c_stop();
91  }
```

Here is the call graph for this function:



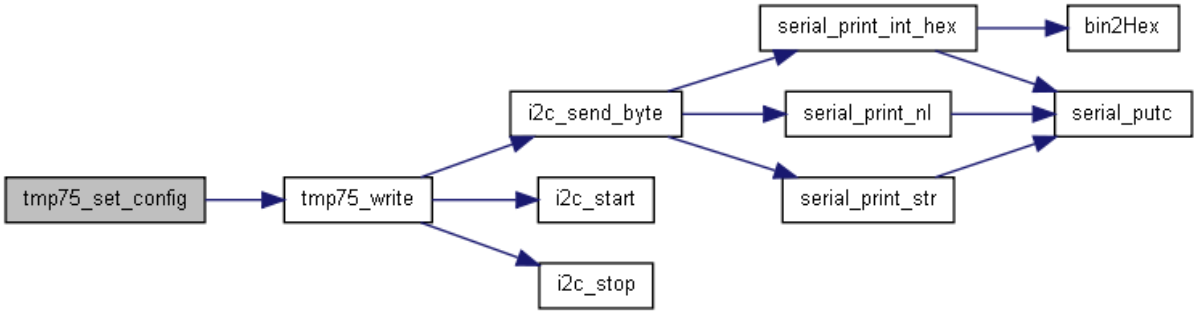
Here is the caller graph for this function:

void tmp75_set_config (uns8 *addr*, uns8 *config*)

Sets the tmp75 config register

```
98 {
99     tmp75_write(addr, TMP75_CONFIG_REGISTER, config);
100 }
```

Here is the call graph for this function:



void tmp75_setup ()

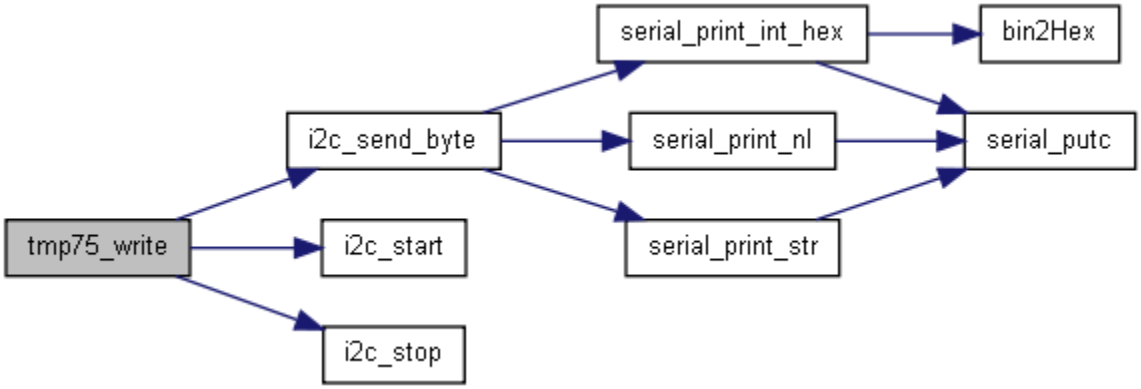
```
93                     {
94     i2c_setup();
95 }
```

uns8 tmp75_write (uns8 *addr*, uns8 *pointer*, uns8 *data*)

```
61                                                     {
62 
63     i2c_start();
64     i2c_send_byte(0x90 + addr);
```

```
65      i2c_send_byte(pointer);
66
67      i2c_send_byte(data);
68
69      i2c_stop();
70  }
```

Here is the call graph for this function:

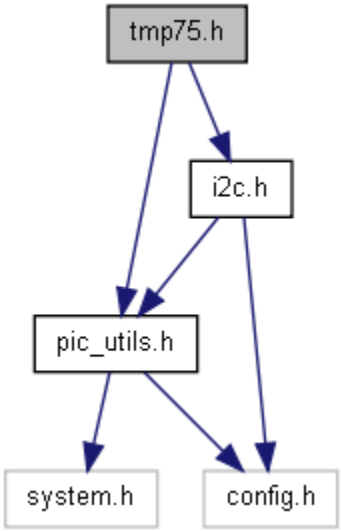


Here is the caller graph for this function:

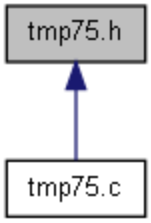
---

tmp75.h File Reference

Routines to access TMP75 temperature sensor.
Include dependency graph for tmp75.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define TMP75_CONF_F0 3
- #define TMP75_CONF_F1 4
- #define TMP75_CONF_OS 7
- #define TMP75_CONF_POL 2
- #define TMP75_CONF_R0 5
- #define TMP75_CONF_R1 6
- #define TMP75_CONF_SD 0
- #define TMP75_CONF_TM 1
- #define TMP75_CONFIG_REGISTER 0b00000001
- #define tmp75_setup() tmp75_setup_io()
- *Setup tmp75 ports and pins.* #define TMP75_TEMP_REGISTER 0b00000000
- #define TMP75_THI_REGISTER 0b00000011
- #define TMP75_TLOW_REGISTER 0b00000010

Functions

- void tmp75_convert_temp (uns8 addr)
- *Start temperature conversion on tmp75.* uns8 tmp75_get_config (uns8 addr)
- *Get tmp75 config register.* uns16 tmp75_get_temp (uns8 addr)
- *Read temperature from tmp75.* void tmp75_set_config (uns8 addr, uns8 config)
- *Set tmp75 config register.* void tmp75_setup_io (void)

Detailed Description

Put the following in your config.h

```
// - - - - - - - - - - - - - - - - - - - -
// TMP75 defines
// - - - - - - - - - - - - - - - - - - - -

#define TMP75_ADDR  0x00
```

Define Documentation

#define TMP75_CONF_F0 3

#define TMP75_CONF_F1 4

#define TMP75_CONF_OS 7

#define TMP75_CONF_POL 2

#define TMP75_CONF_R0 5

#define TMP75_CONF_R1 6

#define TMP75_CONF_SD 0

#define TMP75_CONF_TM 1

#define TMP75_CONFIG_REGISTER 0b00000001

#define tmp75_setup() tmp75_setup_io()

#define TMP75_TEMP_REGISTER 0b00000000

#define TMP75_THI_REGISTER 0b00000011

#define TMP75_TLOW_REGISTER 0b00000010

Function Documentation

void tmp75_convert_temp (uns8 *addr*)

This routine starts the temperature conversion in the tmp75. Issue this command before actually reading the temperature.

needs fixing

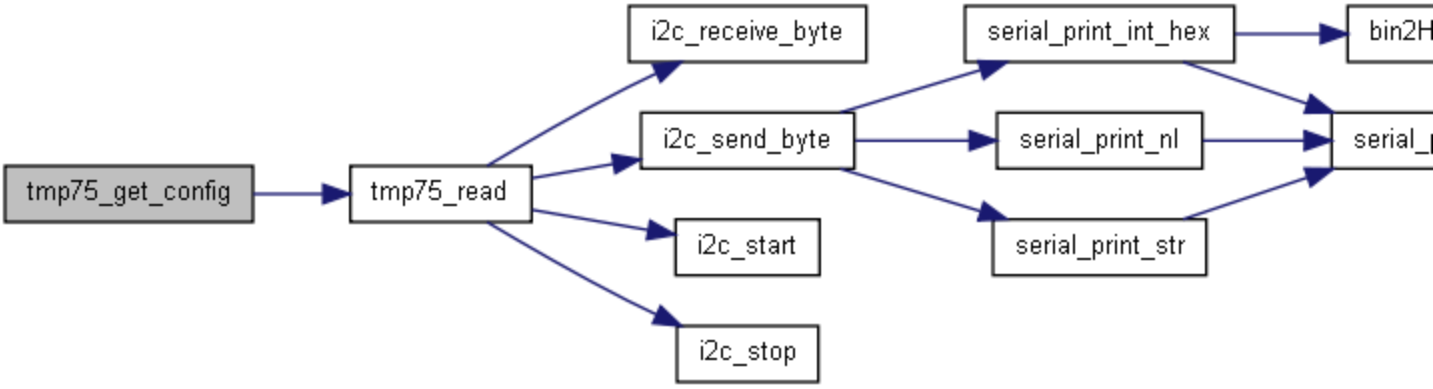
```
107                                     {
109 //  i2c_start();
110 //  i2c_send_byte(0x90 + addr);
111 //  i2c_send_byte(ds1631_start_convert);
112 //  i2c_stop();
113 }
```

uns8 tmp75_get_config (uns8 *addr*)

Gets the tmp75 config register (memory location 0x01)

```
103 {
104     return tmp75_read(addr, TMP75_CONFIG_REGISTER);
105 }
```

Here is the call graph for this function:

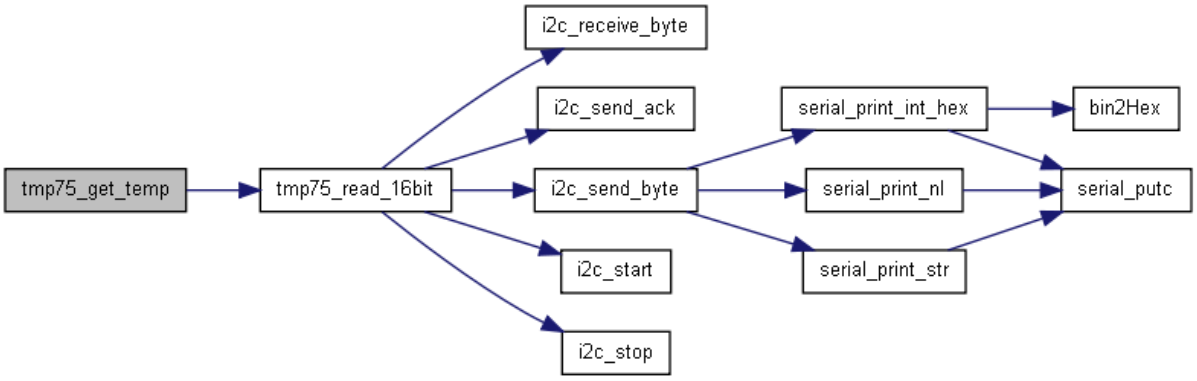


uns16 tmp75_get_temp (uns8 *addr*)

Returns 16bit raw temperature register from tmp75.

```
116 {
117     return tmp75_read_16bit(addr, TMP75_TEMP_REGISTER);
118 
119 }
```

Here is the call graph for this function:

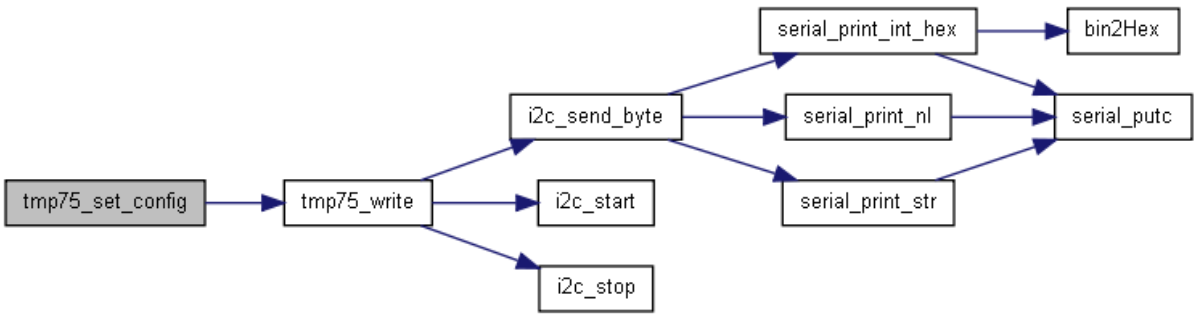


void tmp75_set_config (uns8 *addr*, uns8 *config*)

Sets the tmp75 config register

```
98 {
99     tmp75_write(addr, TMP75_CONFIG_REGISTER, config);
100 }
```

Here is the call graph for this function:

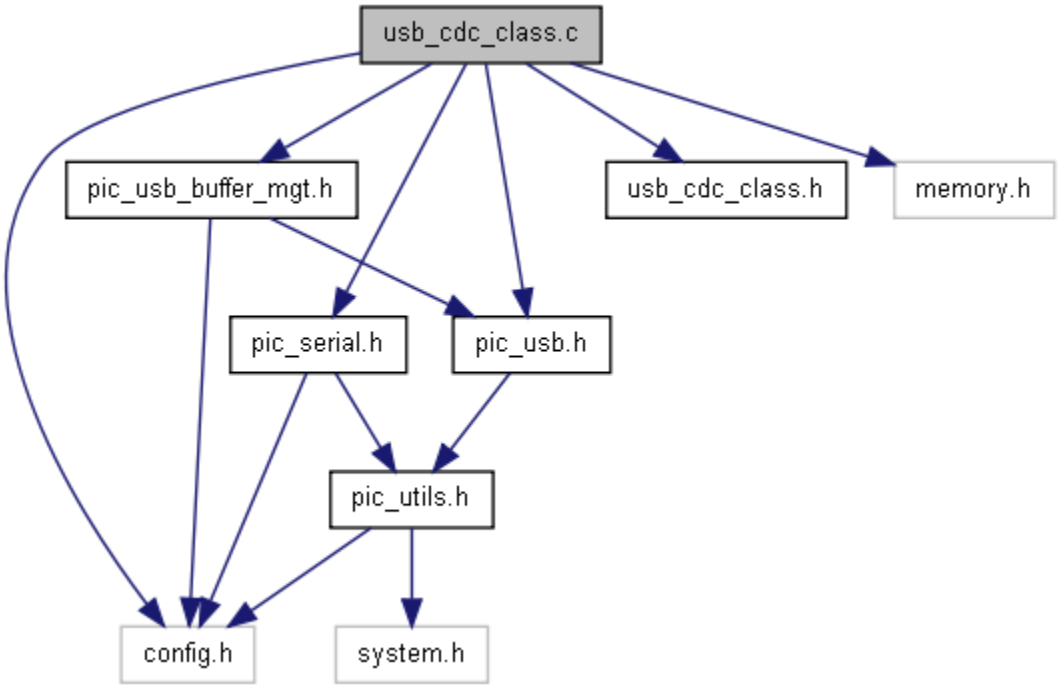


void tmp75_setup_io (void)

---

usb_cdc_class.c File Reference

Pic CDC USB routines.
Include dependency graph for usb_cdc_class.c:



Data Structures

- struct line_coding
- union long_union

Defines

- #define not_SERIAL_STATE 0xa1
- #define req_CLEAR_COMM_FEATURE 0x04
- #define req_GET_COMM_FEATURE 0x03
- #define req_GET_ENCAPSULATED_RESPONSE 0x01
- #define req_GET_LINE_CODING 0x21
- #define req_SEND_BREAK 0x23
- #define req_SEND_ENCAPSULATED_COMMAND 0x00
- #define req_SET_COMM_FEATURE 0x02
- #define req_SET_CONTROL_LINE_STATE 0x22
- #define req_SET_LINE_CODING 0x20

Functions

- uns8 usb_cdc_getc (void)
- *Retrieve a received character from the USB serial port.* void usb_cdc_handle_tx ()
- *Transmit any buffered characters over ther USB virtual serial port.* void usb_cdc_print_int (uns16 i)
- *Print a 16 bit number to the USB virtual serial port.* void usb_cdc_print_str (char *str)
- *Print a string out to the USB virtual serial port.* void usb_cdc_putc (uns8 c)
- *Sends a single character to the USB serial port.* uns8 usb_cdc_rx_avail ()
- *Check to see if a character is available in the receive buffer.* void usb_cdc_set_dcd ()
- void usb_cdc_set_dsr ()
- void usb_cdc_setup ()
- *Set up data structures ready for USB CDC use.* uns8 usb_cdc_tx_empty ()
- *Check to see if the transmit buffer is empty.* void usb_ep_data_in_callback (uns8 end_point, uns16 byte_count)
- *Callback routine triggered when data has been sent to the host.* void usb_ep_data_out_callback (uns8 end_point, uns8 *buffer, uns16 byte_count)
- *Callback routine triggered when data has been sent to the device.* void usb_handle_class_ctrl_read_callback ()
- *Callback routine for a class control read.* void usb_handle_class_ctrl_write_callback (uns8 *data, uns16 count)
- *Callback routine for a class control write.* void usb_handle_class_request_callback (setup_data_packet sdp)
- *Callback routine for a control transfer request that is placed on the class.* void usb_SOF_callback (uns16 frame)

Callback routine triggered each time a start of frame (SOF) has been received.

Variables

- uns8 cdc_rx_buffer [USB_CDC_RX_BUFFER_SIZE]
- uns8 cdc_rx_end = 0
- uns8 cdc_rx_start = 0
- uns8 cdc_tx_buffer [USB_CDC_TX_BUFFER_SIZE]
- uns8 cdc_tx_end = 0
- uns8 cdc_tx_start = 0
- uns8 class_data [8]
- uns16 current_bit_rate
- uns8 data_bits
- long_union dte_rate
- uns8 parity

Detailed Description

Communication Device Class (Serial Port) USB routines

Define Documentation

#define not_SERIAL_STATE 0xa1

#define req_CLEAR_COMM_FEATURE 0x04

#define req_GET_COMM_FEATURE 0x03

#define req_GET_ENCAPSULATED_RESPONSE 0x01

#define req_GET_LINE_CODING 0x21

#define req_SEND_BREAK 0x23

#define req_SEND_ENCAPSULATED_COMMAND 0x00

#define req_SET_COMM_FEATURE 0x02

#define req_SET_CONTROL_LINE_STATE 0x22

#define req_SET_LINE_CODING 0x20

Function Documentation

uns8 usb_cdc_getc ()

Receive a character from the USB serial port. If a character has not been received, this routine will wait indefinately.

Returns:

Byte from the receive buffer.

```
377 {
378     uns8 cdc_rx_char, cdc_rx_next;
379 
380     while(cdc_rx_end == cdc_rx_start);  // wait until there is something received
381 
382     start_crit_sec();   // make sure nobody else can muck with the buffer
383 
384     cdc_rx_char = cdc_rx_buffer[cdc_rx_start];  // get character from the front of the buffer
385     cdc_rx_start++; // increment fifo start
386     if (cdc_rx_start == USB_CDC_RX_BUFFER_SIZE) {   // if we're at the end
387         cdc_rx_start = 0;   // then wrap to the beginning
388     }
389 
390     end_crit_sec(); // now they can muck with the buffer
391 
392     return (cdc_rx_char);   // return the result we first thought of
393 
394 }   // -- getc
```

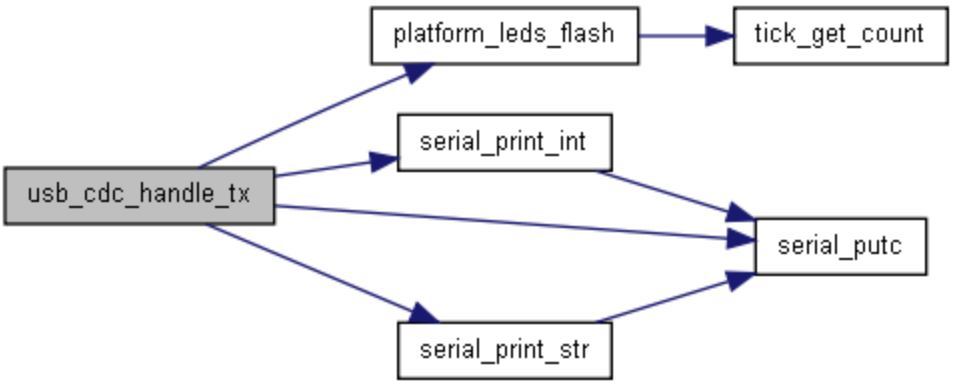
void usb_cdc_handle_tx ()

Generally only used internally, this routine will attempt to place any characters in the transmit queue into the USB buffers. It will fail gracefully if the USB buffer is already owned by the SIE.

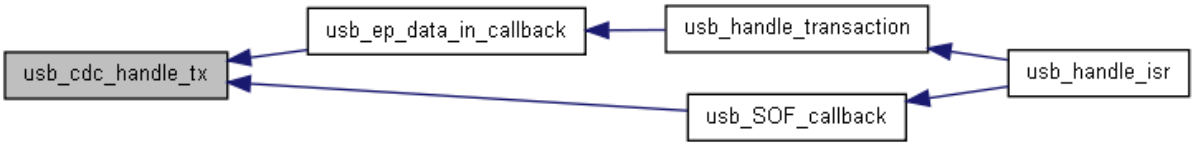
```
398 {
399 uns8 cdc_tx_next;
400 uns8 count;
401 uns16 buffer_size;
402 uns8 *buffer;
```

```
403 buffer descriptor *bd;
404
405     bd = ep in bd location[USB CDC DATA ENDPOINT];
406     if (test_bit(bd->stat, UOWN)) { // if there's already something in play
407         return; // give up
408     }
409
410     buffer_size = ep in buffer size[USB_CDC_DATA_ENDPOINT];
411     buffer = ep in buffer location[USB_CDC_DATA_ENDPOINT];
412
413     if (cdc tx end == cdc tx start) { // anything in the fifo?
414        return; // nope
415     }
416     #ifdef CDC_DEBUG
417         serial putc('<');
418     #endif
419     start crit sec();
420
421     count = 0;
422     while ((cdc tx end != cdc tx start) && (count < buffer_size)) {
423
424         cdc_tx_next = cdc tx start + 1; // get next position
425         if (cdc_tx_next == USB_CDC_TX_BUFFER_SIZE) {    // if we're at the end of the buffer
426             cdc_tx_next = 0;    // wrap to the beginning
427         }
428         buffer[count] = cdc tx buffer[cdc tx start];    // transmit the character
429         #ifdef CDC_DEBUG
430             serial putc(buffer[count]);
431         #endif
432         count++;
433         cdc tx start = cdc_tx_next; // move start position of fifo
434     }
435     if (count > 0) {
436         bd->count = count;
437         bd->addr = (uns16)buffer;
438
439         toggle_bit(bd->stat, DTS);
440         clear_bit(bd->stat, KEN);   // clear the keep bit
441         clear_bit(bd->stat, INCDIS);    // clear the increment disable
442         set bit  (bd->stat, DTSEN);
443         clear_bit(bd->stat, BSTALL);    // clear stall bit
444         clear_bit(bd->stat, BC9);
445         clear_bit(bd->stat, BC8);
446
447         set_bit  (bd->stat, UOWN);  // SIE owns the buffer
448     }
449     end crit sec();
450     #ifdef CDC_DEBUG
451         serial putc('>');
452         serial print str("send=");
453         serial print int(count);
454         serial putc(' ');
455     #endif
456     #ifdef USB_CDC_USE_LEDS
457         platform leds flash(2);
458     #endif
459 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_cdc_print_int (uns16 *i*)

Print a 16 bit unsigned number in decimal to the USB virtual serial port

Parameters:

i the 16 bit number to be printed

```
497                                 {
498
499 char buffer[6]; // up to 5 characters plus \0
500 uns8 count = 5;
501     buffer[5] = '\0';
502     do {
503         count--;
504         buffer[count] = '0' + i % 10;
505         i = i / 10;
506     } while (i > 0);
507     while (buffer[count]) {
508         usb_cdc_putc(buffer[count]);
509         count++;
510     }
511     //serial_print_str(&buffer[count]); //  print it out
512 //  for(count = 0 ; str[count] != 0; count++)
513  //   {
514  //   }
515
516 }
```

Here is the call graph for this function:

void usb_cdc_print_str (char * *str*)

Send a null terminated string out the virtual serial port

Parameters:

str the string to be sent

```
464                                 {
```

```
465
466 uns8 count;
467 buffer descriptor *bd;
468
469     for(count = 0 ; str[count] != 0; count++)
470     {
471         usb_cdc_putc(str[count]);
472     }
473
474     // This will give possibly quicker send:
475
476     //bd = ep_in_bd_location[CDC_DATA_ENDPOINT];
477     //if (!test_bit(bd->stat, UOWN)) {
478     //  usb_cdc_handle_tx();
479     //}
480     // otherwise we wait for the SOF interrupt to send
481 }
```

Here is the call graph for this function:

void usb_cdc_putc (uns8 *c*)

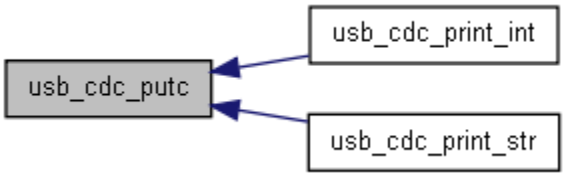
Deliver a single character out the virtual serial port. This routine will add the character to the transmit buffer. The actual buffer will be physically sent on the Start Of Frame (SOF) interrupt (each 1ms) or on the end point interrupt - ie, when the last character or chunk of characters was sent.

Parameters:

c The 8 bit byte to be transmitted.

```
348                             {
349 uns8 cdc_tx_next;
350 bit  my_store_gie;
351 #ifdef CDC_IDE_DEBUG
352 return;
353 #endif
354
355     cdc_tx_next = cdc_tx_end + 1;   // get next buffer position
356     if (cdc_tx_next == USB_CDC_TX_BUFFER_SIZE) {    // if we're at the end
357         cdc_tx_next = 0;    // wrap to the beginning
358     }
359
360     if ((!intcon.GIE) && (cdc_tx_next == cdc_tx_start)) {
361         return;
362     }
363     while (cdc_tx_next == cdc_tx_start) {
364
365     }
366
367     start_crit_sec();
368
369     cdc_tx_buffer[cdc_tx_end] = c; // put it in
370     cdc_tx_end = cdc_tx_next;  // move pointer along
371
372     end_crit_sec();
373
374 }
```

Here is the caller graph for this function:



uns8 usb_cdc_rx_avail ()

If one or more bytes are available in the USB serial port receive buffer, this routine will return true. If there are no bytes available, it will return false.

Returns:

True if buffer is not empty, False if buffer is empty.

```
461 { return cdc_rx_start != cdc_rx_end; }
```

void usb_cdc_set_dcd ()

```
340                         {
341 }
```

void usb_cdc_set_dsr ()

```
343                         {
344 }
```

void usb_cdc_setup ()

Configures the default DTE rate, stop bits etc.

```
488                        {
489
490     current_bit_rate = SPBRG_9600;
491     parity = 0;
492     // 9600
493     dte_rate.as_long = 9600;
494
495 }
```

uns8 usb_cdc_tx_empty ()

Sometimes it is useful to see if the transmit buffer is empty, since then you can be sure your data is well on its way. In the case of USB, this means that the data has been at least placed into the outbound USB buffer; it's not possible to tell until after the fact if the data has actually been squirted out the USB port.

Returns:

True if transmit buffer is empty, False if buffer still has data in it.

```
462 { return cdc_tx_start == cdc_tx_end; }
```

void usb_ep_data_in_callback (uns8 *end_point*, uns16 *byte_count*)

If you have called usb_send_data to transfer data to the host, this routine will be fired once this data has been transferred. You may send more data by using usb_send_data(). Since the current PicPack USB library supports only single buffering, transfer speed is limited by how quickly you can refill the buffer

again. In the future, we may support double buffering (ping poing buffering) which will most likely improve transfer speeds (to be fair, transfer performance has not been a limiting factor in tests so far).

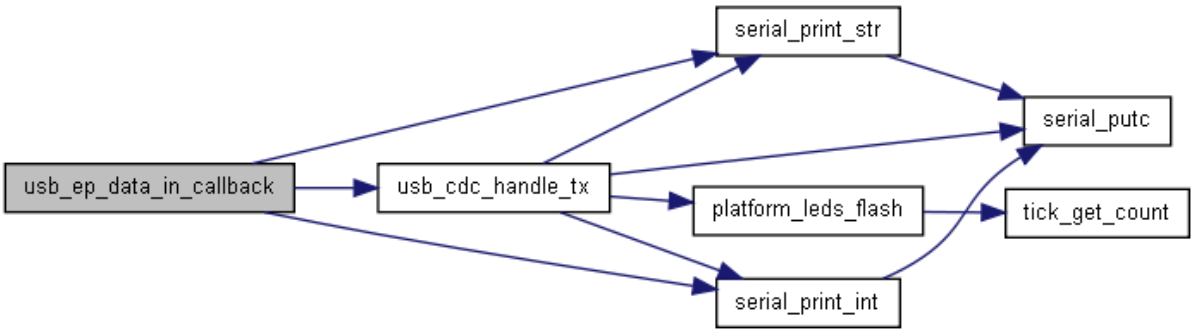
In order for this callback to be triggered, you must define USB_EP_DATA_CALLBACK in your config.h

Parameters:

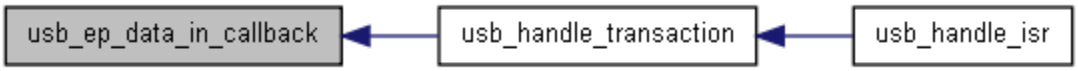
end_point The endpoint on which the data was transferred
byte_count The number of bytes that were actually transferred

```
328                                                                 {
329     #ifdef CDC_DEBUG
330         serial_print_str(" EP data in: ");
331         serial_print_int(byte_count);
332         serial_print_str(" bytes ");
333     #endif
334     // data has been sent, so do we need to send more?
335     if (end_point == USB_CDC_DATA_ENDPOINT) {   // it's the data end point
336         usb_cdc_handle_tx();
337     }
338 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_ep_data_out_callback (uns8 *end_point*, uns8 * *buffer_location*, uns16 *byte_count*)

If data is sent to the device and the endpoint is not endpoint 0 (the control transfer endpoint) then this routine is called. Since the routine is passed the actual hardware buffer location, it is important to pull data out of the buffer as soon as possible in order to free up the buffer to receive more data. The buffer is re-primed only once this routine completes since PicPack only supports single-buffered mode. In the future, we may look at supporting double buffering (ping-pong buffering) in order to be able to receive more data even while this routine is being called.

In order for this callback to be triggered, you must define USB_EP_DATA_CALLBACK in your config.h

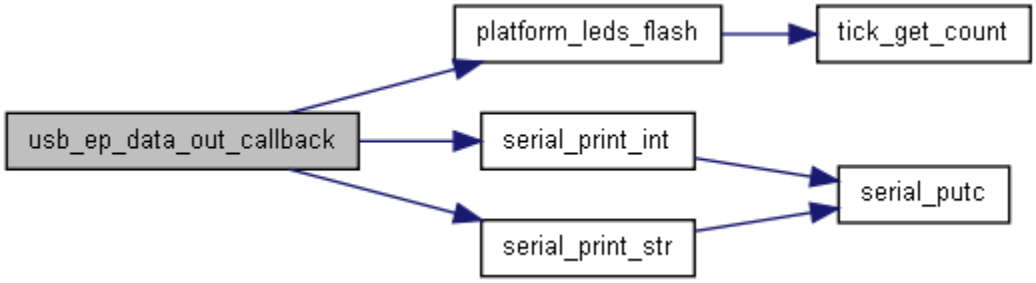
Parameters:

end_point The endpoint the data was sent do
buffer_lcoation The memory location of the USB buffer where the data was received into
byte_count The number of bytes received

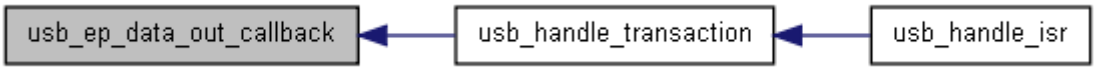
```
289                                                 {
290     uns8 cdc_rx_next;
291     #ifdef CDC_DEBUG
292         serial_print_str(" EP data out: ");
293         serial_print_int(byte_count);
```

```
294         serial_print_str(" bytes ");
295     #endif
296
297     // We have some data!
298
299     if (end_point == USB_CDC_DATA_ENDPOINT) {   // it's the data end point
300         uns8 count;
301         for (count = 0; count < byte_count; count++) {
302             cdc_rx_next = cdc_rx_end + 1;   // get next buffer position
303             if (cdc_rx_next == USB_CDC_RX_BUFFER_SIZE) {    // if we're at the end
304                 cdc_rx_next = 0;    // then wrap to the beginning
305             }
306             if (cdc_rx_next != cdc_rx_start) { // if space in the fifo
307                 cdc_rx_buffer[cdc_rx_end] = buffer[count]; // put it in
308                 cdc_rx_end = cdc_rx_next;  // and move pointer along
309             } else {
310                 // else... just ignore it, we've lost a byte, no room in the inn
311                 break;
312             }
313         }
314     } else {
315         #ifdef CDC_DEBUG
316             serial_print_str("data for ep ");
317             serial_print_int(end_point);
318         #endif
319     }
320
321     #ifdef USB_CDC_USE_LEDS
322         platform_leds_flash(3);
323     #endif
324 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_handle_class_ctrl_read_callback ()

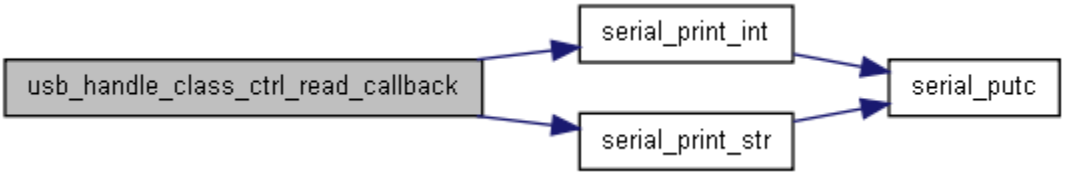
When a control transfer is taking place, this routine is called to indicate that a control read for the class has taken place. Since everything in USB land is all about what has just happened, this callback will occur after data has been transferred to the host. If you wish to send more data to the host, use usb_send_data(), or if your control read has sent all the data required, you will need to indicate that the state has changed by setting the control_mode variable to cm_CTRL_READ_AWAITING_STATUS. This will indicate to the stack that it should now wait for the status packet before completing the control transfer.

To allow this callback to trigger, ensure you define USB_CALLBACK_ON_CLASS_CTRL in your config.h

```
273                                         {
```

```
274         switch (usb_sdp.bRequest) {
275             case req_GET_LINE_CODING:
276                 // we know we've already sent everything, so now wait for status
277                 control_mode = cm_CTRL_READ_AWAITING_STATUS;
278                 break;
279             default:
280                 #ifdef CDC_DEBUG
281                     serial_print_str(" cl read ?? ");
282                     serial_print_int(usb_sdp.bRequest);
283                 #endif
284         }
285 
286 }
```

Here is the call graph for this function:



void usb_handle_class_ctrl_write_callback (uns8 * *data*, uns16 *count*)

When a control transfer is taking place, this routine is called to indicate that a control write for the class has taken place. Since everything in USB land is all about what has just happened, this callback will occur after data has been received by the device. If you expect more data from the host, it will arrive in due course since endpoint 0 will be primed for more data automatically. If you have received all the data from the host, you will need to set the control_mode state variable to cm_CTRL_WRITE_SENDING_STATUS and then actually send the status by calling usb_send_status_ack(). Once the status has actually been sent, the control_mode state will automatically change to cm_IDLE to indicate the transfer has completed.

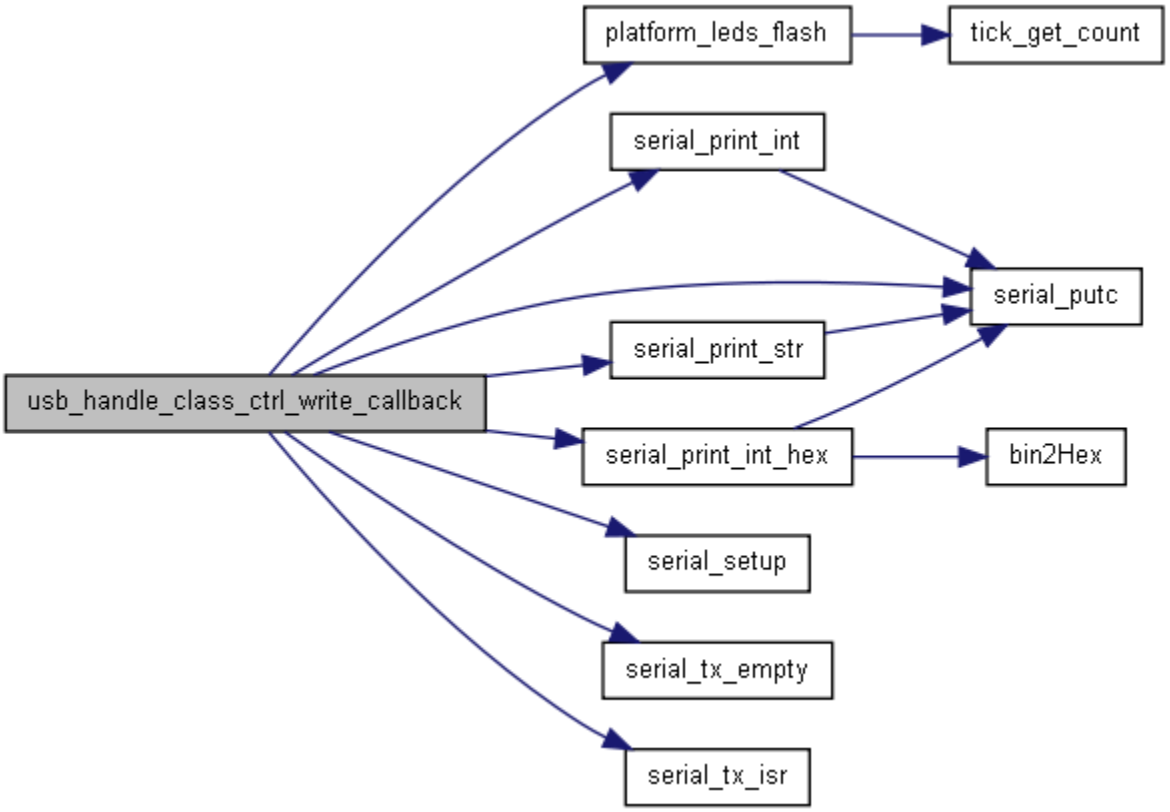
To allow this callback to trigger, ensure you define USB_CALLBACK_ON_CLASS_CTRL in your config.h

```
171                                                                 {
172 
173     switch (usb_sdp.bRequest) {
174         case req_SET_LINE_CODING:
175             // dump it into class_data
176             memcpy(/* dst */ (void *)&class_data,/* src */ (void *)data, count);
177 
178             // Now we need to send an ACK status back
179 
180             usb_send_status_ack();
181             control_mode = cm_CTRL_WRITE_SENDING_STATUS;
182 
183             line_coding *my_lc;
184             my_lc = (line_coding*) &class_data;
185             #ifdef CDC_DEBUG
186                 serial_print_int_hex(my_lc->dte_rate.as_byte_array[0]);
187                 serial_print_int_hex(my_lc->dte_rate.as_byte_array[1]);
188                 serial_print_int_hex(my_lc->dte_rate.as_byte_array[2]);
189                 serial_print_int_hex(my_lc->dte_rate.as_byte_array[3]);
190                 serial_print_str(" st=");
191                 serial_print_int(my_lc->stop_bits);
192                 serial_print_str(" p=");
193                 serial_print_int(my_lc->parity);
194                 serial_print_str(" db=");
195                 serial_print_int(my_lc->data_bits);
196                 serial_print_str(" bit rate: ");
197             #endif
198 
```

```
199             dte rate.as byte array[0] = my_lc->dte rate.as byte array[3];
200             dte rate.as byte array[1] = my_lc->dte rate.as byte array[2];
201             dte rate.as byte array[2] = my_lc->dte rate.as byte array[1];
202             dte rate.as byte array[3] = my_lc->dte rate.as byte array[0];
203             parity = my_lc->parity;
204             data bits = my_lc->data bits;
205 
206             switch (my_lc->dte rate.as long) {
207                 case 2400:
208                     current bit rate = SPBRG_2400;
209                     #ifdef CDC DEBUG
210                     serial print str("2400 ");
211                     #endif
212                     break;
213                 case 4800:
214                     current bit rate = SPBRG_4800;
215                     #ifdef CDC_DEBUG
216                     serial print str("4800 ");
217                     #endif
218                     break;
219                 case 9600:
220                     current bit rate = SPBRG_9600;
221                     #ifdef CDC DEBUG
222                     serial print str("9600 ");
223                     #endif
224                     break;
225                 case 19200:
226                     current bit rate = SPBRG 19200;
227                     #ifdef CDC_DEBUG
228                     serial print str("19200 ");
229                     #endif
230                     break;
231                 case 38400:
232                     current bit rate = SPBRG_38400;
233                     #ifdef CDC_DEBUG
234                     serial print str("38400 ");
235                     #endif
236                     break;
237                 case 115200:
238                     current bit rate = SPBRG 115200;
239                     #ifdef CDC_DEBUG
240                     serial print str("115200 ");
241                     #endif
242                     break;
243                 default:
244                     #ifdef CDC_DEBUG
245                         serial print str("Don't handle this bit rate");
246                     #endif
247             }
248 
249             clear_bit(rcsta, SPEN);
250             clear_bit(txsta, TXEN);
251             clear_bit(rcsta, CREN);
252 
253 
254             serial setup(current bit rate);
255             if (!serial tx empty()) {
256                 #ifdef USB_CDC_USE_LEDS
257                     platform leds flash(1);
258                 #endif
259                 set bit(pie1, TXIE);
260                 serial tx isr();
261             }
262             break;
263         default:
264             #ifdef CDC_DEBUG
265                 serial print str(" ??cw req=");
266                 serial print int hex(usb sdp.bRequest);
267                 serial putc(' ');
268             #endif
269             break;
```

```
270     }
271 }
```

Here is the call graph for this function:



void usb_handle_class_request_callback (setup_data_packet *sdp*)

After receiving a setup packet, where the request is placed on the class, this routine is called. In usb_handle_class_request_callback, you can set up ready for the data stage of the control transfer. The direction of the data stage can be determined by examining test_bit(sdp.bRequest, DATA_STAGE_DIR) although generally it appears to be obvious from the request. The request is stored in sdp.bRequest.

Typically, if it is a control read transfer (that is, it is a request by the host for data), then you will need to move the control_mode state variable to cm_CTRL_READ_DATA_STAGE_CLASS and send data using usb_send_data(). If you only intend to send one packet, you can immediately move the control_mode state variable to cm_CTRL_READ_AWAITING_STATUS to indicate you are waiting for the status to arrive. You could wait for the usb_handle_class_ctrl_read callback and do it (move to cm_CTROL_READ_AWAITING_STATUS) but the PicPack USB stack can handle the control read event for you if you've already switched states.

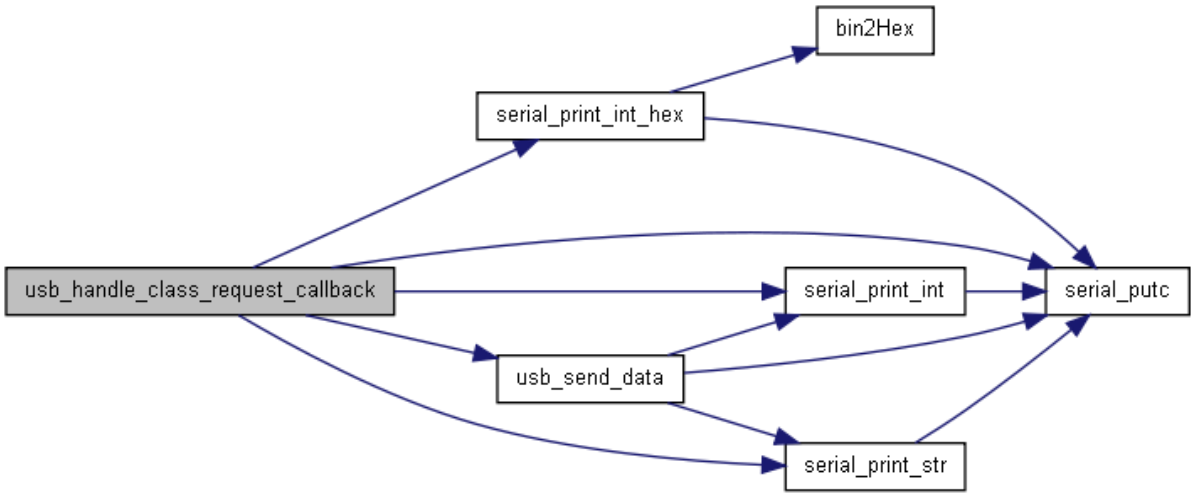
If it is a control write transfer (that is, it is a request by the host to send data to the device), then you will need to move the control_mode state variable to cm_CTRL_WRITE_DATA_STAGE_CLASS. Then, the usb_handle_class_ctrl_write will be fired when data is received by the device in the data stage.

To allow this callback to trigger, ensure you define USB_CALLBACK_ON_CLASS_CTRL in your config.h

```
115                                                         {
116
117     switch (sdp.bRequest) {
```

```
118         case req_SET_LINE_CODING:
119             // we now expect the line coding to arrive in the data stage
120 
121             #ifdef CDC_DEBUG
122                 serial_print_str("SET_LINE ");
123             #endif
124             control_mode = cm_CTRL_WRITE_DATA_STAGE_CLASS;
125             break;
126         case req_GET_LINE_CODING:
127             #ifdef CDC_DEBUG
128                 serial_print_str("GET LINE ");
129                 serial_print_str(" len=");
130                 serial_print_int(sdp.wLength);
131                 serial_putc(' ');
132             #endif
133             //control_mode = cm_CTRL_READ_DATA_STAGE_CLASS;
134             control_mode = cm_CTRL_READ_DATA_STAGE_CLASS;
135             //  need to prime ep0 IN with some funky data here
136             line_coding my_line_coding;
137 
138             // We stored dte rate from ealier
139             my_line_coding.dte_rate.as_byte_array[0] = dte_rate.as_byte_array[3];
140             my_line_coding.dte_rate.as_byte_array[1] = dte_rate.as_byte_array[2];
141             my_line_coding.dte_rate.as_byte_array[2] = dte_rate.as_byte_array[1];
142             my_line_coding.dte_rate.as_byte_array[3] = dte_rate.as_byte_array[0];
143             my_line_coding.stop_bits = 0;   // 1 stop bit
144             my_line_coding.data_bits = data_bits;
145             my_line_coding.parity = parity;
146 
147             usb_send_data(/*ep*/ 0, /*data*/ (uns8 *)&my_line_coding, /*count*/
sizeof(my_line_coding), /*first*/ 1);
148             // actually we know this will be the last packet, so go straight to waiting for
the status ack
149             control_mode = cm_CTRL_READ_AWAITING_STATUS;
150 
151             break;
152         case req_SET_CONTROL_LINE_STATE:
153             #ifdef CDC_DEBUG
154                 serial_print_str("scls=");//dtr = bit 0, rts = bit 1
155                 serial_print_int_hex(sdp.wValue);
156             #endif
157             // no data, so just ack the status
158             // !!! set dtr, rts
159             usb_send_status_ack();
160             control_mode = cm_CTRL_WRITE_SENDING_STATUS;
161             // Could put a callback here for your own code when DTR or RTS change
162             break;
163         default:
164             #ifdef CDC_DEBUG
165                 serial_print_str("??r=");
166                 serial_print_int(sdp.bRequest);
167             #endif
168     }
169 }
```

Here is the call graph for this function:



void usb_SOF_callback (uns16 *frame*)

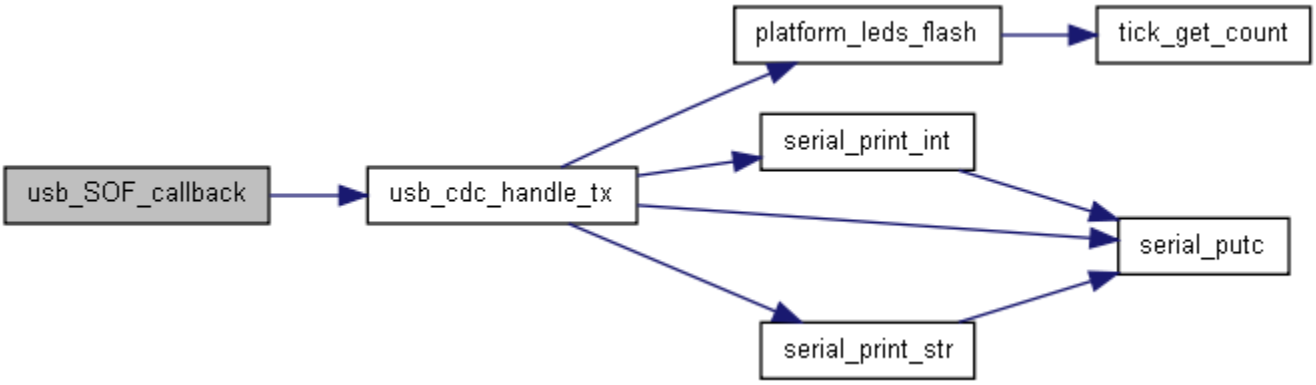
Frames in USB occur each 1ms. A SOF packet is sent to each device at the start of each frame. This is a really neat way of getting a 1ms timer without any further work.

Parameters:

frame The frame number. Frames will wrap at 65535.

```
483                                 {
484     // we don't care about the frame number, we only care if there's something to send...
485     usb_cdc_handle_tx();    // start transmission
486 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

Variable Documentation

uns8 cdc_rx_buffer[USB_CDC_RX_BUFFER_SIZE]

Receive fifo

uns8 cdc_rx_end = 0
Receive fifo end point

uns8 cdc_rx_start = 0
Receive fifo start point

uns8 cdc_tx_buffer[USB_CDC_TX_BUFFER_SIZE]
Transmit fifo

uns8 cdc_tx_end = 0
Transmit fifo end point

uns8 cdc_tx_start = 0
Transmit fifo start point

uns8 class_data[8]

uns16 current_bit_rate

uns8 data_bits

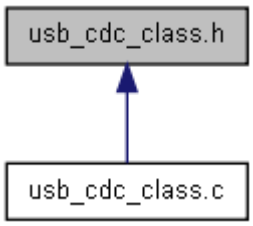
long_union dte_rate

uns8 parity

---

usb_cdc_class.h File Reference

USB Communications Device Class (Serial Port) routines.
This graph shows which files directly or indirectly include this file:



Defines

- #define PARITY_EVEN 2
- #define PARITY_MARK 3
- #define PARITY_NONE 0
- #define PARITY_ODD 1
- #define PARITY_SPACE 4

Functions

- uns8 usb_cdc_getc ()

- *Retrieve a received character from the USB serial port.* void usb_cdc_handle_tx ()
- *Transmit any buffered characters over ther USB virtual serial port.* void usb_cdc_print_int (uns16 i)
- *Print a 16 bit number to the USB virtual serial port.* void usb_cdc_print_str (char *str)
- *Print a string out to the USB virtual serial port.* void usb_cdc_putc (uns8 c)
- *Sends a single character to the USB serial port.* uns8 usb_cdc_rx_avail ()
- *Check to see if a character is available in the receive buffer.* void usb_cdc_setup ()
- *Set up data structures ready for USB CDC use.* uns8 usb_cdc_tx_empty ()

Check to see if the transmit buffer is empty. Variables

- uns8 data_bits
- uns8 parity

Detailed Description

Define Documentation

#define PARITY_EVEN 2

#define PARITY_MARK 3

#define PARITY_NONE 0

#define PARITY_ODD 1

#define PARITY_SPACE 4

Function Documentation

uns8 usb_cdc_getc ()

Receive a character from the USB serial port. If a character has not been received, this routine will wait indefinately.

Returns:

Byte from the receive buffer.

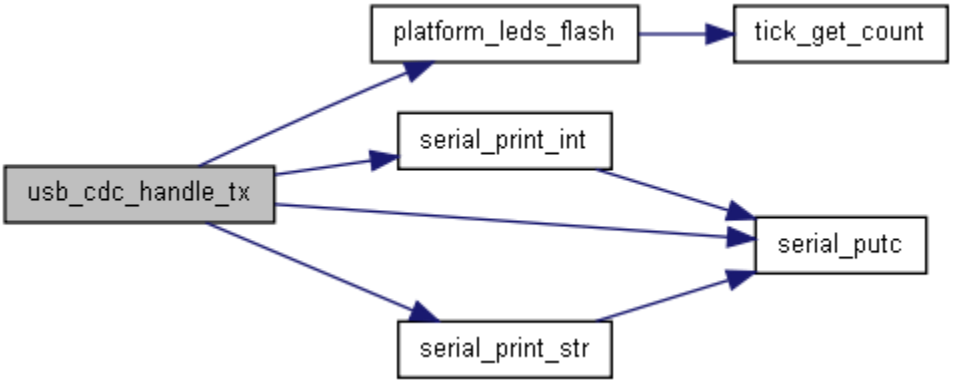
```
377 {
378     uns8 cdc_rx_char, cdc_rx_next;
379 
380     while(cdc_rx_end == cdc_rx_start);  // wait until there is something received
381 
382     start_crit_sec();   // make sure nobody else can muck with the buffer
383 
384     cdc_rx_char = cdc_rx_buffer[cdc_rx_start];  // get character from the front of the buffer
385     cdc_rx_start++; // increment fifo start
386     if (cdc_rx_start == USB_CDC_RX_BUFFER_SIZE) {   // if we're at the end
387         cdc_rx_start = 0;   // then wrap to the beginning
388     }
389 
390     end_crit_sec(); // now they can muck with the buffer
391 
392     return (cdc_rx_char);   // return the result we first thought of
393 
394 }   // -- getc
```

void usb_cdc_handle_tx ()

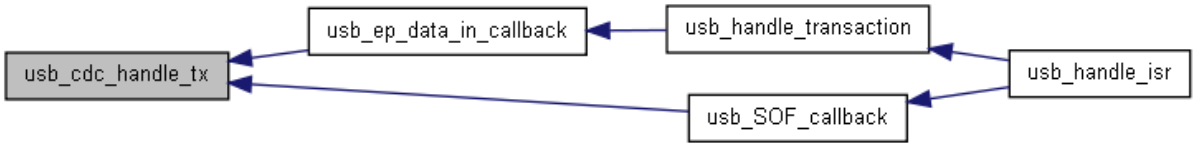
Generally only used internally, this routine will attempt to place any characters in the transmit queue into the USB buffers. It will fail gracefully if the USB buffer is already owned by the SIE.

```
398 {
399 uns8 cdc_tx_next;
400 uns8 count;
401 uns16 buffer_size;
402 uns8 *buffer;
403 buffer_descriptor *bd;
404 
405     bd = ep_in_bd_location[USB_CDC_DATA_ENDPOINT];
406     if (test_bit(bd->stat, UOWN)) { // if there's already something in play
407         return; // give up
408     }
409 
410     buffer_size = ep_in_buffer_size[USB_CDC_DATA_ENDPOINT];
411     buffer = ep_in_buffer_location[USB_CDC_DATA_ENDPOINT];
412 
413     if (cdc_tx_end == cdc_tx_start) { // anything in the fifo?
414        return; // nope
415     }
416     #ifdef CDC_DEBUG
417         serial_putc('<');
418     #endif
419     start_crit_sec();
420 
421     count = 0;
422     while ((cdc_tx_end != cdc_tx_start) && (count < buffer_size)) {
423 
424         cdc_tx_next = cdc_tx_start + 1; // get next position
425         if (cdc_tx_next == USB_CDC_TX_BUFFER_SIZE) {    // if we're at the end of the buffer
426             cdc_tx_next = 0;    // wrap to the beginning
427         }
428         buffer[count] = cdc_tx_buffer[cdc_tx_start];    // transmit the character
429         #ifdef CDC_DEBUG
430             serial_putc(buffer[count]);
431         #endif
432         count++;
433         cdc_tx_start = cdc_tx_next; // move start position of fifo
434     }
435     if (count > 0) {
436         bd->count = count;
437         bd->addr = (uns16)buffer;
438 
439         toggle_bit(bd->stat, DTS);
440         clear_bit(bd->stat, KEN);   // clear the keep bit
441         clear_bit(bd->stat, INCDIS);    // clear the increment disable
442         set_bit  (bd->stat, DTSEN);
443         clear_bit(bd->stat, BSTALL);    // clear stall bit
444         clear_bit(bd->stat, BC9);
445         clear_bit(bd->stat, BC8);
446 
447         set_bit  (bd->stat, UOWN);  // SIE owns the buffer
448     }
449     end_crit_sec();
450     #ifdef CDC_DEBUG
451         serial_putc('>');
452         serial_print_str("send=");
453         serial_print_int(count);
454         serial_putc(' ');
455     #endif
456     #ifdef USB_CDC_USE_LEDS
457         platform_leds_flash(2);
458     #endif
459 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void usb_cdc_print_int (uns16 *i*)

Print a 16 bit unsigned number in decimal to the USB virtual serial port

Parameters:

i the 16 bit number to be printed

```
497                                 {
498 
499 char buffer[6]; // up to 5 characters plus \0
500 uns8 count = 5;
501     buffer[5] = '\0';
502     do {
503         count--;
504         buffer[count] = '0' + i % 10;
505         i = i / 10;
506     } while (i > 0);
507     while (buffer[count]) {
508         usb_cdc_putc(buffer[count]);
509         count++;
510     }
511     //serial_print_str(&buffer[count]); //  print it out
512 //  for(count = 0 ; str[count] != 0; count++)
513  //   {
514  //   }
515 
516 }
```

Here is the call graph for this function:

void usb_cdc_print_str (char * *str*)

Send a null terminated string out the virtual serial port

Parameters:

str the string to be sent

```
464                                     {
465
466 uns8 count;
467 buffer_descriptor *bd;
468
469     for(count = 0 ; str[count] != 0; count++)
470     {
471         usb_cdc_putc(str[count]);
472     }
473
474     // This will give possibly quicker send:
475
476     //bd = ep_in_bd_location[CDC_DATA_ENDPOINT];
477     //if (!test_bit(bd->stat, UOWN)) {
478     //  usb_cdc_handle_tx();
479     //}
480     // otherwise we wait for the SOF interrupt to send
481 }
```

Here is the call graph for this function:

void usb_cdc_putc (uns8 *c*)

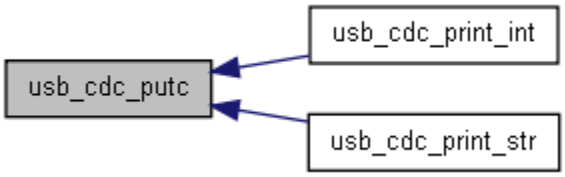
Deliver a single character out the virtual serial port. This routine will add the character to the transmit buffer. The actual buffer will be physically sent on the Start Of Frame (SOF) interrupt (each 1ms) or on the end point interrupt - ie, when the last character or chunk of characters was sent.

Parameters:

c The 8 bit byte to be transmitted.

```
348                             {
349 uns8 cdc_tx_next;
350 bit  my_store_gie;
351 #ifdef CDC_IDE_DEBUG
352 return;
353 #endif
354
355     cdc_tx_next = cdc_tx_end + 1;   // get next buffer position
356     if (cdc_tx_next == USB_CDC_TX_BUFFER_SIZE) {    // if we're at the end
357         cdc_tx_next = 0;    // wrap to the beginning
358     }
359
360     if ((!intcon.GIE) && (cdc_tx_next == cdc_tx_start)) {
361         return;
362     }
363     while (cdc_tx_next == cdc_tx_start) {
364
365     }
366
367     start_crit_sec();
368
369     cdc_tx_buffer[cdc_tx_end] = c; // put it in
370     cdc_tx_end = cdc_tx_next;  // move pointer along
371
372     end_crit_sec();
373
374 }
```

Here is the caller graph for this function:



uns8 usb_cdc_rx_avail ()

If one or more bytes are available in the USB serial port receive buffer, this routine will return true. If there are no bytes available, it will return false.

Returns:

True if buffer is not empty, False if buffer is empty.

```
461 { return cdc_rx_start != cdc_rx_end; }
```

void usb_cdc_setup ()

Configures the default DTE rate, stop bits etc.

```
488                         {
489
490     current_bit_rate = SPBRG_9600;
491     parity = 0;
492     // 9600
493     dte_rate.as_long = 9600;
494
495 }
```

uns8 usb_cdc_tx_empty ()

Sometimes it is useful to see if the transmit buffer is empty, since then you can be sure your data is well on its way. In the case of USB, this means that the data has been at least placed into the outbound USB buffer; it's not possible to tell until after the fact if the data has actually been squirted out the USB port.

Returns:

True if transmit buffer is empty, False if buffer still has data in it.

```
462 { return cdc_tx_start == cdc_tx_end; }
```

Variable Documentation

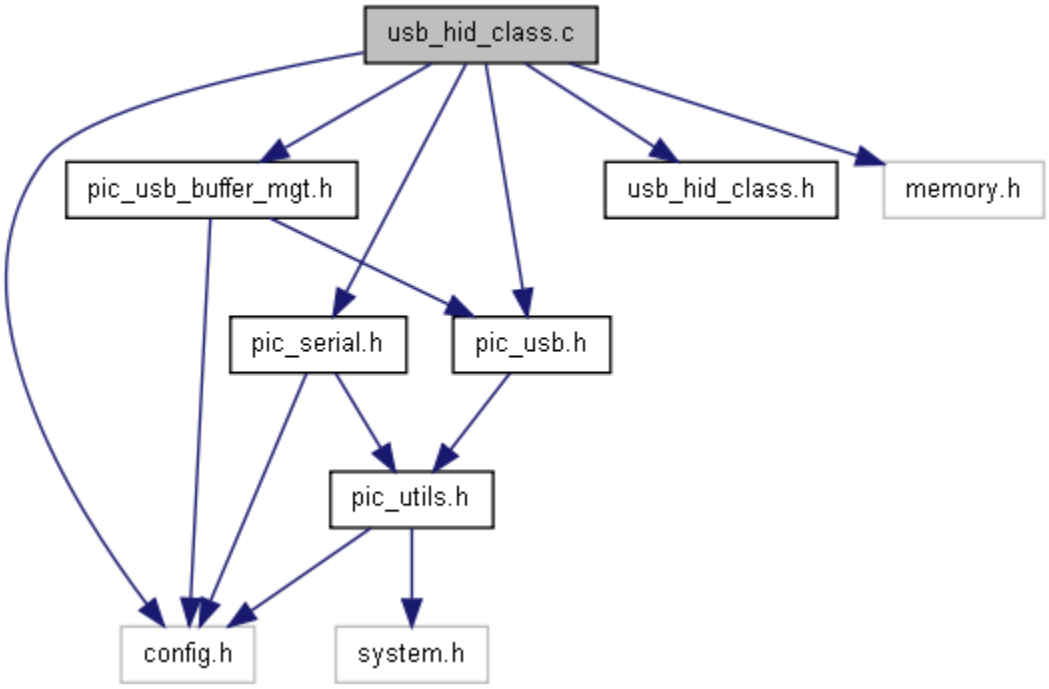
uns8 data_bits

uns8 parity

---

usb_hid_class.c File Reference

Callbacks for implementing USB HID class.
Include dependency graph for usb_hid_class.c:



Functions

- void usb_handle_class_ctrl_read_callback ()
- *Callback routine for a class control read.* void usb_handle_class_ctrl_write_callback (uns8 *data, uns16 count)
- *Callback routine for a class control write.* void usb_handle_class_request_callback (setup_data_packet sdp)

Callback routine for a control transfer request that is placed on the class.

Detailed Description

Function Documentation

void usb_handle_class_ctrl_read_callback ()

When a control transfer is taking place, this routine is called to indicate that a control read for the class has taken place. Since everything in USB land is all about what has just happened, this callback will occur after data has been transferred to the host. If you wish to send more data to the host, use usb_send_data(), or if your control read has sent all the data required, you will need to indicate that the state has changed by setting the control_mode variable to cm_CTRL_READ_AWAITING_STATUS. This will indicate to the stack that it should now wait for the status packet before completing the control transfer.

To allow this callback to trigger, ensure you define USB_CALLBACK_ON_CLASS_CTRL in your config.h

```
54                                              {
55 }
```

Here is the caller graph for this function:

void usb_handle_class_ctrl_write_callback (uns8 * *data*, uns16 *count*)

When a control transfer is taking place, this routine is called to indicate that a control write for the class has taken place. Since everything in USB land is all about what has just happened, this callback will occur after data has been received by the device. If you expect more data from the host, it will arrive in due course since endpoint 0 will be primed for more data automatically. If you have received all the data from the host, you will need to set the control_mode state variable to cm_CTRL_WRITE_SENDING_STATUS and then actually send the status by calling usb_send_status_ack(). Once the status has actually been sent, the control_mode state will automatically change to cm_IDLE to indicate the transfer has completed.

To allow this callback to trigger, ensure you define USB_CALLBACK_ON_CLASS_CTRL in your config.h

```
57                                                          {
58 }
```

Here is the caller graph for this function:

void usb_handle_class_request_callback (setup_data_packet *sdp*)

After receiving a setup packet, where the request is placed on the class, this routine is called. In usb_handle_class_request_callback, you can set up ready for the data stage of the control transfer. The direction of the data stage can be determined by examining test_bit(sdp.bRequest, DATA_STAGE_DIR) although generally it appears to be obvious from the request. The request is stored in sdp.bRequest.

Typically, if it is a control read transfer (that is, it is a request by the host for data), then you will need to move the control_mode state variable to cm_CTRL_READ_DATA_STAGE_CLASS and send data using usb_send_data(). If you only intend to send one packet, you can immediately move the control_mode state variable to cm_CTRL_READ_AWAITING_STATUS to indicate you are waiting for the status to arrive. You could wait for the usb_handle_class_ctrl_read callback and do it (move to cm_CTROL_READ_AWAITING_STATUS) but the PicPack USB stack can handle the control read event for you if you've already switched states.

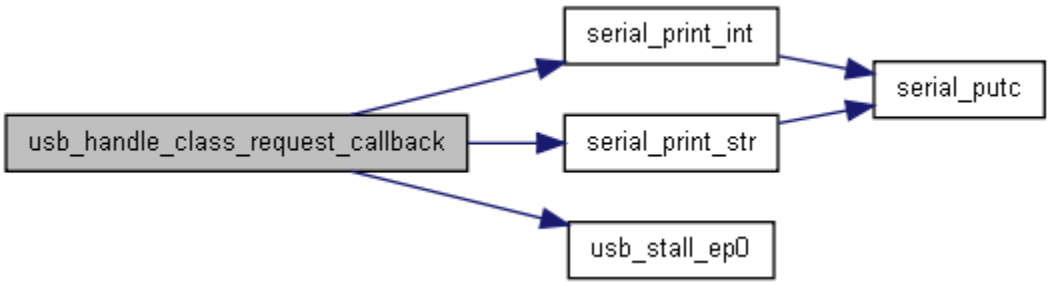
If it is a control write transfer (that is, it is a request by the host to send data to the device), then you will need to move the control_mode state variable to cm_CTRL_WRITE_DATA_STAGE_CLASS. Then, the usb_handle_class_ctrl_write will be fired when data is received by the device in the data stage.

To allow this callback to trigger, ensure you define USB_CALLBACK_ON_CLASS_CTRL in your config.h

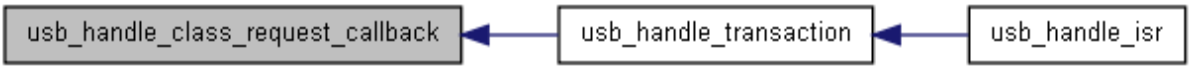
```
60                                                          {
61     #ifdef USB_DEBUG
62         serial_print_str("Class request: ");
63         serial_print_int(sdp.bRequest);
64     #endif
65
66     switch(sdp.bRequest) {
67         case req_GET_REPORT:
68             #ifdef USB_DEBUG
69                 serial_print_str(" Set_idle ");
70             #endif
```

```
71              break;
72          case req_GET_IDLE:
73              break;
74          case req_GET_PROTOCOL:
75              break;
76          case req_SET_REPORT:
77              #ifdef USB_DEBUG
78                  serial_print_str(" Set_report ");
79              #endif
80              break;
81          case req_SET_IDLE:
82              #ifdef USB_DEBUG
83                  serial_print_str(" Set_idle ");
84              #endif
85              // we don't support whatever they want
86              usb_stall_ep0();
87              break;
88          case req_SET_PROTOCOL:
89              break;
90
91      }
92
93  }
```

Here is the call graph for this function:



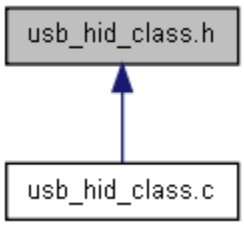
Here is the caller graph for this function:



---

usb_hid_class.h File Reference

Helper definitions for USB HID class.
This graph shows which files directly or indirectly include this file:



Defines

- #define req_GET_IDLE 0x02
- #define req_GET_PROTOCOL 0x03
- #define req_GET_REPORT 0x01
- #define req_SET_IDLE 0x0a
- #define req_SET_PROTOCOL 0x0b
- #define req_SET_REPORT 0x09

Detailed Description

Define Documentation

#define req_GET_IDLE 0x02

#define req_GET_PROTOCOL 0x03

#define req_GET_REPORT 0x01

#define req_SET_IDLE 0x0a

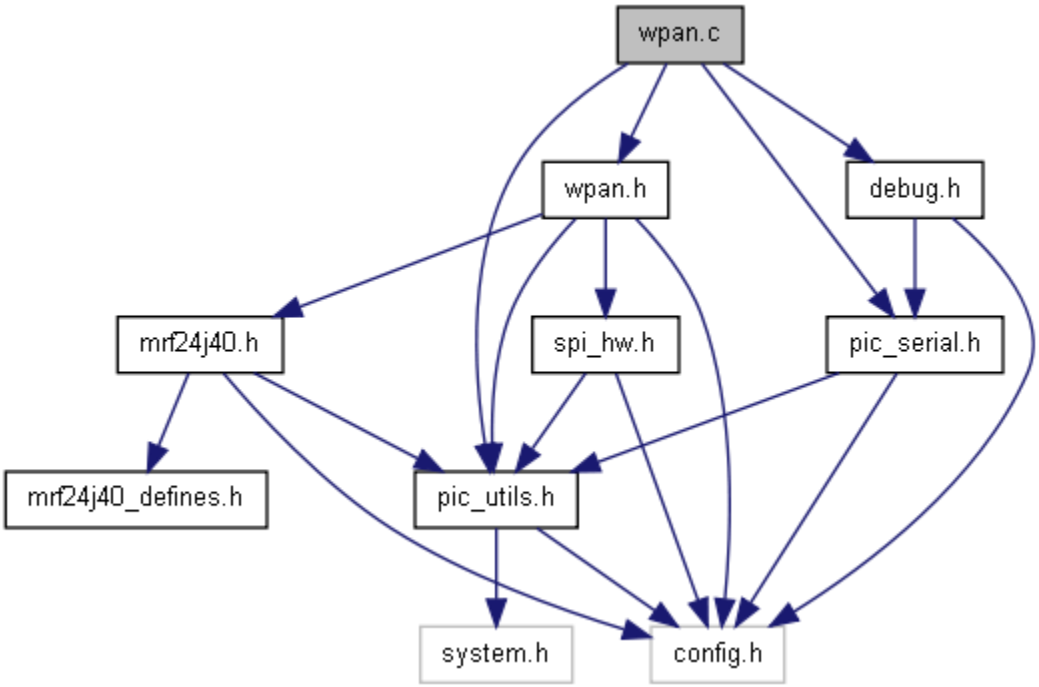
#define req_SET_PROTOCOL 0x0b

#define req_SET_REPORT 0x09

---

wpan.c File Reference

Wireless Personal Area Network routines.
Include dependency graph for wpan.c:



Defines

- #define debug_on

Functions

- void mrf24j40_receive_callback ()
- *Callback is actioned when mrf24j40 has a packet received off air.* void mrf24j40_transmit_callback (uns8 status, uns8 retries, uns8 channel_busy)
- *Callback is actioned when mrf24j40 has finished transmitting a packet, or failed to do so.* void wpan_handle_receive ()
- *Handle reception of packet.* void wpan_init ()
- *Initialise this and lower layers ready for use.* void wpan_print_address (wpan_address *addr)
- *Print the address of the packet.* void wpan_print_frame_type (uns8 frame_type)
- *Print information about the WPAN frame type.* void wpan_print_mac_command (uns8 *data)
- *Debug routine that prints out the mac command type.* void wpan_process ()
- *Process WPAN layer.* void wpan_setup_io ()

Setup IO as requred. Variables

- uns8 pkt_received = 0
- uns8 receive_lost = 0
- uns8 wpan_rx_buffer [50]
- uns8 wpan_rx_count

Detailed Description

Define Documentation

#define debug_on

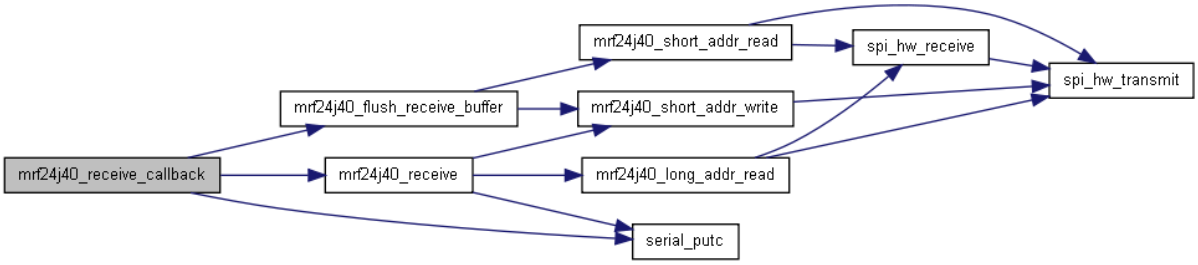
Function Documentation

void mrf24j40_receive_callback ()

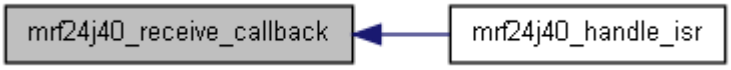
Callback indicating the mrf24j40 has a packet ready.

```
56                                  {
57     serial_putc('r');
58     if (pkt_received) {
59         receive_lost++;
60         debug_str("<Lost receive>");
61         mrf24j40_flush_receive_buffer();
62     } else {
63         pkt_received++;
64         wpan_rx_count = mrf24j40_receive(&wpan_rx_buffer, sizeof(wpan_rx_buffer));
65 
66         debug_str("Rx: ");
67         debug_int(wpan_rx_count);
68         debug_str(" bytes ");
69     }
70 
71     // this will get picked up in wpan process
72 
73 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



void mrf24j40_transmit_callback (uns8 *status*, uns8 *retries*, uns8 *channel_busy*)

Callback indicating the mrf24j40 has finished the transmission sequence.

Parameters:

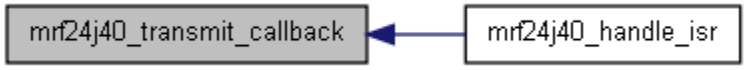
status Set to 0 for success or 1 for failure
retries Set to the number of retries
channel_busy Set to 1 if failure was due to the channel being busy.

```
77                                                          { // 1 if fail due to channel busy
78     debug_str(" <Tx: ");
79     if (!status) {
80         debug_str(" Good");
81     } else {
82         debug_str(" Fail");
83         if (channel_busy) {
84             debug_str(" (channel busy)");
```

```
85          }
86      }
87      debug_str(" Retries: ");
88      debug_int(retries);
89      debug_str(">\n");
90
91      wpan_data_transmitted_callback(status, retries, channel_busy);
92
93  }
```

Here is the call graph for this function:

Here is the caller graph for this function:



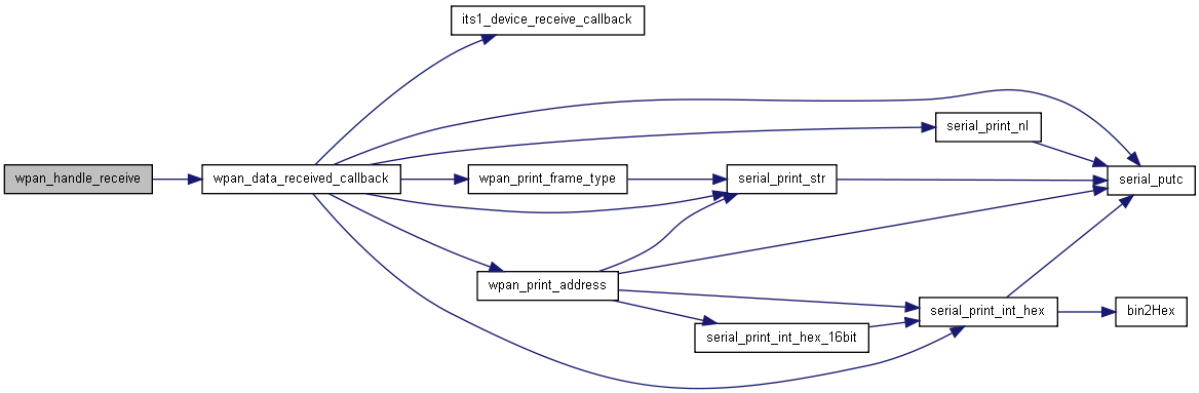
void wpan_handle_receive ()

Internal routine for WPAN layer to handle the reception of a packet.

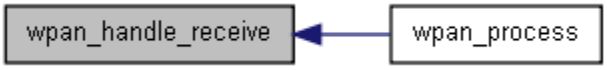
```
225                             {
226
227 wpan_address addr;
228
229 uns8 temp;
230 uns8 count;
231 uns8 frame_type;
232 uns8 buffer_pos;
233
234
235     // destination address
236     addr.dest_address_type = wpan_rx_buffer[1] >> 6;    // fc lsb dest address type
237
238     buffer_pos = 0;
239
240     switch (addr.dest_address_type) {
241         case WPAN_ADDR_TYPE_EXTENDED: // extended address
242
243             // get dest pan
244             addr.dest_pan_id = wpan_rx_buffer[4];   // msb
245             addr.dest_pan_id <<= 8;
246             addr.dest_pan_id += wpan_rx_buffer[3];  // lsb
247             // get extended address
248             for (count = 0; count < 8; count++) {
249                 addr.dest_ea[count] = wpan_rx_buffer[5+count];
250             }
251             // might be faster to unwind this loop...
252             buffer_pos = 13;
253             break;
254         case WPAN_ADDR_TYPE_SHORT: // short address
255             // get dest pan
256             addr.dest_pan_id = wpan_rx_buffer[4];   // msb
257             addr.dest_pan_id <<= 8;
258             addr.dest_pan_id += wpan_rx_buffer[3];  // lsb
259             // get short address
260             addr.dest_sa = wpan_rx_buffer[6];   // msb
261             addr.dest_sa <<= 8;
262             addr.dest_sa += wpan_rx_buffer[5];  // lsb
263             buffer_pos = 7;
264             break;
265         case WPAN_ADDR_TYPE_NONE: // no address
266             buffer_pos = 3;
267             break;
```

```
268     }
269
270     // source address
271
272     addr.source address type = (wpan rx buffer[1] >> 2) & 0b00000011;    // fc lsb src address
type
273
274     switch (addr.source address type) {
275         case WPAN ADDR TYPE EXTENDED: // extended address
276             // get source pan
277             if (test bit(wpan rx buffer[0], 6)) {    // pan id compression
278                 addr.source pan id = addr.dest pan id;
279             } else {
280                 addr.source pan id = wpan rx buffer[buffer_pos+1];  // msb
281                 addr.source pan id <<= 8;
282                 addr.source pan id += wpan rx buffer[buffer_pos];   // lsb
283                 buffer_pos += 2;
284             }
285             // get extended address
286             for (count = 0; count < 8; count++) {
287                 addr.source ea[count] = wpan rx buffer[buffer_pos++];
288             }
289
290             break;
291         case WPAN ADDR TYPE SHORT: // short address
292             if (test_bit(wpan rx buffer[0], 6)) {    // pan id compression
293                 addr.source pan id = addr.dest pan id;
294             } else {
295                 // get pan
296                 addr.source pan id = wpan rx buffer[buffer_pos+1];  // msb
297                 addr.source pan id <<= 8;
298                 addr.source pan id += wpan rx buffer[buffer pos];   // lsb
299                 buffer_pos += 2;
300             }
301             // get short address
302             addr.source sa = wpan rx buffer[buffer_pos+1];  // msb
303             addr.source sa <<= 8;
304             addr.source sa += wpan rx buffer[buffer_pos];   // lsb
305             buffer_pos += 2;
306             break;
307         case WPAN ADDR TYPE NONE: // no address
308             break;
309     }
310     //debug var(" buffer pos = ", buffer pos);
311     //debug_var(" wpan_rx_count = ", wpan_rx_count);
312     // data should be at buffer_pos
313     frame_type = wpan rx buffer[0] & 0b00000111;    // frame type
314     switch(frame_type) {
315         case FRAME TYPE MAC COMMAND:
316             //wpan_handle_mac_command(&addr, &wpan_rx_buffer[buffer_pos], wpan_rx_count -
buffer_pos);
317             break;
318         case FRAME TYPE BEACON:
319             //wpan_handle_beacon(&addr, &wpan_rx_buffer[buffer_pos], wpan_rx_count -
buffer_pos);
320             break;
321         case FRAME TYPE DATA:
322             wpan data received callback(&addr, &wpan rx buffer[buffer_pos], wpan rx count -
buffer_pos - 4,
323                                         /* lqi */ wpan rx buffer[wpan rx count-2], /* rssi
*/ wpan rx buffer[wpan rx count-1]);
324             break;
325         // We'll never see an ACK frame, that should be handled by the MRF
326     }
327
328     pkt received = 0;
329 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

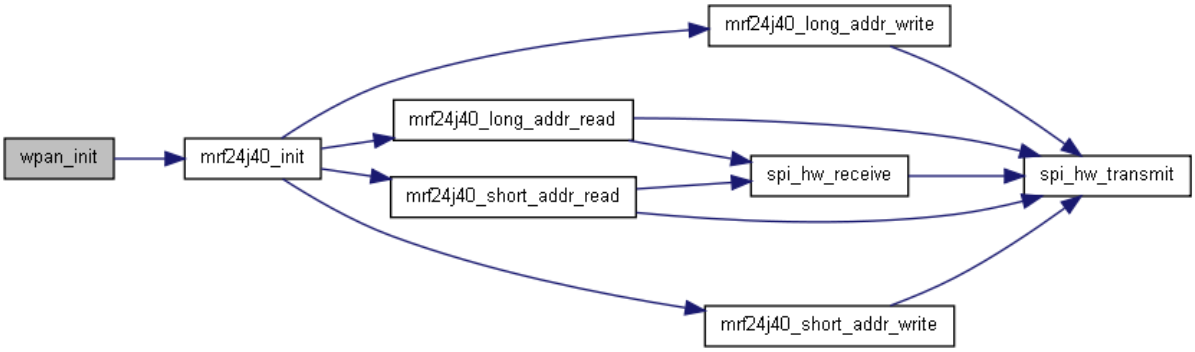


void wpan_init ()

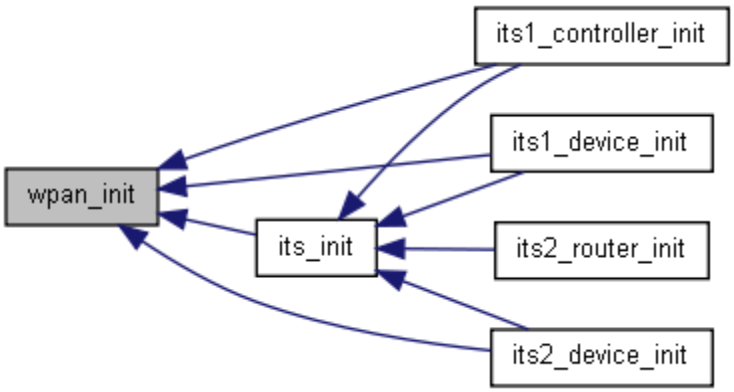
Initialise underlying hardware (typically mrf24j40)

```
336                     {
337
338     mrf24j40_init();
339
340 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

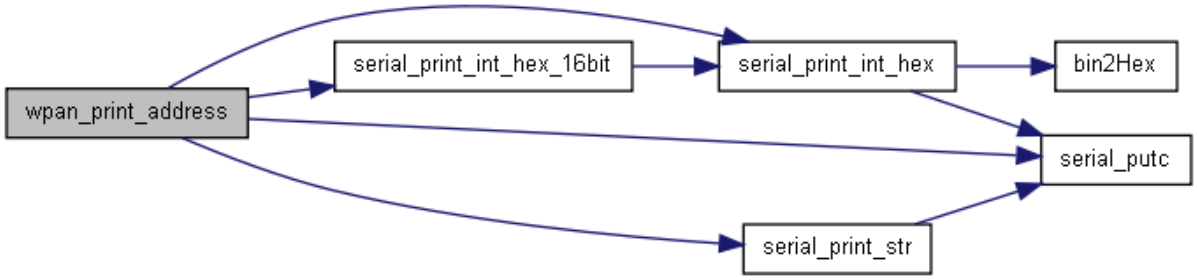


void wpan_print_address (wpan_address * *addr*)

Print out the source and destination address

```
96                                                  {
97
98 uns8 count;
99
100     debug_str("\nPKT:\nSrc ");
101     if (addr->source_address_type) { // ie, not 0
102         serial_print_str(" PAN: ");
103         serial_print_int_hex_16bit(addr->source_pan_id);
104
105         switch (addr->source_address_type) {
106             case WPAN_ADDR_TYPE_EXTENDED:
107                 serial_print_str(" EA: ");
108                 for (count = 7; count != 255; count--) {
109                     serial_print_int_hex(addr->source_ea[count]);
110                     serial_putc(' ');
111                 }
112                 break;
113             case WPAN_ADDR_TYPE_SHORT:
114                 serial_print_str(" SA: ");
115                 serial_print_int_hex_16bit(addr->source_sa);
116                 serial_putc(' ');
117                 break;
118         }
119     } else {
120         serial_print_str(" (blank) ");
121     }
122
123     serial_print_str(" Dest ");
124
125     if (addr->dest_address_type) { // ie, not 0
126         serial_print_str(" PAN: ");
127         serial_print_int_hex_16bit(addr->dest_pan_id);
128
129         switch (addr->dest_address_type) {
130             case WPAN_ADDR_TYPE_EXTENDED:
131                 serial_print_str(" EA: ");
132                 for (count = 7; count != 255; count--) {
133                     serial_print_int_hex(addr->dest_ea[count]);
134                     serial_putc(' ');
135                 }
136                 break;
137             case WPAN_ADDR_TYPE_SHORT:
138                 serial_print_str(" SA: ");
139                 serial_print_int_hex_16bit(addr->dest_sa);
140                 serial_putc(' ');
141                 break;
142         }
143     } else {
144         serial_print_str(" (blank) ");
145     }
146 }
```

Here is the call graph for this function:

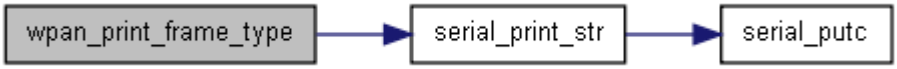


Here is the caller graph for this function:

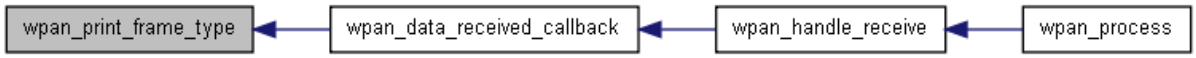
void wpan_print_frame_type (uns8 *frame_type*)

```
148                                             {
149
150     switch(frame type) {
151         case FRAME TYPE BEACON:
152             serial print str(" Beacon ");
153             break;
154         case FRAME TYPE DATA:
155             serial print str(" Data ");
156             break;
157         case FRAME TYPE ACK:
158             serial print str(" Ack ");
159             break;
160         case FRAME TYPE MAC COMMAND:
161             serial print str(" Mac ");
162             break;
163     }
164 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



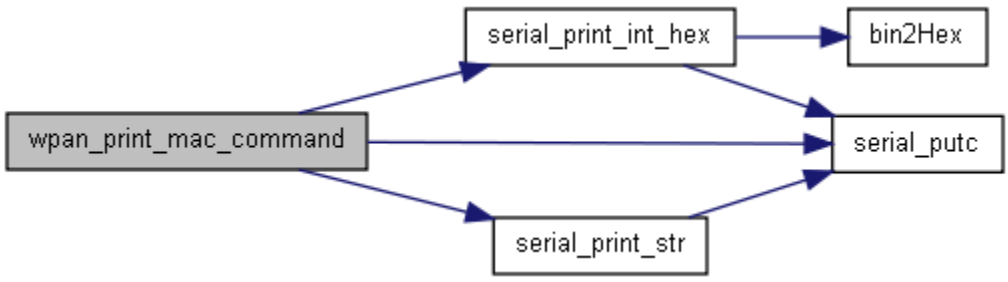
void wpan_print_mac_command (uns8 * *data*)

Print the MAC layer command details

```
166                                         {
167
168 uns8 mac_command = data[0]; // Uses less flash this way
169
170     switch (data[0]) {
171
172         case MAC CMD ASSOC REQ:
173             serial print str("ASSOC REQ (Capability: ");
174             serial print int hex(data[1]);
175             serial putc(')');
176             break;
177         case MAC CMD ASSOC RES:
```

```
178             serial_print_str("ASSOC RES (Short addr: ");
179             serial_print_int_hex(data[2]);  // msb
180             serial_print_int_hex(data[1]);  // lsb
181             serial_print_str(" Status: ");
182             serial_print_int_hex(data[3]);
183             serial_putc(')');
184             break;
185         case MAC_CMD_DISASSOC:
186             serial_print_str("DISASSOC (Reason: ");
187             serial_print_int_hex(data[1]);
188             serial_putc(')');
189             break;
190         case MAC_CMD_DATA_REQ:
191             serial_print_str("DATA REQ");
192             break;
193         case MAC_CMD_PAN_ID_CONFLICT:
194             serial_print_str("PAN ID CONFLICT");
195             break;
196         case MAC_CMD_ORPHAN:
197             serial_print_str("ORPHAN");
198             break;
199         case MAC_CMD_BEACON_REQ:
200             serial_print_str("BEACON REQ");
201             break;
202         case MAC_CMD_COORD_REALIGN:
203             serial_print_str("COORD REALIGN (data TBD)");
204             break;
205         case MAC_CMD_GTS_REQ:
206             serial_print_str("GTS REQ (Char: ");
207             serial_print_int_hex(data[1]);
208             serial_putc(')');
209             break;
210     }
211 }
```

Here is the call graph for this function:

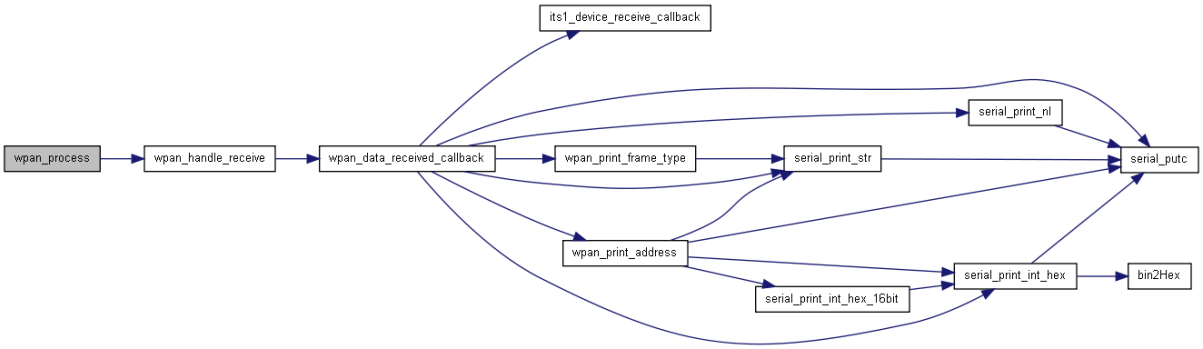


void wpan_process ()

Call wpan_process() regularly to handle events at the WPAN layer. Typically this includes handling any received packets, processing them, and handing up to a higher layer.

```
214                     {
215 
216     if (pkt_received) {
217         wpan_handle_receive();
218     }
219 }
```

Here is the call graph for this function:



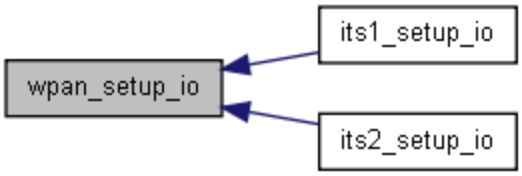
void wpan_setup_io ()

Set up ports and pins for WPAN use

```
332                         {
333     mrf24j40_setup_io();
334 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



Variable Documentation

uns8 pkt_received = 0

uns8 receive_lost = 0

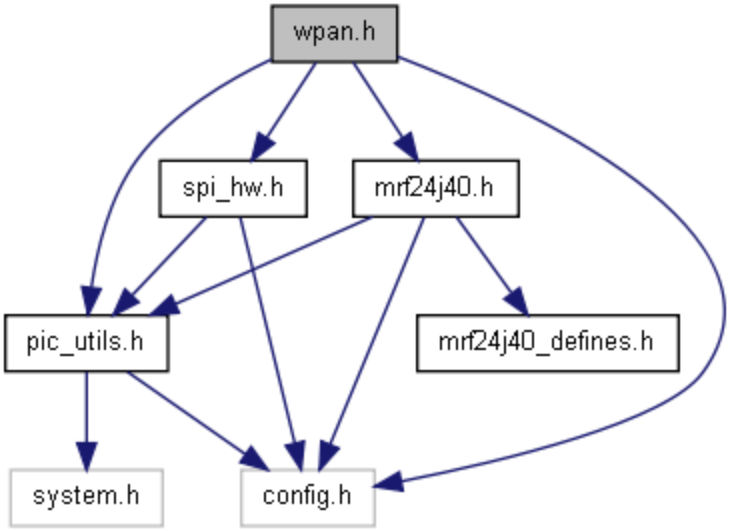
uns8 wpan_rx_buffer[50]

uns8 wpan_rx_count

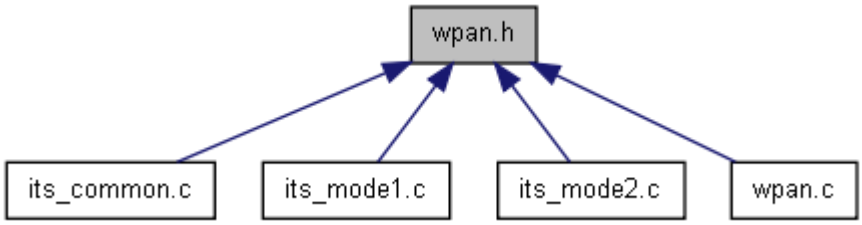
---

wpan.h File Reference

Wireless Personal Area Network routines.
Include dependency graph for wpan.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct wpan_address

Defines

- #define FRAME_TYPE_ACK 0b010
- #define FRAME_TYPE_BEACON 0b000
- #define FRAME_TYPE_DATA 0b001
- #define FRAME_TYPE_MAC_COMMAND 0b011
- #define MAC_CMD_ASSOC_REQ 0x01
- #define MAC_CMD_ASSOC_RES 0x02
- #define MAC_CMD_BEACON_REQ 0x07
- #define MAC_CMD_COORD_REALIGN 0x08
- #define MAC_CMD_DATA_REQ 0x04
- #define MAC_CMD_DISASSOC 0x03
- #define MAC_CMD_GTS_REQ 0x09
- #define MAC_CMD_ORPHAN 0x06
- #define MAC_CMD_PAN_ID_CONFLICT 0x05
- #define WPAN_ADDR_TYPE_EXTENDED 0b00000011
- #define WPAN_ADDR_TYPE_NONE 0b00000000
- #define WPAN_ADDR_TYPE_SHORT 0b00000010

Functions

- void wpan_data_received_callback (wpan_address *addr, uns8 *data, uns8 data_size, uns8 lqi, uns8 rssi)

- *Callback for data received.* void wpan_data_transmitted_callback (uns8 status, uns8 retries, uns8 channel_busy)
- *Callback for data transmitted.* void wpan_handle_receive ()
- *Handle reception of packet.* void wpan_init ()
- *Initialise this and lower layers ready for use.* void wpan_print_address (wpan_address *addr)
- *Print the address of the packet.* void wpan_print_frame_type (uns8 frame_type)
- *Print information about the WPAN frame type.* void wpan_print_mac_command (uns8 *data)
- *Debug routine that prints out the mac command type.* void wpan_process ()
- *Process WPAN layer.* void wpan_setup_io ()

Setup IO as requred. Variables

- uns8 pkt_received

Detailed Description

Put this in your config.h

```
// ------------------------
// WPAN (802.15.4) wireless
// ------------------------

#define WPAN_USE_MRF24J50
// ------------------------
```

Define Documentation

#define FRAME_TYPE_ACK 0b010

#define FRAME_TYPE_BEACON 0b000

#define FRAME_TYPE_DATA 0b001

#define FRAME_TYPE_MAC_COMMAND 0b011

#define MAC_CMD_ASSOC_REQ 0x01

#define MAC_CMD_ASSOC_RES 0x02

#define MAC_CMD_BEACON_REQ 0x07

#define MAC_CMD_COORD_REALIGN 0x08

#define MAC_CMD_DATA_REQ 0x04

#define MAC_CMD_DISASSOC 0x03

#define MAC_CMD_GTS_REQ 0x09

#define MAC_CMD_ORPHAN 0x06

#define MAC_CMD_PAN_ID_CONFLICT 0x05

#define WPAN_ADDR_TYPE_EXTENDED 0b00000011

#define WPAN_ADDR_TYPE_NONE 0b00000000

#define WPAN_ADDR_TYPE_SHORT 0b00000010

Function Documentation

void wpan_data_received_callback (wpan_address * *addr*, uns8 * *data*, uns8 *data_size*, uns8 *lqi*, uns8 *rssi*)

After the packet has been processed and the address extracted, it is delivered to the callback function.

void wpan_data_transmitted_callback (uns8 *status*, uns8 *retries*, uns8 *channel_busy*)

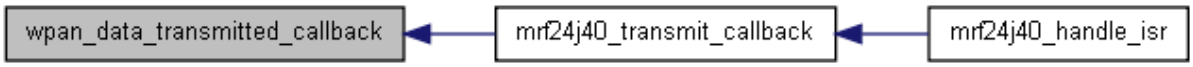
Once the lower layers have attempted to transmit the packet, the results will be presented to the callback.

Parameters:

status Set to 0 for success or 1 for failure
retries Set to the number of retries
channel_busy Set to 1 if failure was due to the channel being busy.

```
804                                                                     {
805     its2_transmitting = 0;
806 }
```

Here is the caller graph for this function:



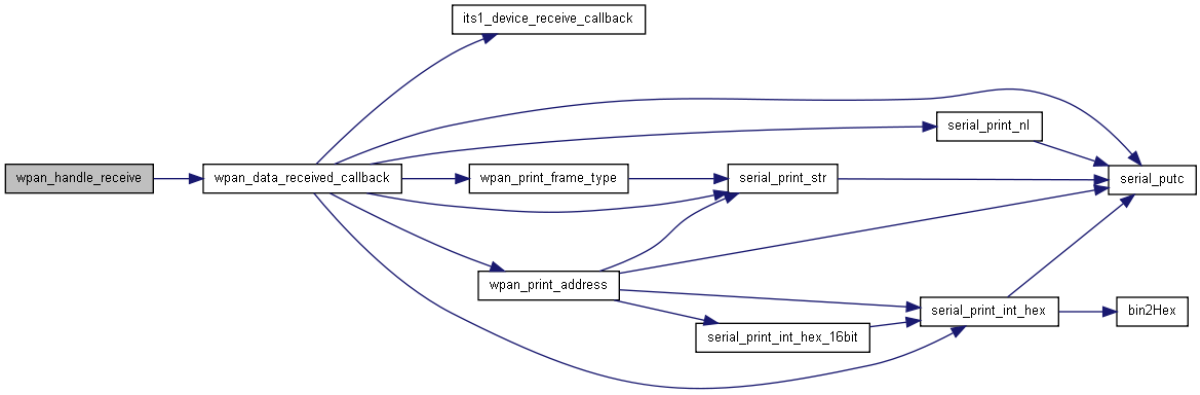
void wpan_handle_receive ()

Internal routine for WPAN layer to handle the reception of a packet.

```
225                             {
226
227 wpan_address addr;
228
229 uns8 temp;
230 uns8 count;
231 uns8 frame_type;
232 uns8 buffer_pos;
233
234
235     // destination address
236     addr.dest_address_type = wpan_rx_buffer[1] >> 6;    // fc lsb dest address type
237
238     buffer_pos = 0;
239
240     switch (addr.dest_address_type) {
241         case WPAN_ADDR_TYPE_EXTENDED: // extended address
242
243             // get dest pan
244             addr.dest_pan_id = wpan_rx_buffer[4];   // msb
245             addr.dest_pan_id <<= 8;
246             addr.dest_pan_id += wpan_rx_buffer[3];  // lsb
247             // get extended address
248             for (count = 0; count < 8; count++) {
249                 addr.dest_ea[count] = wpan_rx_buffer[5+count];
250             }
251             // might be faster to unwind this loop...
252             buffer_pos = 13;
253             break;
254         case WPAN_ADDR_TYPE_SHORT: // short address
255             // get dest pan
256             addr.dest_pan_id = wpan_rx_buffer[4];   // msb
257             addr.dest_pan_id <<= 8;
258             addr.dest_pan_id += wpan_rx_buffer[3];  // lsb
259             // get short address
260             addr.dest_sa = wpan_rx_buffer[6];   // msb
261             addr.dest_sa <<= 8;
262             addr.dest_sa += wpan_rx_buffer[5];  // lsb
263             buffer_pos = 7;
264             break;
265         case WPAN_ADDR_TYPE_NONE: // no address
266             buffer_pos = 3;
267             break;
268     }
269
270     // source address
271
272     addr.source_address_type = (wpan_rx_buffer[1] >> 2) & 0b00000011;    // fc lsb src address
type
273
274     switch (addr.source_address_type) {
275         case WPAN_ADDR_TYPE_EXTENDED: // extended address
276             // get source pan
277             if (test_bit(wpan_rx_buffer[0], 6)) {   // pan id compression
278                 addr.source_pan_id = addr.dest_pan_id;
279             } else {
280                 addr.source_pan_id = wpan_rx_buffer[buffer_pos+1];  // msb
281                 addr.source_pan_id <<= 8;
282                 addr.source_pan_id += wpan_rx_buffer[buffer_pos];   // lsb
283                 buffer_pos += 2;
284             }
285             // get extended address
```

```
286             for (count = 0; count < 8; count++) {
287                 addr.source_ea[count] = wpan_rx_buffer[buffer_pos++];
288             }
289 
290             break;
291         case WPAN_ADDR_TYPE_SHORT: // short address
292             if (test_bit(wpan_rx_buffer[0], 6)) {   // pan id compression
293                 addr.source_pan_id = addr.dest_pan_id;
294             } else {
295                 // get pan
296                 addr.source_pan_id = wpan_rx_buffer[buffer_pos+1];  // msb
297                 addr.source_pan_id <<= 8;
298                 addr.source_pan_id += wpan_rx_buffer[buffer_pos];   // lsb
299                 buffer_pos += 2;
300             }
301             // get short address
302             addr.source_sa = wpan_rx_buffer[buffer_pos+1];  // msb
303             addr.source_sa <<= 8;
304             addr.source_sa += wpan_rx_buffer[buffer_pos];   // lsb
305             buffer_pos += 2;
306             break;
307         case WPAN_ADDR_TYPE_NONE: // no address
308             break;
309     }
310     //debug_var(" buffer pos = ", buffer_pos);
311     //debug_var(" wpan_rx_count = ", wpan_rx_count);
312     // data should be at buffer_pos
313     frame_type = wpan_rx_buffer[0] & 0b00000111;    // frame type
314     switch(frame_type) {
315         case FRAME_TYPE_MAC_COMMAND:
316             //wpan_handle_mac_command(&addr, &wpan_rx_buffer[buffer_pos], wpan_rx_count -
buffer_pos);
317             break;
318         case FRAME_TYPE_BEACON:
319             //wpan_handle_beacon(&addr, &wpan_rx_buffer[buffer_pos], wpan_rx_count -
buffer_pos);
320             break;
321         case FRAME_TYPE_DATA:
322             wpan_data_received_callback(&addr, &wpan_rx_buffer[buffer_pos], wpan_rx_count -
buffer_pos - 4,
323                                         /* lqi */ wpan_rx_buffer[wpan_rx_count-2], /* rssi
*/ wpan_rx_buffer[wpan_rx_count-1]);
324             break;
325         // We'll never see an ACK frame, that should be handled by the MRF
326     }
327 
328     pkt_received = 0;
329 }
```

Here is the call graph for this function:



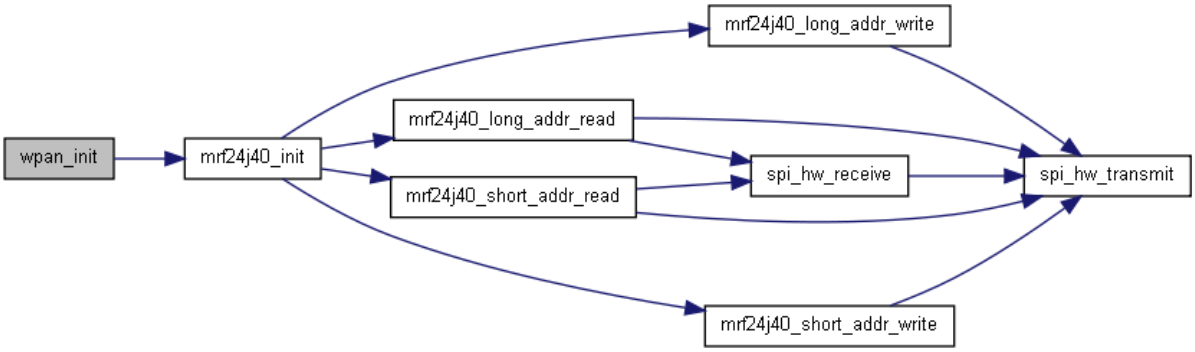
Here is the caller graph for this function:

void wpan_init ()

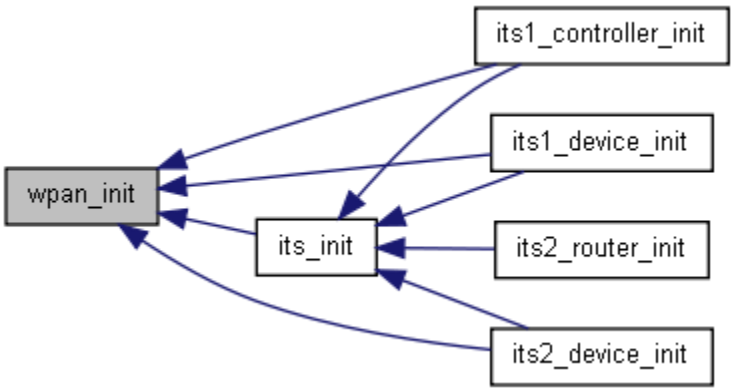
Initialise underlying hardware (typically mrf24j40)

```
336                     {
337
338     mrf24j40_init();
339
340 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



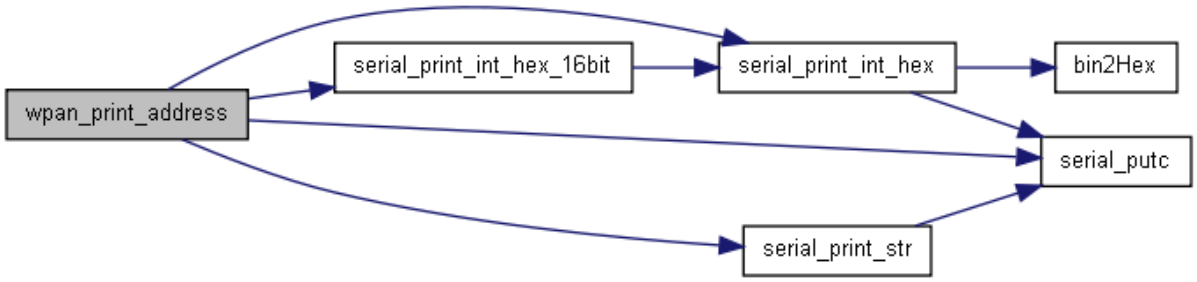
void wpan_print_address (wpan_address * *addr*)

Print out the source and destination address

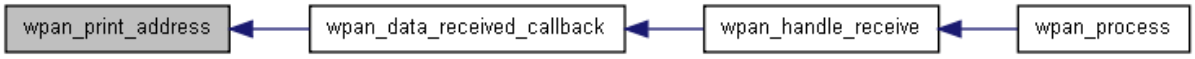
```
96                                              {
97
98 uns8 count;
99
100     debug_str("\nPKT:\nSrc ");
101     if (addr->source_address_type) { // ie, not 0
102         serial_print_str(" PAN: ");
103         serial_print_int_hex_16bit(addr->source_pan_id);
104
105         switch (addr->source_address_type) {
106             case WPAN_ADDR_TYPE_EXTENDED:
107                 serial_print_str(" EA: ");
108                 for (count = 7; count != 255; count--) {
```

```
109                     serial_print_int_hex(addr->source_ea[count]);
110                     serial_putc(' ');
111                 }
112                 break;
113             case WPAN_ADDR_TYPE_SHORT:
114                 serial_print_str(" SA: ");
115                 serial_print_int_hex_16bit(addr->source_sa);
116                 serial_putc(' ');
117                 break;
118         }
119     } else {
120         serial_print_str(" (blank) ");
121     }
122 
123     serial_print_str(" Dest ");
124 
125     if (addr->dest_address_type) { // ie, not 0
126         serial_print_str(" PAN: ");
127         serial_print_int_hex_16bit(addr->dest_pan_id);
128 
129         switch (addr->dest_address_type) {
130             case WPAN_ADDR_TYPE_EXTENDED:
131                 serial_print_str(" EA: ");
132                 for (count = 7; count != 255; count--) {
133                     serial_print_int_hex(addr->dest_ea[count]);
134                     serial_putc(' ');
135                 }
136                 break;
137             case WPAN_ADDR_TYPE_SHORT:
138                 serial_print_str(" SA: ");
139                 serial_print_int_hex_16bit(addr->dest_sa);
140                 serial_putc(' ');
141                 break;
142         }
143     } else {
144         serial_print_str(" (blank) ");
145     }
146 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

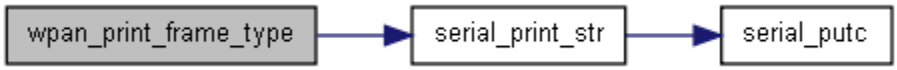


void wpan_print_frame_type (uns8 *frame_type*)

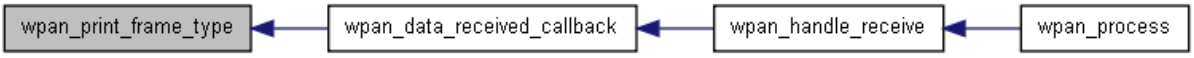
```
148                                         {
149 
150     switch(frame_type) {
151         case FRAME_TYPE_BEACON:
152             serial_print_str(" Beacon ");
153             break;
```

```
154         case FRAME_TYPE_DATA:
155             serial_print_str(" Data ");
156             break;
157         case FRAME_TYPE_ACK:
158             serial_print_str(" Ack ");
159             break;
160         case FRAME_TYPE_MAC_COMMAND:
161             serial_print_str(" Mac ");
162             break;
163     }
164 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



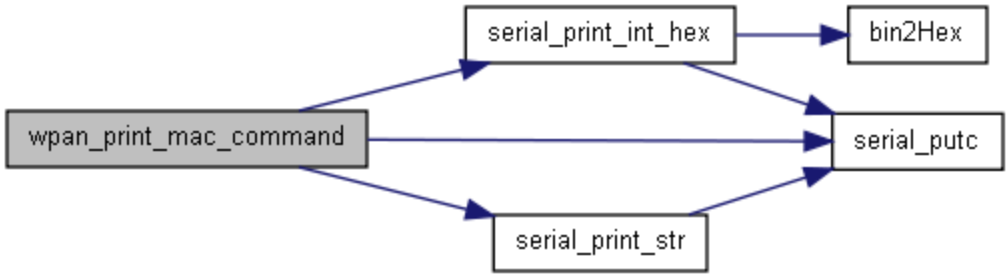
void wpan_print_mac_command (uns8 * *data*)

Print the MAC layer command details

```
166                                         {
167 
168 uns8 mac_command = data[0]; // Uses less flash this way
169 
170     switch (data[0]) {
171 
172         case MAC_CMD_ASSOC_REQ:
173             serial_print_str("ASSOC REQ (Capability: ");
174             serial_print_int_hex(data[1]);
175             serial_putc(')');
176             break;
177         case MAC_CMD_ASSOC_RES:
178             serial_print_str("ASSOC RES (Short addr: ");
179             serial_print_int_hex(data[2]);  // msb
180             serial_print_int_hex(data[1]);  // lsb
181             serial_print_str(" Status: ");
182             serial_print_int_hex(data[3]);
183             serial_putc(')');
184             break;
185         case MAC_CMD_DISASSOC:
186             serial_print_str("DISASSOC (Reason: ");
187             serial_print_int_hex(data[1]);
188             serial_putc(')');
189             break;
190         case MAC_CMD_DATA_REQ:
191             serial_print_str("DATA REQ");
192             break;
193         case MAC_CMD_PAN_ID_CONFLICT:
194             serial_print_str("PAN ID CONFLICT");
195             break;
196         case MAC_CMD_ORPHAN:
197             serial_print_str("ORPHAN");
198             break;
199         case MAC_CMD_BEACON_REQ:
200             serial_print_str("BEACON REQ");
201             break;
202         case MAC_CMD_COORD_REALIGN:
203             serial_print_str("COORD REALIGN (data TBD)");
204             break;
205         case MAC_CMD_GTS_REQ:
206             serial_print_str("GTS REQ (Char: ");
207             serial_print_int_hex(data[1]);
```

```
208             serial_putc(')');
209             break;
210     }
211 }
```

Here is the call graph for this function:

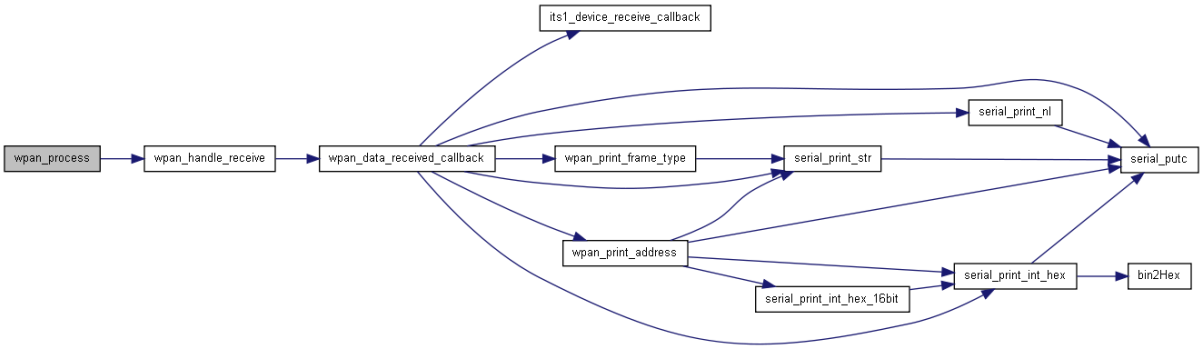


void wpan_process ()

Call wpan_process() regularly to handle events at the WPAN layer. Typically this includes handling any received packets, processing them, and handing up to a higher layer.

```
214                     {
215
216     if (pkt_received) {
217         wpan_handle_receive();
218     }
219 }
```

Here is the call graph for this function:



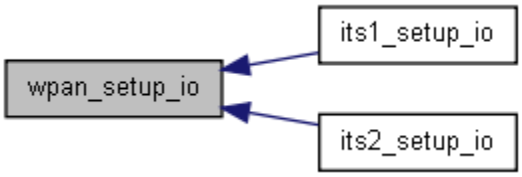
void wpan_setup_io ()

Set up ports and pins for WPAN use

```
332                     {
333     mrf24j40_setup_io();
334 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



Variable Documentation

uns8 pkt_received

Index