uC

0.1

Generated by Doxygen 1.5.8

Wed May 19 20:56:57 2010

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Module Documentation

4.1 Driver unit version 1

Main file of the driver unit.

Files

- file board.h

  Board specific defines.

- file init.c

  Initialization routines for the driver unit.

- file init.h

  Initialization routines for the driver unit.

- file main.c

  Main file of the driver unit.

4.1.1 Detailed Description

Main file of the driver unit.

Author:

Mikael Larsmark, SM2WMV

Date:

2008-04-06

```
#include "driver_unit/main.h"
```

4.2 Driver unit version 2

Main file of the driver unit.

Files

- file board.h

  Driver unit board defines.

- file init.c

  Initialization routines for the driver unit.

- file init.h

  Initialization routines for the driver unit.

- file main.c

  Main file of the driver unit.

4.2.1 Detailed Description

Main file of the driver unit.

Author:

Mikael Larsmark, SM2WMV

Date:

2009-03-16

```
#include "driver_unit_v2/main.h"
```

4.3 Event QUEUE library

Files

- file event_queue.c

  Event queue.

4.3.1 Detailed Description

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "event_queue.h"
```

4.4 Front panel

Files

- file antenna_ctrl.c
  Antenna control functions.
- file antenna_ctrl.h
  Antenna control functions.
- file band_ctrl.c
  Band control functions.
- file band_ctrl.h
  Band control functions.
- file board.h
  Front panel board defines.
- file computer_interface.c
  Interface towards the computer.
- file computer_interface.h
  Interface towards the computer.
- file display.c
  The serial interface to configure the device and control it.
- file display.h
  The serial interface to configure the device and control it.
- file ds1307.c
  Main file of the front panel.
- file ds1307.h
  Realtime clock.
- file eeprom.c
  EEPROM functions.
- file eeprom.h
  EEPROM functions.
- file eeprom_m24.c
  EEPROM hardware functions.
- file eeprom_m24.h
  EEPROM hardware functions.
- file errors.h

List of error codes.

- file event_handler.c

  Event handler of various things.

- file event_handler.h

  Event handler of various things.

- file init.c

  Initialization routines for the front panel.

- file init.h

  Initialization routines for the front panel.

- file interrupt_handler.c

  Handles different external interrupts.

- file interrupt_handler.h

  Handles different external interrupts.

- file led_control.c

  Front panel LED control functions.

- file led_control.h

  Front panel LED control functions.

- file main.c

  Main file of the front panel.

- file menu.c

  Menu system.

- file menu.h

  Menu system.

- file pictures.h

  Pictures which can be viewed on the display.

- file powermeter.c

  Power meter.

- file powermeter.h

  Power meter functions.

- file radio_interface.c

  Radio interface, such as PTT AMP, PTT Radio, CAT etc.

- file radio_interface.h

  Radio interface, such as PTT AMP, PTT Radio, CAT etc.

- file remote_control.c

    Remote control of the openASC box.

- file remote_control.h

    Remote control of the openASC box.

- file rotary_encoder.c

    Rotary encoder functions.

- file rotary_encoder.h

    Rotary encoder functions.

- file sequencer.c

    Sequencer.

- file sequencer.h

    Sequencer.

- file sub_menu.c

    Antenna sub menu functions.

- file sub_menu.h

    Antenna sub menu functions.

- file usart.c

    USART routines.

- file usart.h

    USART routines.

Defines

- #define GLCD_LEFT 0
- #define GLCD_ON_CTRL 0x3E

4.4.1 Detailed Description

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/main.h"
```

4.4.2 Define Documentation

4.4.2.1 #define GLCD_LEFT 0

```
#include "glcd.h"
```

Overview

This library (or API) allows you to draw dots, lines, boxes, circles, and text on most monochrome graphic LCDs. An easily expandable font file (5x7-pixel characters) is provided for all basic ASCII characters (0x20-0x7F hex, 32-127 decimal). An expandable graphic font file is provided for defining specialty characters or custom icons. Because this library is designed to work with many different kinds of LCDs, it needs a graphic LCD driver such as ks0108.c to enable it to talk to the LCD.

Note:

For full text-output functionality, you may wish to use the rprintf functions along with this driver.

Definition at line 46 of file glcd.h.

4.4.2.2 #define GLCD_ON_CTRL 0x3E

```
#include "front_panel/ks0108.h"
```

Overview

This display driver performs the basic functions necessary to access any graphic LCD based on the KS0108 or HD61202 controller chip. For more advanced functions, use this driver in conjunction with glcd.c. KS0108/HD61202 displays typically range in size from 64x32 pixels to 128x128 pixels and up to 3" square. To determine whether a display is compatible, you should look for the above controller chips to be mounted on the PC board attached to the display glass. The controller chips are about 1/2" x 3/4" and have 80+ pins. On larger displays, you may also see slave LCD driver chips with the numbers KS0107 or HD61203. The display will likely have an 18 or 20-pin interface. The interface from the LCD to an AVR processor does not require any additional hardware. If you can locate a datasheet for your display, that plus the information in the ks0108conf.h file should be all you need to get hooked up.

Definition at line 49 of file ks0108.h.

Referenced by glcd_init().

4.5 General I/O card

Main file of the General I/O card.

Files

- file board.h

  General I/O board defines.

- file init.c

  Initialization routines for the General I/O card.

- file init.h

  Initialization routines for the General I/O card.

- file main.c

  Main file of the General I/O card.

4.5.1 Detailed Description

Main file of the General I/O card.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-05-18

```
#include "general_io/main.h"
```

4.6 I2C Serial Interface Function Library (i2c.c)

```
#include "i2c.h"
```

Overview

This library provides the high-level functions needed to use the I2C serial interface supported by the hardware of several AVR processors. The library is functional but has not been exhaustively tested yet and is still expanding. Thanks to the standardization of the I2C protocol and register access, the send and receive commands are everything you need to talk to thousands of different I2C devices including: EEPROMS, Flash memory, MP3 players, A/D and D/A converters, electronic potentiometers, etc.

About I2C

I2C (pronounced "eye-squared-see") is a two-wire bidirectional network designed for easy transfer of information between a wide variety of intelligent devices. Many of the Atmel AVR series processors have hardware support for transmitting and receiving using an I2C-type bus. In addition to the AVRs, there are thousands of other parts made by manufacturers like Philips, Maxim, National, TI, etc that use I2C as their primary means of communication and control. Common device types are A/D & D/A converters, temp sensors, intelligent battery monitors, MP3 decoder chips, EEPROM chips, multiplexing switches, etc.

I2C uses only two wires (SDA and SCL) to communicate bidirectionally between devices. I2C is a multidrop network, meaning that you can have several devices on a single bus. Because I2C uses a 7-bit number to identify which device it wants to talk to, you cannot have more than 127 devices on a single bus.

I2C ordinarily requires two 4.7K pull-up resistors to power (one each on SDA and SCL), but for small numbers of devices (maybe 1-4), it is enough to activate the internal pull-up resistors in the AVR processor. To do this, set the port pins, which correspond to the I2C pins SDA/SCL, high. For example, on the mega163, sbi(PORTC, 0); sbi(PORTC, 1);.

For complete information about I2C, see the Philips Semiconductor website. They created I2C and have the largest family of devices that work with I2C.

Many manufacturers market I2C bus devices under a different or generic bus name like "Two-Wire Interface". This is because Philips still holds "I2C" as a trademark. For example, SMBus and SMBus devices are hardware compatible and closely related to I2C. They can be directly connected to an I2C bus along with other I2C devices are are generally accessed in the same way as I2C devices. SMBus is often found on modern motherboards for temp sensing and other low-level control tasks.

4.7 Internal communication routines

Files

- file internal_comm.c

  The internal communication routines.

- file internal_comm_commands.h

  The internal communication commands.

- file internal_comm_rx_queue.c

  The internal communication RX QUEUE.

- file internal_comm_rx_queue.h

  The internal communication RX QUEUE.

- file internal_comm_tx_queue.c

  The internal communication TX QUEUE.

4.7.1 Detailed Description

When using these routines for the internal communication it's important to initialize the pointers for the transmit and recieve data before any of the other functions are used. This is done by using the void internal_comm_init(void (*func_ptr_rx)(UC_MESSAGE), void (*func_ptr_-tx)(char)); where func_ptr_rx and func_ptr_rx should point the functions which take the argument of UC_MESSAGE.

Doing it this way makes the routines adaptable do different hardware, you just change the routine for TX and RX of data.

When a message has been recieved it will be added to the RX queue and by polling communication by using internal_comm_poll_rx_queue(void) if there is a message in the queue it will get sent to the routine which was specified in the initialization routine.

You will also need to poll the internal_comm_poll_tx_queue() at x intervals so that messages are sent when the tx queue isn't empty.

4.8 Motherboard

Main file of the motherboard.

Files

- file board.h

  Motherboard defines.

- file computer_interface.c

  Interface towards the computer.

- file computer_interface.h

  Interface towards the computer.

- file init.c

  Initialization routines for the motherboard.

- file init.h

  Initialization routines for the motherboard.

- file main.c

  Main file of the motherboard.

- file usart.c

  Motherboard USART routines.

- file usart.h

  Motherboard USART routines.

4.8.1 Detailed Description

Main file of the motherboard.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "motherboard/main.h"
```

4.9 A/D Converter Function Library (a2d.c)

Defines

- #define ADC_PRESCALE_DIV2 0x00

  0x01,0x00 -> CPU clk/2

- #define ADC_PRESCALE_DIV4 0x02

  0x02 -> CPU clk/4

- #define ADC_PRESCALE_DIV8 0x03

  0x03 -> CPU clk/8

- #define ADC_PRESCALE_DIV16 0x04

  0x04 -> CPU clk/16

- #define ADC_PRESCALE_DIV32 0x05

  0x05 -> CPU clk/32

- #define ADC_PRESCALE_DIV64 0x06

  0x06 -> CPU clk/64

- #define ADC_PRESCALE_DIV128 0x07

  0x07 -> CPU clk/128

- #define **ADC_PRESCALE** ADC_PRESCALE_DIV64
- #define **ADC_PRESCALE_MASK** 0x07
- #define ADC_REFERENCE_AREF 0x00

  0x00 -> AREF pin, internal VREF turned off

- #define ADC_REFERENCE_AVCC 0x01

  0x01 -> AVCC pin, internal VREF turned off

- #define ADC_REFERENCE_RSVD 0x02

  0x02 -> Reserved

- #define ADC_REFERENCE_256V 0x03

  0x03 -> Internal 2.56V VREF

- #define **ADC_REFERENCE** ADC_REFERENCE_AVCC
- #define **ADC_REFERENCE_MASK** 0xC0
- #define **ADC_MUX_MASK** 0x1F
- #define **ADC_CH_ADC0** 0x00
- #define **ADC_CH_ADC1** 0x01
- #define **ADC_CH_ADC2** 0x02
- #define **ADC_CH_ADC3** 0x03
- #define **ADC_CH_ADC4** 0x04
- #define **ADC_CH_ADC5** 0x05
- #define **ADC_CH_ADC6** 0x06
- #define **ADC_CH_ADC7** 0x07

- #define ADC_CH_122V 0x1E

  1.22V voltage reference

- #define ADC_CH_AGND 0x1F

  AGND.

- #define **ADC_CH_0_0_DIFF10X** 0x08
- #define **ADC_CH_1_0_DIFF10X** 0x09
- #define **ADC_CH_0_0_DIFF200X** 0x0A
- #define **ADC_CH_1_0_DIFF200X** 0x0B
- #define **ADC_CH_2_2_DIFF10X** 0x0C
- #define **ADC_CH_3_2_DIFF10X** 0x0D
- #define **ADC_CH_2_2_DIFF200X** 0x0E
- #define **ADC_CH_3_2_DIFF200X** 0x0F
- #define **ADC_CH_0_1_DIFF1X** 0x10
- #define **ADC_CH_1_1_DIFF1X** 0x11
- #define **ADC_CH_2_1_DIFF1X** 0x12
- #define **ADC_CH_3_1_DIFF1X** 0x13
- #define **ADC_CH_4_1_DIFF1X** 0x14
- #define **ADC_CH_5_1_DIFF1X** 0x15
- #define **ADC_CH_6_1_DIFF1X** 0x16
- #define **ADC_CH_7_1_DIFF1X** 0x17
- #define **ADC_CH_0_2_DIFF1X** 0x18
- #define **ADC_CH_1_2_DIFF1X** 0x19
- #define **ADC_CH_2_2_DIFF1X** 0x1A
- #define **ADC_CH_3_2_DIFF1X** 0x1B
- #define **ADC_CH_4_2_DIFF1X** 0x1C
- #define **ADC_CH_5_2_DIFF1X** 0x1D
- #define ADC_PRESCALE_DIV2 0x00

  0x01,0x00 -> CPU clk/2

- #define ADC_PRESCALE_DIV4 0x02

  0x02 -> CPU clk/4

- #define ADC_PRESCALE_DIV8 0x03

  0x03 -> CPU clk/8

- #define ADC_PRESCALE_DIV16 0x04

  0x04 -> CPU clk/16

- #define ADC_PRESCALE_DIV32 0x05

  0x05 -> CPU clk/32

- #define ADC_PRESCALE_DIV64 0x06

  0x06 -> CPU clk/64

- #define ADC_PRESCALE_DIV128 0x07

  0x07 -> CPU clk/128

- #define **ADC_PRESCALE** ADC_PRESCALE_DIV64

- #define **ADC_PRESCALE_MASK** 0x07
- #define ADC_REFERENCE_AREF 0x00

  0x00 -> AREF pin, internal VREF turned off

- #define ADC_REFERENCE_AVCC 0x01

  0x01 -> AVCC pin, internal VREF turned off

- #define ADC_REFERENCE_RSVD 0x02

  0x02 -> Reserved

- #define ADC_REFERENCE_256V 0x03

  0x03 -> Internal 2.56V VREF

- #define **ADC_REFERENCE** ADC_REFERENCE_AVCC
- #define **ADC_REFERENCE_MASK** 0xC0
- #define **ADC_MUX_MASK** 0x1F
- #define **ADC_CH_ADC0** 0x00
- #define **ADC_CH_ADC1** 0x01
- #define **ADC_CH_ADC2** 0x02
- #define **ADC_CH_ADC3** 0x03
- #define **ADC_CH_ADC4** 0x04
- #define **ADC_CH_ADC5** 0x05
- #define **ADC_CH_ADC6** 0x06
- #define **ADC_CH_ADC7** 0x07
- #define ADC_CH_122V 0x1E

  1.22V voltage reference

- #define ADC_CH_AGND 0x1F

  AGND.

- #define **ADC_CH_0_0_DIFF10X** 0x08
- #define **ADC_CH_1_0_DIFF10X** 0x09
- #define **ADC_CH_0_0_DIFF200X** 0x0A
- #define **ADC_CH_1_0_DIFF200X** 0x0B
- #define **ADC_CH_2_2_DIFF10X** 0x0C
- #define **ADC_CH_3_2_DIFF10X** 0x0D
- #define **ADC_CH_2_2_DIFF200X** 0x0E
- #define **ADC_CH_3_2_DIFF200X** 0x0F
- #define **ADC_CH_0_1_DIFF1X** 0x10
- #define **ADC_CH_1_1_DIFF1X** 0x11
- #define **ADC_CH_2_1_DIFF1X** 0x12
- #define **ADC_CH_3_1_DIFF1X** 0x13
- #define **ADC_CH_4_1_DIFF1X** 0x14
- #define **ADC_CH_5_1_DIFF1X** 0x15
- #define **ADC_CH_6_1_DIFF1X** 0x16
- #define **ADC_CH_7_1_DIFF1X** 0x17
- #define **ADC_CH_0_2_DIFF1X** 0x18
- #define **ADC_CH_1_2_DIFF1X** 0x19
- #define **ADC_CH_2_2_DIFF1X** 0x1A
- #define **ADC_CH_3_2_DIFF1X** 0x1B
- #define **ADC_CH_4_2_DIFF1X** 0x1C
- #define **ADC_CH_5_2_DIFF1X** 0x1D

Functions

- void a2dInit (void)
- void a2dOff (void)

  Turn off A/D converter.

- void a2dSetPrescaler (unsigned char prescale)
- void a2dSetReference (unsigned char ref)
- void a2dSetChannel (unsigned char ch)

  sets the a2d input channel

- void a2dStartConvert (void)

  start a conversion on the current a2d input channel

- u08 a2dIsComplete (void)

  return TRUE if conversion is complete

- unsigned short a2dConvert10bit (unsigned char ch)
- unsigned char a2dConvert8bit (unsigned char ch)

4.9.1 Detailed Description

```
#include "a2d.h"
```

Overview

This library provides an easy interface to the analog-to-digital converter available on many AVR processors. Updated to support the ATmega128.

4.9.2 Function Documentation

4.9.2.1 unsigned short a2dConvert10bit (unsigned char *ch*)

Starts a conversion on A/D channel# ch, returns the 10-bit value of the conversion when it is finished.

Definition at line 88 of file a2d.c.

Referenced by a2dConvert8bit().

4.9.2.2 unsigned char a2dConvert8bit (unsigned char *ch*)

Starts a conversion on A/D channel# ch, returns the 8-bit value of the conversion when it is finished.

Definition at line 104 of file a2d.c.

References a2dConvert10bit().

4.9.2.3 void a2dInit (void)

Initializes the A/D converter. Turns ADC on and prepares it for use.

Definition at line 34 of file a2d.c.

References a2dSetPrescaler(), and a2dSetReference().

4.9.2.4 void a2dSetPrescaler (unsigned char *prescale*)

Sets the division ratio of the A/D converter clock. This function is automatically called from a2dInit() with a default value.

Definition at line 56 of file a2d.c.

Referenced by a2dInit().

4.9.2.5 void a2dSetReference (unsigned char *ref*)

Configures which voltage reference the A/D converter uses. This function is automatically called from a2dInit() with a default value.

Definition at line 62 of file a2d.c.

Referenced by a2dInit().

4.10 Character LCD Driver for HD44780/SED1278-based displays (lcd.c)

```
#include "lcd.h"
```

Overview

This display driver provides an interface to the most common type of character LCD, those based on the HD44780 or SED1278 controller chip (about 90% of character LCDs use one of these chips). The display driver can interface to the display through the CPU memory bus, or directly via I/O port pins. When using the direct I/O port mode, no additional interface hardware is needed except for a contrast potentiometer. Supported functions include initialization, clearing, scrolling, cursor positioning, text writing, and loading of custom characters or icons (up to 8). Although these displays are simple, clever use of the custom characters can allow you to create animations or simple graphics. The "progress bar" function that is included in this driver is an example of graphics using limited custom-chars.

The driver now supports both 8-bit and 4-bit interface modes.

For full text output functionality, you may wish to use the rprintf functions along with this driver

4.11 BUS communication

Files

- file bus.c
  The communication bus protocol used in the openASC project.
- file bus_commands.h
  Global commands for the WMV communication bus.
- file bus_ping.c
  The communication bus ping control.
- file bus_ping.h
  The communication bus ping control.
- file bus_rx_queue.c
  FIFO queue for the RXed messages.
- file bus_rx_queue.h
  FIFO queue for the RXed messages.
- file bus_tx_queue.c
  FIFO queue for the TXed messages.
- file bus_tx_queue.h
  FIFO queue for the TXed messages.
- file bus_usart.c
  Driver unit USART routines.
- file bus_usart.h
  BUS usart routines.

4.11.1 Detailed Description

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "wmv_bus/bus.h"
```

Chapter 5

Class Documentation

5.1 BUS_MESSAGE Struct Reference

`#include <bus.h>`

Public Attributes

- unsigned char from_addr

  From which address the bus message was sent.

- unsigned char to_addr

  To which address the bus message should be sent.

- unsigned char checksum

  The checksum of the message.

- unsigned char flags

  Different flags.

- unsigned char cmd

  The command of the message.

- unsigned char length

  The length of the data sent in the message.

- unsigned char data [BUS_MESSAGE_DATA_SIZE]

  The variables wanted to be sent.

5.1.1 Detailed Description

BUS message structure

Definition at line 224 of file bus.h.

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

- wmv_bus/bus.h

5.2 bus_status_struct Struct Reference

`#include <bus.h>`

Public Attributes

- unsigned char frame_counter

  Counter which keeps track of which frame (slot) we are in.

- unsigned char ext_addr

  The address of this device.

- unsigned int lower_limit

  The lower limit of the time slot.

- unsigned int upper_limit

  The upper limit of the time slot.

- unsigned char device_count

  Nr of devices on the bus.

- unsigned int device_count_mult

  Nr of devices are x 4 so we don't need to do the division.

- unsigned int flags
- unsigned char char_count

  The char count of the received message.

- unsigned char send_count

  The number of times the last message was sent.

- unsigned char prev_char

  The previous character received.

- unsigned char wraparounds

  The number of wrap arounds.

5.2.1 Detailed Description

The bus status structure

Definition at line 262 of file bus.h.

5.2.2 Member Data Documentation

5.2.2.1 unsigned int bus_status_struct::flags

See details in defines

Definition at line 276 of file bus.h.

Referenced by bus_allowed_to_send(), bus_check_tx_status(), bus_init(), bus_is_master(), bus_message_acked(), bus_message_nacked(), bus_resend_message(), bus_reset_rx_status(), bus_reset_tx_status(), bus_send_message(), bus_set_is_master(), and ISR().

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

- wmv_bus/bus.h

5.3 bus_struct_ping_status Struct Reference

Struct which contains information of the bus ping information.

`#include <bus_ping.h>`

Public Attributes

- unsigned char addr

  The address of the device.
- unsigned char device_type

  The type of device it is, see bus.h for details.
- unsigned int time_last_ping

  The time since the last ping did occur in ms.
- unsigned char flags

  Flags, see defines above.
- unsigned char data [2]

  Data from the device, content varies depending on device_id.

5.3.1 Detailed Description

Struct which contains information of the bus ping information.

Definition at line 20 of file bus_ping.h.

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

- wmv_bus/bus_ping.h

5.4 computer_comm_struct Struct Reference

Computer interface communication struct.

Public Attributes

- char * tx_buffer

  The serial tx buffer.

- char * tx_buffer_start

  The start of the serial tx buffer.

- unsigned int tx_buffer_length

  The length of the data in the buffer.

- char * rx_buffer

  The serial rx buffer.

- char * rx_buffer_start

  The start of the serial rx buffer.

- unsigned char data_in_tx_buffer

  Different flags.

- unsigned char flags

  Flags for computer comm, defined in this file.

- unsigned char command

  The current command.

- unsigned char length

  Length of the data field.

- unsigned int count

  Current byte count.

5.4.1 Detailed Description

Computer interface communication struct.

Definition at line 184 of file computer_interface.c.

5.4.2 Member Data Documentation

5.4.2.1 unsigned char computer_comm_struct::command

The current command.

The command.

Definition at line 200 of file computer_interface.c.

Referenced by computer_interface_parse_data(), and ISR().

5.4.2.2 unsigned int computer_comm_struct::count

Current byte count.

Number of bytes received.

Definition at line 204 of file computer_interface.c.

Referenced by ISR().

5.4.2.3 unsigned char computer_comm_struct::flags

Flags for computer comm, defined in this file.

Various flags.

Definition at line 198 of file computer_interface.c.

Referenced by computer_interface_activate_setup(), computer_interface_deactivate_setup(), computer_interface_is_active(), computer_interface_parse_data(), and ISR().

5.4.2.4 unsigned char computer_comm_struct::length

Length of the data field.

Data length.

Definition at line 202 of file computer_interface.c.

Referenced by computer_interface_parse_data(), and ISR().

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

- front_panel/computer_interface.c
- motherboard/computer_interface.c

5.5 driver_status_struct Struct Reference

`#include <board.h>`

Public Attributes

- unsigned char driver_output_owner [20]

  *The address of the device that last changed the status of the output */.*
- unsigned char driver_output_type [20]

  The type of message that activated the output.
- unsigned long driver_output_state

  The state of the driver output if it's high or low.
- unsigned char flags

  Flags.
- unsigned char driver_output_new_owner [20]

  *The address of the device that last changed the status of the output */.*
- unsigned char driver_output_new_type [20]

  The type of message that activated the output.
- unsigned char ptt_interlock_input [7]

5.5.1 Detailed Description

Structure of the driver output status. It contains information of which address an output was activated/deactivated from last and also it's status

Definition at line 116 of file board.h.

5.5.2 Member Data Documentation

5.5.2.1 unsigned char driver_status_struct::ptt_interlock_input[7]

PTT interlock input, byte1 = input 1, byte2 = input 2...byte7 = input 7 0 = not active, If not 0 then the value does correspond to the address of the openASC box, so 5 for example means address 5 corresponds to that ptt interlock input

Definition at line 50 of file main.h.

Referenced by bus_parse_message().

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

- driver_unit/board.h
- driver_unit_v2/main.h

5.6 EVENT_MESSAGE Struct Reference

Event message used for timing of events.

`#include <event_queue.h>`

Public Attributes

- void(∗ func)(void)

  The function we wish to run at the specified time.

- unsigned int time_target

  The target time where we wish to event to occur.

- unsigned char id

  The event id, can be used to drop a certain type of messages.

5.6.1 Detailed Description

Event message used for timing of events.

Definition at line 30 of file event_queue.h.

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

- event_queue.h

5.7 powermeter_struct Struct Reference

Struct which contains information of the power meter status.

`#include <powermeter.h>`

Public Attributes

- unsigned int curr_fwd_pwr_value
  Current forward power in watts.
- unsigned int curr_ref_pwr_value
  Current reflected power in watts.
- unsigned int curr_vswr_value
  VSWR represented as an integer, 152 means 1.52:1 in SWR.
- unsigned char pickup_addr
  Address of the power meter pickup.
- unsigned int text_update_rate
  The update in ms of the text on the display.
- unsigned int bargraph_update_rate
  The update rate in ms of the bargraph.
- unsigned int vswr_limit
  The VSWR limit of when the radios PTT should be deactivated and the device set into ERROR mode.

5.7.1 Detailed Description

Struct which contains information of the power meter status.

Definition at line 27 of file powermeter.h.

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

- front_panel/powermeter.h

5.8 PS2_STRUCT Struct Reference

Struct of the PS/2 interface status.

```
#include <main.h>
```

Public Attributes

- unsigned char started

  To see if we have started to read a keyboard command.
- unsigned char bit_count

  Number of bytes we have receieved.
- unsigned char data

  The actual data received.
- unsigned char data_ready

  Flag to indicate that data is ready.
- unsigned char transmit

  Flag to indicate that we are transmitting.
- unsigned char parity

  The parity byte.
- unsigned char tx_data

  Transmit data.
- unsigned char prev_cmd

  Previous command.

5.8.1 Detailed Description

Struct of the PS/2 interface status.

Definition at line 37 of file main.h.

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

- motherboard/main.h

5.9 rx_linked_list Struct Reference

The structure of the RX circular buffer.

`#include <internal_comm_rx_queue.h>`

Public Attributes

- UC_MESSAGE message [INTERNAL_COMM_RX_QUEUE_SIZE]

  The list of messages.
- unsigned char first

  The index of the first message in the list.
- unsigned char last

  The index of the last message in the list.
- BUS_MESSAGE message [BUS_RX_QUEUE_SIZE]

  List of bus messages.

5.9.1 Detailed Description

The structure of the RX circular buffer.

Definition at line 32 of file internal_comm_rx_queue.h.

5.9.2 Member Data Documentation

5.9.2.1 unsigned char rx_linked_list::first

The index of the first message in the list.

The first item in the list.

Definition at line 36 of file internal_comm_rx_queue.h.

Referenced by int_comm_rx_queue_add(), int_comm_rx_queue_drop(), int_comm_rx_-queue_dropall(), int_comm_rx_queue_get(), int_comm_rx_queue_init(), int_comm_rx_-queue_is_empty(), rx_queue_add(), rx_queue_drop(), rx_queue_dropall(), rx_queue_get(), rx_queue_init(), and rx_queue_is_empty().

5.9.2.2 unsigned char rx_linked_list::last

The index of the last message in the list.

The last item in the list.

Definition at line 38 of file internal_comm_rx_queue.h.

Referenced by int_comm_rx_queue_add(), int_comm_rx_queue_dropall(), int_comm_rx_-queue_init(), int_comm_rx_queue_is_empty(), rx_queue_add(), rx_queue_dropall(), rx_-queue_init(), and rx_queue_is_empty().

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

- internal_comm_rx_queue.h
- wmv_bus/bus.h

5.10 struct_antenna Struct Reference

Structure of an antenna.

`#include <antenna_ctrl.h>`

Public Attributes

- unsigned int struct_size

  The size of this structure.

- unsigned char sub_menu_type [4]

  The type of sub menu it is.

- unsigned char antenna_text_length [4]

  The length of the text for the antennas.

- char antenna_text [4][ANTENNA_TEXT_SIZE]

  The text for the antennas.

- unsigned char antenna_flag [4]

  Antenna flags.

- int antenna_direction [4]

  The direction of the antennas.

- unsigned int antenna_comb_allowed
- unsigned char antenna_output_length [15]

  The length of the antenna output strings.

- unsigned char antenna_comb_output_str [15][ANTENNA_OUTPUT_COMB_SIZE]
- unsigned char rotator_addr [4]

  The address to the rotator which controls the antenna.

- unsigned char rotator_sub_addr [4]

  The SUB address to the rotator which controls the antenna.

- unsigned int rotator_max_rotation [4]

  The number of degrees the rotator can turn, this might be for example 450 degrees for YAESU.

- unsigned int rotator_min_heading [4]

  The minimum heading of the rotator, this can also be negative numbers if starting point is not at 0 degrees.

- unsigned char rotator_delay [4]

  The delay from a rotation has occured to it can start to rotate again (in seconds).

- unsigned char rotator_flags [4]

  The rotator flags.

- unsigned char rotator_view_360_deg

  Does the rotator have 360 degree view? Should it show 0-360 degree or start_point + rotation, maybe 90 - 500 deg.

- unsigned char default_antenna

  The default antenna index (0-3).

5.10.1 Detailed Description

Structure of an antenna.

Definition at line 55 of file antenna_ctrl.h.

5.10.2 Member Data Documentation

5.10.2.1 unsigned int struct_antenna::antenna_comb_allowed

This is used to show what antenna combinations that are allowed Bit0 = ANT 1 Bit1 = ANT 2 Bit2 = ANT 3 Bit3 = ANT 4 Bit4 = ANT 1 + ANT 2 Bit5 = ANT 1 + ANT 3 Bit6 = ANT 1 + ANT 4 Bit7 = ANT 2 + ANT 3 Bit8 = ANT 2 + ANT 4 Bit9 = ANT 3 + ANT 4 Bit10 = ANT 1 + ANT 2 + ANT 3 Bit11 = ANT 1 + ANT 2 + ANT 4 Bit12 = ANT 1 + ANT 3 + ANT 4 Bit13 = ANT 2 + ANT 3 + ANT 4 Bit14 = ANT 1 + ANT 2 + ANT 3 + ANT 4

Definition at line 85 of file antenna_ctrl.h.

Referenced by antenna_ctrl_comb_allowed(), antenna_ctrl_get_comb_allowed(), antenna_-ctrl_set_comb_allowed(), and computer_interface_parse_data().

5.10.2.2 unsigned char struct_antenna::antenna_comb_output_-str[15][ANTENNA_OUTPUT_COMB_SIZE]

The antenna output strings which contains what outputs that should be activated when the antenna combination is chosen

Definition at line 90 of file antenna_ctrl.h.

Referenced by antenna_ctrl_get_output_comb(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_set_output_comb(), and computer_interface_parse_data().

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

- front_panel/antenna_ctrl.h

5.11 struct_band Struct Reference

Struct of band data.

```
#include <band_ctrl.h>
```

Public Attributes

- unsigned int struct_size

  The size of this structure.
- unsigned int low_portion_low_limit

  The low limit of the lower portion of the band.
- unsigned int low_portion_high_limit

  The high limit of the lower portion of the band.
- unsigned int high_portion_low_limit

  The low limit of the higher portion of the band.
- unsigned int high_portion_high_limit

  The high limit of the higher portion of the band.
- unsigned char band_high_output_str_length

  The length of the high output str.
- unsigned char band_low_output_str_length

  The length of the low output str.
- unsigned char band_high_output_str [BAND_OUTPUT_STR_SIZE]

  These outputs are activated when you enter the high area of a band.
- unsigned char band_low_output_str [BAND_OUTPUT_STR_SIZE]

  These outputs are activated when you enter the low area of a band.

5.11.1 Detailed Description

Struct of band data.

Definition at line 29 of file band_ctrl.h.

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

- front_panel/band_ctrl.h

5.12 struct_band_limits Struct Reference

Struct of the band limits.

```
#include <band_ctrl.h>
```

Public Attributes

- unsigned int low_portion_low_limit

  The low limit of the lower portion of the band.

- unsigned int low_portion_high_limit

  The high limit of the lower portion of the band.

- unsigned int high_portion_low_limit

  The low limit of the higher portion of the band.

- unsigned int high_portion_high_limit

  The high limit of the higher portion of the band.

5.12.1 Detailed Description

Struct of the band limits.

Definition at line 51 of file band_ctrl.h.

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

- front_panel/band_ctrl.h

5.13 struct_coupler_settings Struct Reference

Struct which contains information of the coupler.

`#include <input.h>`

Public Attributes

- unsigned char coupler_name [COUPLER_NAME_LENGTH]

    The name of the coupler.

- unsigned int fwd_scale_value [10]

    The value which the read RMS voltage should be multiplied with.

- unsigned int ref_scale_value [10]

    The value which the read RMS voltage should be multiplied with.

- unsigned int power_limit

    The power limit of the coupler, high (in watts).

- unsigned char **pickup_type**

5.13.1 Detailed Description

Struct which contains information of the coupler.

Struct which contains information of the pickup type.

Definition at line 28 of file input.h.

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

- powermeter/display_unit/input.h
- powermeter/sensor_unit/input.h

5.14 struct_eeprom_table Struct Reference

The EEPROM table.

`#include <eeprom.h>`

Public Attributes

- unsigned int struct_size

  The size of this structure.

- unsigned int antenna [9]

  The start address of the antenna structure in the EEPROM memory.

- unsigned int band [9]

  The start address of the band structure in the EEPROM memory.

- unsigned int rx_antennas

  The start address of the RX antenna structure in the EEPROM memory.

- unsigned int settings

  The start address of the setting structure in the EEPROM memory.

- unsigned int radio_settings

  The start address of the radio settings structure in the EEPROM memory.

- unsigned int struct_ptt

  The start address of the sequencer.

- unsigned int antenna1_sub_menu [9]

  The sub menus of antenna 1.

- unsigned int antenna2_sub_menu [9]

  The sub menus of antenna 2.

- unsigned int antenna3_sub_menu [9]

  The sub menus of antenna 3.

- unsigned int antenna4_sub_menu [9]

  The sub menus of antenna 4.

- unsigned int rx_antenna_sub_menu [10]

  The sub menus of the rx antennas.

- unsigned int runtime_settings

  Runtime settings, such as backlight level etc.

5.14.1 Detailed Description

The EEPROM table.

Definition at line 38 of file eeprom.h.

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

- front_panel/eeprom.h

5.15 struct_menu_option Struct Reference

Struct of a menu option.

`#include <menu.h>`

Public Attributes

- char ∗ text

  Menu option text.

5.15.1 Detailed Description

Struct of a menu option.

Definition at line 29 of file menu.h.

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

- front_panel/menu.h

5.16 struct_menu_text Struct Reference

Menu text structs.

`#include <menu.h>`

Public Attributes

- unsigned char pos

  Position nr in the menu system.

- char ∗ header

  Header text.

- struct_menu_option ∗ options

  Pointer to the options.

- unsigned char option_count

  Number of options.

- unsigned char option_type

5.16.1 Detailed Description

Menu text structs.

Definition at line 35 of file menu.h.

5.16.2 Member Data Documentation

5.16.2.1 unsigned char struct_menu_text::option_type

Which kind of option 0 = regular option 1 = numbers 2 = nothing

Definition at line 49 of file menu.h.

Referenced by menu_show_text().

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

- front_panel/menu.h

5.17 struct_ptt Struct Reference

PTT Sequencer struct.

`#include <sequencer.h>`

Public Attributes

- unsigned int struct_size

  The size of this structure in bytes.
- struct_ptt_sequencer computer

  The PTT SEQUENCER for the computer input.
- struct_ptt_sequencer footswitch

  The PTT SEQUENCER for the footswitch input.
- struct_ptt_sequencer radio_sense

  The PTT SEQUENCER for the radio sense input.
- unsigned char ptt_input

5.17.1 Detailed Description

PTT Sequencer struct.

Definition at line 105 of file sequencer.h.

5.17.2 Member Data Documentation

5.17.2.1 unsigned char struct_ptt::ptt_input

Bit 0 = Footswitch

Bit 1 = Radio sense lower floor

Bit 2 = Radio sense upper floor

Bit 3 = Computer RTS

Bit 4 = Inverted radio sense

Bit 5 = Inverted Computer RTS

Bit 6 = Inhibit polarity (0=active low, 1=active high)

Definition at line 122 of file sequencer.h.

Referenced by computer_interface_activate_setup(), computer_interface_parse_data(), sequencer_computer_rts_activated(), sequencer_computer_rts_deactivated(), sequencer_footsw_pressed(), sequencer_footsw_released(), sequencer_get_radio_sense(), sequencer_get_rts_polarity(), and sequencer_get_sense_polarity().

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

- front_panel/sequencer.h

5.18 struct_ptt_sequencer Struct Reference

All the delays are divided with 10 so 100 is really 1000 ms which makes the maximium delay 2550 ms.

`#include <sequencer.h>`

Public Attributes

- unsigned char radio_pre_delay
  The delay before the radio is PTTed after the input PTT has been activated.
- unsigned char radio_post_delay
  The delay after the input PTT has been released and the radio PTT is released.
- unsigned char amp_pre_delay
  The delay before the amp is PTTed after the input PTT has been activated.
- unsigned char amp_post_delay
  The delay after the input PTT has been released and the amp PTT is released.
- unsigned char inhibit_pre_delay
  The delay before the inhibit is activated after the input PTT has been activated.
- unsigned char inhibit_post_delay
  The delay after the input PTT has been released and the inhibit pin is released.
- unsigned char antennas_post_delay
  The delay after theinput PTT has been released and the antennas are switched.
- unsigned char active
  Flags on which sequencer variables that are enabled.

5.18.1 Detailed Description

All the delays are divided with 10 so 100 is really 1000 ms which makes the maximium delay 2550 ms.

Definition at line 83 of file sequencer.h.

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

- front_panel/sequencer.h

5.19 struct_radio_settings Struct Reference

Radio settings struct.

`#include <radio_interface.h>`

Public Attributes

- unsigned int struct_size
  The size of this structure.
- unsigned char radio_model
  Which model of the radio.
- unsigned char interface_type
  Which kind of interface to detect band.
- unsigned char baudrate
  Baudrate of the radio, used in serial mode.
- unsigned char stopbits
  Number of stop bits, used in serial mode.
- unsigned char civ_addr
  If it's an ICOM, what is the CI-V address to the radio.
- unsigned char ptt_mode
  What kind of PTT mode, inhibit, static etc.
- unsigned char ptt_input
- unsigned char poll_interval
- unsigned char cat_enabled
  The CAT is enabled.

5.19.1 Detailed Description

Radio settings struct.

Definition at line 93 of file radio_interface.h.

5.19.2 Member Data Documentation

5.19.2.1 unsigned char struct_radio_settings::poll_interval

The interval to poll the band information from the radio, this should be set in 10th ms, so for example 100 means 1000ms.

Definition at line 116 of file radio_interface.h.

Referenced by computer_interface_parse_data(), radio_interface_get_poll_interval(), and radio_interface_set_poll_interval().

5.19.2.2 unsigned char struct_radio_settings::ptt_input

From which input should we monitor the radio PTT? Bit 0 = Radio sense lower floor Bit 1 = Radio sense upper floor Bit 2 = Inverted sense pin (if this is set the box will sense PTT as LOW)

Definition at line 113 of file radio_interface.h.

Referenced by computer_interface_parse_data(), radio_interface_get_ptt_input(), and radio_interface_set_ptt_input().

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

- front_panel/radio_interface.h

5.20 struct_radio_status Struct Reference

The radio status struct.

`#include <radio_interface.h>`

Public Attributes

- unsigned int current_freq

  The radios current frequency.

- unsigned int new_freq

  The radios new frequency.

- unsigned char current_band

  The radios current band.

- unsigned char box_sent_request

5.20.1 Detailed Description

The radio status struct.

Definition at line 122 of file radio_interface.h.

5.20.2 Member Data Documentation

5.20.2.1 unsigned char struct_radio_status::box_sent_request

Variable which is set if the openASC box has sent a request to the radio, used to know if we can just redirect the data from the radio to the computer or if it should be thrown away

Definition at line 131 of file radio_interface.h.

Referenced by radio_communicaton_timeout(), radio_get_cat_status(), and radio_interface_-init().

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

- front_panel/radio_interface.h

5.21 struct_runtime_settings Struct Reference

Settings like status but which should be saved into the EEPROM.

`#include <main.h>`

Public Attributes

- unsigned char lcd_backlight_value

  The value of the LCD backlight, 0-100%.
- unsigned char amplifier_ptt_output

  Amp PTT output status, 1 = ON, 0 = OFF.
- unsigned char radio_ptt_output

  Radio PTT output status, 1 = ON, 0 = OFF.
- unsigned char inhibit_state

  Show if the device is inhibited or not.
- unsigned char band_change_mode

  Band change mode.

5.21.1 Detailed Description

Settings like status but which should be saved into the EEPROM.

Definition at line 272 of file main.h.

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

- front_panel/main.h

5.22 struct_rx_antennas Struct Reference

Struct which contains information of the rx antennas.

`#include <antenna_ctrl.h>`

Public Attributes

- unsigned int struct_size

  The size of this structure.
- unsigned char name_length [10]

  The length of the antenna names.
- char name [10][RX_ANTENNA_NAME_LENGTH]

  RX antenna name.
- unsigned char output_length [10]

  RX antenna output str length.
- char output_str [10][RX_ANTENNA_OUTPUT_STR_LENGTH]

  RX antenna output str.
- unsigned char band_output_length [4]

  The length of the band output data.
- char band_output_str [4][RX_ANTENNA_BAND_OUTPUT_STR_LENGTH]

  Band output str.

5.22.1 Detailed Description

Struct which contains information of the rx antennas.

Definition at line 37 of file antenna_ctrl.h.

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

- front_panel/antenna_ctrl.h

5.23 struct_setting Struct Reference

Settings struct.

`#include <main.h>`

Public Attributes

- unsigned int struct_size

  The size of this structure.

- unsigned char network_address

  This device address on the communication bus.

- unsigned char network_device_count

  The number of devices on the bus.

- unsigned char network_device_is_master

  Device is the master unit.

- unsigned char ext_key_assignments [17]

  The external keypad assignments.

- unsigned char powermeter_address

  The powermeter address.

- unsigned int powermeter_vswr_limit

  The powermeter VSWR alarm limit (250 means 2.5:1).

- unsigned int powermeter_update_rate_text

  The powermeter update rate on the display text (0 means it's disabled, both text and bargraph).

- unsigned int powermeter_update_rate_bargraph

  The powermeter update rate on the display bargraph.

- unsigned char ptt_interlock_input

5.23.1 Detailed Description

Settings struct.

Definition at line 197 of file main.h.

5.23.2 Member Data Documentation

5.23.2.1 unsigned char struct_setting::ptt_interlock_input

Which PTT input of various boards this openASC box is configured to use, this is sent out in PING messages and is saved in the various boxes so that they are aware of which TX ACTIVE input they should listen to 0 = None, 1-7 inputs

Definition at line 219 of file main.h.

Referenced by computer_interface_parse_data().

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

- front_panel/main.h

5.24 struct_status Struct Reference

This struct only contains information that is temporary.

`#include <main.h>`

Public Attributes

- unsigned int buttons_current_state

  The current state of the buttons.

- unsigned int buttons_last_state

  The last state of the buttons.

- unsigned char ext_devices_current_state

  The current state of the ext devices.

- unsigned char ext_devices_last_state

  The last state of the ext devices.

- unsigned char selected_ant

  Bit 0-3 = TX, Bit 4-7 = RX.

- unsigned char selected_band

  The currently selected band.

- unsigned char new_band

  The variable for changing to a new band.

- unsigned char current_band_portion

  CURRENT Band portion selected.

- unsigned char new_band_portion

  NEW Band portion selected.

- unsigned int new_beamheading

  The variable for the new beamheading.

- unsigned char function_status

  The status of different functions, like rx ant etc, see defines above.

- unsigned char current_display_level
- unsigned char current_display
- unsigned char selected_rx_antenna
- unsigned char knob_function
- unsigned char antenna_to_rotate
- unsigned char rotator_step_resolution
- unsigned char last_rx_antenna
- unsigned char sub_menu_antenna_index
- unsigned int curr_fwd_ad_value

Current A/D value for the forward power.

- unsigned int curr_ref_ad_value

  Current A/D value for the ref power.

- unsigned int curr_fwd_power

  Current forward power (in Watts).

- unsigned int curr_ref_power

  Current reflected power (in Watts).

- double curr_vswr

  Current VSWR.

5.24.1 Detailed Description

This struct only contains information that is temporary.

Definition at line 223 of file main.h.

5.24.2 Member Data Documentation

5.24.2.1 unsigned char struct_status::antenna_to_rotate

Which antenna to rotate

Definition at line 260 of file main.h.

Referenced by event_poll_buttons(), event_pulse_sensor_down(), event_pulse_sensor_up(), event_rotate_button_pressed(), event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_button3_pressed(), and event_tx_button4_pressed().

5.24.2.2 unsigned char struct_status::current_display

0 = openASC logo, 1 = antenna info, 2 = menu system, 3 = shutdown view

Definition at line 253 of file main.h.

Referenced by band_ctrl_change_band(), display_update_radio_freq(), event_internal_-comm_parse_message(), event_poll_buttons(), event_pulse_sensor_down(), event_pulse_-sensor_up(), and event_update_display().

5.24.2.3 unsigned char struct_status::current_display_level

0 = openASC logo, 1 = curr band level, 2 = sub menu

Definition at line 251 of file main.h.

Referenced by band_ctrl_change_band(), display_show_rx_ant(), display_show_set_-heading(), display_show_sub_menu(), display_update(), event_poll_buttons(), and event_-sub_button_pressed().

5.24.2.4 unsigned char struct_status::knob_function

Knob function

Definition at line 258 of file main.h.

Referenced by event_poll_buttons(), event_pulse_sensor_down(), event_pulse_sensor_up(), event_rotate_button_pressed(), event_rxant_button_pressed(), and set_knob_function().

5.24.2.5 unsigned char struct_status::last_rx_antenna

The last RX antenna used

Definition at line 265 of file main.h.

Referenced by event_rxant_button_pressed().

5.24.2.6 unsigned char struct_status::rotator_step_resolution

Rotator resolution chosen

Definition at line 262 of file main.h.

Referenced by event_pulse_sensor_down(), event_pulse_sensor_up(), and main().

5.24.2.7 unsigned char struct_status::selected_rx_antenna

The currently selected RX antenna, -1 if none selected

Definition at line 256 of file main.h.

Referenced by band_ctrl_change_band(), event_pulse_sensor_down(), event_pulse_sensor_-up(), event_rxant_button_pressed(), event_set_rx_antenna(), event_update_display(), and main().

5.24.2.8 unsigned char struct_status::sub_menu_antenna_index

The sub menu antenna index we are changing

Definition at line 268 of file main.h.

Referenced by event_pulse_sensor_down(), event_pulse_sensor_up(), event_sub_button_-pressed(), and main().

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

- front_panel/main.h
- powermeter/display_unit/input.h
- powermeter/sensor_unit/input.h

5.25 struct_sub_menu_array Struct Reference

Struct of a sub menu with the type array.

`#include <sub_menu.h>`

Public Attributes

- unsigned int struct_size

  The size of this structure.

- unsigned char direction_count

  Number of directions.

- unsigned char direction_name [8][SUB_MENU_ARRAY_NAME_SIZE]

  The directions.

- unsigned char output_str_dir_length [8]

  The length of the output str.

- unsigned char output_str_dir [8][SUB_MENU_ARRAY_STR_SIZE]

  The output strings of the different directions.

5.25.1 Detailed Description

Struct of a sub menu with the type array.

Definition at line 29 of file sub_menu.h.

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

- front_panel/sub_menu.h

5.26 struct_uc_com Struct Reference

`#include <internal_comm.h>`

Public Attributes

- unsigned char flags

    Various flags, defined in this file.

- unsigned char checksum

    The checksum.

- unsigned char char_count

    Number of characters received.

5.26.1 Detailed Description

Variables used for the communication between the two uCs

Definition at line 106 of file internal_comm.h.

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

- internal_comm.h

5.27 tx_linked_list Struct Reference

The structure of the TX circular buffer.

`#include <internal_comm_tx_queue.h>`

Public Attributes

- UC_MESSAGE message [INTERNAL_COMM_TX_QUEUE_SIZE]

  A UC_MESSAGE.

- unsigned char first

  first position in the list

- unsigned char last

  last position in the list

- BUS_MESSAGE message [BUS_TX_QUEUE_SIZE]

  The bus messages.

5.27.1 Detailed Description

The structure of the TX circular buffer.

Definition at line 26 of file internal_comm_tx_queue.h.

5.27.2 Member Data Documentation

5.27.2.1 unsigned char tx_linked_list::first

first position in the list

The first position in the list.

Definition at line 30 of file internal_comm_tx_queue.h.

Referenced by int_comm_int_comm_tx_queue_init(), int_comm_tx_queue_add(), int_-comm_tx_queue_drop(), int_comm_tx_queue_dropall(), int_comm_tx_queue_get(), int_-comm_tx_queue_is_empty(), tx_queue_add(), tx_queue_drop(), tx_queue_dropall(), tx_-queue_get(), tx_queue_init(), and tx_queue_is_empty().

5.27.2.2 unsigned char tx_linked_list::last

last position in the list

The last position in the list.

Definition at line 32 of file internal_comm_tx_queue.h.

Referenced by int_comm_int_comm_tx_queue_init(), int_comm_tx_queue_add(), int_-comm_tx_queue_dropall(), int_comm_tx_queue_is_empty(), tx_queue_add(), tx_queue_-dropall(), tx_queue_init(), and tx_queue_is_empty().

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

- internal_comm_tx_queue.h
- wmv_bus/bus.h

5.28 UC_MESSAGE Struct Reference

`#include <internal_comm.h>`

Public Attributes

- unsigned char checksum

  The checksum of the message.

- unsigned char cmd

  The command of the message.

- unsigned char length

  The length of the data sent in the message.

- unsigned char data [UC_MESSAGE_DATA_SIZE]

  The variables wanted to be sent.

5.28.1 Detailed Description

uC message structure, used for communication between the uCs

Definition at line 94 of file internal_comm.h.

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

- internal_comm.h

Chapter 6

File Documentation

6.1 driver_unit/board.h File Reference

Board specific defines.

Classes

- struct driver_status_struct

Defines

- #define FLAG_TXRX_MODE_ENABLED 0
- #define ADDRESS_PORT PORTD

  Address input port ∗/.

- #define ADDRESS_BIT0 4

  Address input BIT 0 port offset ∗/.

- #define ADDRESS_BIT1 5

  Address input BIT 1 port offset ∗/.

- #define ADDRESS_BIT2 6

  Address input BIT 2 port offset ∗/.

- #define ADDRESS_BIT3 7

  Address input BIT 3 port offset ∗/.

- #define DRIVER_OUTPUT_1 2

  Driver output 1 port offset.

- #define DRIVER_OUTPUT_2 3

  Driver output 2 port offset.

- #define DRIVER_OUTPUT_3 2

Driver output 3 port offset.

- #define DRIVER_OUTPUT_4 3

    Driver output 4 port offset.

- #define DRIVER_OUTPUT_5 4

    Driver output 5 port offset.

- #define DRIVER_OUTPUT_6 5

    Driver output 6 port offset.

- #define DRIVER_OUTPUT_7 6

    Driver output 7 port offset.

- #define DRIVER_OUTPUT_8 7

    Driver output 8 port offset.

- #define DRIVER_OUTPUT_9 7

    Driver output 9 port offset.

- #define DRIVER_OUTPUT_10 6

    Driver output 10 port offset.

- #define DRIVER_OUTPUT_11 5

    Driver output 11 port offset.

- #define DRIVER_OUTPUT_12 4

    Driver output 12 port offset.

- #define DRIVER_OUTPUT_13 3

    Driver output 13 port offset.

- #define DRIVER_OUTPUT_14 2

    Driver output 14 port offset.

- #define DRIVER_OUTPUT_15 1

    Driver output 15 port offset.

- #define DRIVER_OUTPUT_16 0

    Driver output 16 port offset.

- #define DRIVER_OUTPUT_17 0

    Driver output 17 port offset.

- #define DRIVER_OUTPUT_18 1

    Driver output 18 port offset.

- #define DRIVER_OUTPUT_19 2

    Driver output 19 port offset.

- #define DRIVER_OUTPUT_20 3
  Driver output 20 port offset.
- #define DRIVER_STATUS_OFF 0
  Driver status for output OFF.
- #define DRIVER_STATUS_ON 1
  Driver status for output ON.

6.1.1 Detailed Description

Board specific defines.

Author:

Mikael Larsmark, SM2WMV

Date:

2008-04-06

```
#include "driver_unit/board.h"
```

Definition in file board.h.

6.1.2 Define Documentation

6.1.2.1 #define FLAG_TXRX_MODE_ENABLED 0

Flag to indicate if the TX/RX mode is enabled

Definition at line 112 of file board.h.

6.2 driver_unit_v2/board.h File Reference

Driver unit board defines.

6.2.1 Detailed Description

Driver unit board defines.

Author:

Mikael Larsmark, SM2WMV

Date:

2009-03-16

```
#include "driver_unit_v2/board.h"
```

Definition in file board.h.

6.3 front_panel/board.h File Reference

Front panel board defines.

Defines

- #define LED_TX_BUTTON1_BIT 7

  Bit offset of TX button 1 LED.
- #define LED_TX_BUTTON2_BIT 5

  Bit offset of TX button 2 LED.
- #define LED_TX_BUTTON3_BIT 3

  Bit offset of TX button 3 LED.
- #define LED_TX_BUTTON4_BIT 1

  Bit offset of TX button 4 LED.
- #define LED_ERROR_BIT 7

  Bit offset of Error LED.
- #define LED_PTT_GREEN_BIT 0

  Bit offset of PTT Green LED.
- #define LED_PTT_RED_BIT 1

  Bit offset of PTT Red LED.
- #define LED_ROTATION_ACTIVE_BIT 6

  Bit offset of rotation active LED.
- #define LED_RX_BUTTON1_BIT 3

  Bit offset of RX button 1 LED.
- #define LED_RX_BUTTON2_BIT 5

  Bit offset of RX button 2 LED.
- #define LED_RX_BUTTON3_BIT 4

  Bit offset of RX button 3 LED.
- #define LED_RX_BUTTON4_BIT 6

  Bit offset of RX button 4 LED.
- #define LED_ROTATE_BIT 4

  Bit offset of LED rotate.
- #define LED_TXRX_BIT 2

  Bit offset of LED TX/RX mode.
- #define LED_RXANT_BIT 6

Bit offset of LED RX ANTENNA.

- #define LED_SUBMENU_BIT 4
  Bit offset of LED SUB MENU.
- #define LED_MENU_BIT 7
  bit offset of LED MENU
- #define LED_AUX_BIT 7
  Bit offset of LED for MENU system.
- #define PULSE_SENSOR_BIT1 6
  Bit offset of the pulse sensor.
- #define PULSE_SENSOR_BIT2 7
  Bit offset of the pulse sensor.
- #define BUTTON1_TX_BIT 6
  Bit offset of TX button 1.
- #define BUTTON2_TX_BIT 4
  Bit offset of TX button 2.
- #define BUTTON3_TX_BIT 2
  Bit offset of TX button 3.
- #define BUTTON4_TX_BIT 0
  Bit offset of TX button 4.
- #define BUTTON1_RX_BIT 2
  Bit offset of RX button 1.
- #define BUTTON2_RX_BIT 4
  Bit offset of RX button 2.
- #define BUTTON3_RX_BIT 6
  Bit offset of RX button 3.
- #define BUTTON4_RX_BIT 5
  Bit offset of RX button 4.
- #define BUTTON_ROTATE_BIT 5
  Bit offset of rotate button.
- #define BUTTON_TXRX_BIT 3
  Bit offset of TX/RX mode button.
- #define BUTTON_RXANT_BIT 7
  Bit offset of RX Antenna button.

- #define BUTTON_SUBMENU_BIT 5

  Bit offset of SUB MENU button.
- #define BUTTON_MENU_BIT 6

  Bit offset of MENU button.
- #define BUTTON_PULSE_BIT 2

  Bit offset of PULSE SENSOR button.
- #define BUTTON_AUX1_BIT 0

  Bit offset of AUX 1 button.
- #define BUTTON_AUX2_BIT 1

  Bit offset of AUX 2 button.
- #define EXT_RADIO_SENSE1_BIT 4

  Bit offset of the EXT Radio sense 1.
- #define EXT_RADIO_SENSE2_BIT 2

  Bit offset of the EXT Radio sense 2.
- #define EXT_FOOTSWITCH_BIT 3

  Bit offset of the footswitch.
- #define EXT_USB1_DTR_BIT 4

  Bit offset of the USB 1 DTR.
- #define EXT_USB2_DTR_BIT 5

  Bit offset of the USB 2 DTR.
- #define EXT_USB2_RTS_BIT 6

  Bit offset of the USB 2 RTS.
- #define AMPLIFIER_OUTPUT_BIT 2

  Bit offset of the amplifier output.
- #define TX_ACTIVE_OUTPUT_BIT 5

  Bit offset of the tx active output.
- #define RADIO_INHIBIT_OUTPUT_BIT 5

  Bit offset of the inhibit output.
- #define FLAG_BUTTON1_TX_BIT 0

  Flag is set if the TX ANTENNA #1 button is pressed.
- #define FLAG_BUTTON2_TX_BIT 1

  Flag is set if the TX ANTENNA #2 button is pressed.
- #define FLAG_BUTTON3_TX_BIT 2

  Flag is set if the TX ANTENNA #3 button is pressed.

- #define FLAG_BUTTON4_TX_BIT 3

  Flag is set if the TX ANTENNA #4 button is pressed.
- #define FLAG_BUTTON1_RX_BIT 4

  Flag is set if the RX ANTENNA #1 button is pressed.
- #define FLAG_BUTTON2_RX_BIT 5

  Flag is set if the RX ANTENNA #2 button is pressed.
- #define FLAG_BUTTON3_RX_BIT 6

  Flag is set if the RX ANTENNA #3 button is pressed.
- #define FLAG_BUTTON4_RX_BIT 7

  Flag is set if the RX ANTENNA #4 button is pressed.
- #define FLAG_BUTTON_MENU_BIT 8

  Flag is set if the menu button is pressed.
- #define FLAG_BUTTON_ROTATE_BIT 9

  Flag is set if the Rotate button is pressed.
- #define FLAG_BUTTON_TXRX_BIT 10

  Flag is set if the TX/RX mode button is pressed.
- #define FLAG_BUTTON_RXANT_BIT 11

  Flag is set if the RX ANTENNA button is pressed.
- #define FLAG_BUTTON_SUBMENU_BIT 12

  Flag is set if the AUX button is pressed.
- #define FLAG_BUTTON_PULSE_BIT 13

  Flag is set if the pulse sensor button is pressed.
- #define FLAG_BUTTON_AUX1_BIT 14

  Flag is set if the AUX button 1 is pressed.
- #define FLAG_BUTTON_AUX2_BIT 15

  Flag is set if the AUX button 2 is pressed.
- #define STATUS_RADIO_SENSE1_BIT 0

  This bit shows the status of the radio sense input on floor 1.
- #define STATUS_FOOTSWITCH_BIT 1

  This bit shows the status of the footswitch input.
- #define STATUS_RADIO_SENSE2_BIT 2

  This bit shows the status of the radio sense input on floor 2.
- #define STATUS_USB1_DTR_BIT 3

This bit shows the status of the USB DTR on USB port 1.

- #define STATUS_USB2_DTR_BIT 4

  This bit shows the status of the USB DTR on USB port 2.

- #define STATUS_USB2_RTS_BIT 5

  This bit shows the status of the USB RTS on USB port 2.

6.3.1 Detailed Description

Front panel board defines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/board.h"
```

Definition in file board.h.

6.4 general_io/board.h File Reference

General I/O board defines.

6.4.1 Detailed Description

General I/O board defines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-05-18

```
#include "general_io/board.h"
```

Definition in file board.h.

6.5 motherboard/board.h File Reference

Motherboard defines.

Defines

- #define DRIVER_OUTPUT_1 0

  Driver output 1 port offset.

- #define DRIVER_OUTPUT_2 1

  Driver output 2 port offset.

- #define DRIVER_OUTPUT_3 2

  Driver output 3 port offset.

- #define DRIVER_OUTPUT_4 3

  Driver output 4 port offset.

- #define DRIVER_OUTPUT_5 4

  Driver output 5 port offset.

- #define DRIVER_OUTPUT_6 5

  Driver output 6 port offset.

- #define DRIVER_OUTPUT_7 6

  Driver output 7 port offset.

- #define DRIVER_OUTPUT_8 7

  Driver output 8 port offset.

- #define DRIVER_OUTPUT_9 7

  Driver output 9 port offset.

- #define DRIVER_OUTPUT_10 6

  Driver output 10 port offset.

- #define DRIVER_OUTPUT_11 5

  Driver output 11 port offset.

- #define DRIVER_OUTPUT_12 4

  Driver output 12 port offset.

- #define AUX_X11_PIN3 0

  AUX pin #3 on the X11 connector.

- #define AUX_X11_PIN8 1

  AUX pin #8 on the X11 connector.

- #define AUX_X11_PIN4 2

AUX pin #4 on the X11 connector.

- #define AUX_X11_PIN5 3

  AUX pin #5 on the X11 connector, relay output (draws to either +12V or GND).

- #define AUX_X11_PIN9 4

  AUX pin #9 on the X11 connector, relay output (draws to either +12V or GND).

6.5.1 Detailed Description

Motherboard defines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "board.h"
```

Definition in file board.h.

6.6 driver_unit/global.h File Reference

AVRlib project global include.

```
#include "../avrlibdefs.h"
#include "../avrlibtypes.h"
```

6.6.1 Detailed Description

AVRlib project global include.

Author:

Pascal Stang and Mikael Larsmark, SM2WMV

Date:

2008-04-13

Definition in file global.h.

6.7 driver_unit_v2/global.h File Reference

AVRlib project global include.

`#include "../avrlibdefs.h"`

`#include "../avrlibtypes.h"`

6.7.1 Detailed Description

AVRlib project global include.

Author:

Pascal Stang and Mikael Larsmark, SM2WMV

Date:

2009-03-16

Definition in file global.h.

6.8 general_io/global.h File Reference

AVRlib project global include.

```
#include "../avrlibdefs.h"
#include "../avrlibtypes.h"
```

6.8.1 Detailed Description

AVRlib project global include.

Author:

Pascal Stang and Mikael Larsmark, SM2WMV

Date:

2010-05-18

Definition in file global.h.

6.9 wmv_bus/global.h File Reference

AVRlib project global include.

Defines

- #define F_CPU 14745000

  The CPU speed.

- #define CYCLES_PER_US ((F_CPU+500000)/1000000)

  Cycles per us.

6.9.1 Detailed Description

AVRlib project global include.

Author:

Pascal Stang and Mikael Larsmark, SM2WMV

Date:

2008-04-13

Definition in file global.h.

6.10 driver_unit/init.c File Reference

Initialization routines for the driver unit.

`#include <stdio.h>`

`#include <avr/io.h>`

`#include <avr/interrupt.h>`

Defines

- #define OCR0_1MS 14

  Used for timer compare to match 1 ms.

Functions

- void init_timer_0 (void)
- void init_timer_2 (void)
- void init_ports (void)

6.10.1 Detailed Description

Initialization routines for the driver unit.

Author:

Mikael Larsmark, SM2WMV

Date:

2009-03-16

```
#include "driver_unit/init.c"
```

Definition in file init.c.

6.10.2 Function Documentation

6.10.2.1 void init_ports (void)

Set the direction of the ports

Definition at line 56 of file init.c.

Referenced by main().

6.10.2.2 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 35 of file init.c.

Referenced by main().

6.10.2.3 void init_timer_2 (void)

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Definition at line 45 of file init.c.

Referenced by main().

6.11 driver_unit_v2/init.c File Reference

Initialization routines for the driver unit.

`#include <stdio.h>`

`#include <avr/io.h>`

`#include <avr/interrupt.h>`

Defines

- #define OCR0_1MS 14

  Used for timer compare to match 1 ms.

Functions

- void init_timer_0 (void)
- void init_timer_2 (void)
- void init_ports (void)

6.11.1 Detailed Description

Initialization routines for the driver unit.

Author:

Mikael Larsmark, SM2WMV

Date:

2009-03-16

```
#include "driver_unit_v2/init.c"
```

Definition in file init.c.

6.11.2 Function Documentation

6.11.2.1 void init_ports (void)

Set the direction of the ports

Definition at line 53 of file init.c.

6.11.2.2 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 33 of file init.c.

References OCR0_1MS.

6.11.2.3 void init_timer_2 (void)

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Definition at line 42 of file init.c.

6.12 front_panel/init.c File Reference

Initialization routines for the front panel.

```
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "usart.h"
#include "board.h"
```

Defines

- #define OCR0_1MS 14

    Used for timer compare to match 1 ms.

Functions

- void init_usart_computer (void)
- void init_usart (void)
- void init_timer_0 (void)
- void init_timer_2 (void)
- void init_ports (void)
- void init_backlight (void)

6.12.1 Detailed Description

Initialization routines for the front panel.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/init.c"
```

Definition in file init.c.

6.12.2 Function Documentation

6.12.2.1 void init_backlight (void)

Initialize the backlight (Which is pulse width modulated so we can set the contrast)

Definition at line 105 of file init.c.

Referenced by main().

6.12.2.2 void init_ports (void)

Set the direction of the ports

Definition at line 72 of file init.c.

References BUTTON1_RX_BIT, BUTTON1_TX_BIT, BUTTON2_RX_BIT, BUTTON2_-TX_BIT, BUTTON3_RX_BIT, BUTTON3_TX_BIT, BUTTON4_RX_BIT, BUTTON4_-TX_BIT, BUTTON_AUX1_BIT, BUTTON_AUX2_BIT, BUTTON_MENU_BIT, BUTTON_PULSE_BIT, BUTTON_ROTATE_BIT, BUTTON_RXANT_BIT, BUTTON_-SUBMENU_BIT, and BUTTON_TXRX_BIT.

6.12.2.3 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 51 of file init.c.

References OCR0_1MS.

6.12.2.4 void init_timer_2 (void)

Initializes timer 2

Definition at line 62 of file init.c.

6.12.2.5 void init_usart (void)

Initializes the USART for the communication bus

Definition at line 41 of file init.c.

Referenced by main().

6.12.2.6 void init_usart_computer (void)

Init the UART for the computer communication

Definition at line 33 of file init.c.

References usart1_init(), usart1_receive_loopback(), and usart1_transmit().

Referenced by main().

6.13 general_io/init.c File Reference

Initialization routines for the General I/O card.

```
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>
```

Defines

- #define OCR0_1MS 14

  Used for timer compare to match 1 ms.

Functions

- void init_timer_0 (void)
- void init_timer_2 (void)
- void init_ports (void)

6.13.1 Detailed Description

Initialization routines for the General I/O card.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-05-18

```
#include "general_io/init.c"
```

Definition in file init.c.

6.13.2 Function Documentation

6.13.2.1 void init_ports (void)

Set the direction of the ports

Definition at line 56 of file init.c.

6.13.2.2 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 35 of file init.c.

References OCR0_1MS.

6.13.2.3 void init_timer_2 (void)

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Definition at line 45 of file init.c.

6.14 motherboard/init.c File Reference

Initialization routines for the motherboard.

`#include <stdio.h>`

`#include <avr/io.h>`

`#include <avr/interrupt.h>`

`#include "usart.h"`

Defines

- #define OCR0_1MS 14

  Used for timer compare to match 1 ms.

Functions

- void init_usart (void)

  Initializes the USART for the communication bus.

- void init_timer_0 (void)
- void init_ports (void)

  Set the direction of the ports.

6.14.1 Detailed Description

Initialization routines for the motherboard.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "init.h"
```

Definition in file init.c.

6.14.2 Function Documentation

6.14.2.1 void init_ports (void)

Set the direction of the ports.

Set the direction of the ports

Set the direction of the ports

Definition at line 54 of file init.c.

6.14.2.2 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 43 of file init.c.

References OCR0_1MS.

6.14.2.3 void init_usart (void)

Initializes the USART for the communication bus.

Initializes the USART for the communication bus

Definition at line 32 of file init.c.

References usart0_init(), usart0_receive_loopback(), and usart0_transmit().

6.15 driver_unit/init.h File Reference

Initialization routines for the driver unit.

Functions

- void init_timer_0 (void)
- void init_timer_2 (void)
- void init_ports (void)

    Set the direction of the ports.

6.15.1 Detailed Description

Initialization routines for the driver unit.

Author:

Mikael Larsmark, SM2WMV

Date:

2009-03-16

```
#include "driver_unit/init.h"
```

Definition in file init.h.

6.15.2 Function Documentation

6.15.2.1 void init_ports (void)

Set the direction of the ports.

Set the direction of the ports

Set the direction of the ports

Definition at line 56 of file init.c.

6.15.2.2 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 35 of file init.c.

6.15.2.3 void init_timer_2 (void)

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Initializes timer 2

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Definition at line 45 of file init.c.

6.16 driver_unit_v2/init.h File Reference

Initialization routines for the driver unit.

Functions

- void init_timer_0 (void)
- void init_timer_2 (void)
- void init_ports (void)

  Set the direction of the ports.

6.16.1 Detailed Description

Initialization routines for the driver unit.

Author:

Mikael Larsmark, SM2WMV

Date:

2009-03-16

```
#include "driver_unit_v2/init.h"
```

Definition in file init.h.

6.16.2 Function Documentation

6.16.2.1 void init_ports (void)

Set the direction of the ports.

Set the direction of the ports

Set the direction of the ports

Set the direction of the ports

Set the direction of the ports

Definition at line 56 of file init.c.

6.16.2.2 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 35 of file init.c.

6.16.2.3 void init_timer_2 (void)

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Initializes timer 2

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Definition at line 45 of file init.c.

6.17 front_panel/init.h File Reference

Initialization routines for the front panel.

Functions

- void init_timer_0 (void)
- void init_timer_2 (void)
- void init_ports (void)

  Set the direction of the ports.
- void init_usart_computer (void)
- void init_usart (void)

  Initializes the USART for the communication bus.
- void init_backlight (void)

6.17.1 Detailed Description

Initialization routines for the front panel.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/init.h"
```

Definition in file init.h.

6.17.2 Function Documentation

6.17.2.1 void init_backlight (void)

Initialize the backlight (Which is pulse width modulated so we can set the contrast)

Definition at line 105 of file init.c.

Referenced by main().

6.17.2.2 void init_ports (void)

Set the direction of the ports.

Set the direction of the ports

Set the direction of the ports

Set the direction of the ports

Set the direction of the ports

Definition at line 56 of file init.c.

6.17.2.3 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 35 of file init.c.

6.17.2.4 void init_timer_2 (void)

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Initializes timer 2

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Definition at line 45 of file init.c.

6.17.2.5 void init_usart (void)

Initializes the USART for the communication bus.

Initializes the USART for the communication bus

Definition at line 41 of file init.c.

6.17.2.6 void init_usart_computer (void)

Init the UART for the computer communication

Definition at line 33 of file init.c.

References usart1_init(), usart1_receive_loopback(), and usart1_transmit().

Referenced by main().

6.18 general_io/init.h File Reference

Initialization routines for the General I/O card.

Functions

- void init_timer_0 (void)
- void init_timer_2 (void)
- void init_ports (void)

  Set the direction of the ports.

6.18.1 Detailed Description

Initialization routines for the General I/O card.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-05-18

```
#include "general_io/init.h"
```

Definition in file init.h.

6.18.2 Function Documentation

6.18.2.1 void init_ports (void)

Set the direction of the ports.

Set the direction of the ports

Set the direction of the ports

Set the direction of the ports

Set the direction of the ports

Definition at line 56 of file init.c.

6.18.2.2 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 35 of file init.c.

6.18.2.3 void init_timer_2 (void)

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Initializes timer 2

Initializes timer 2, used for the communication bus and the interrupt is caught in bus.c

Definition at line 45 of file init.c.

6.19 motherboard/init.h File Reference

Initialization routines for the motherboard.

Functions

- void init_timer_0 (void)
- void init_ports (void)

  Set the direction of the ports.
- void init_usart (void)

  Initializes the USART for the communication bus.

6.19.1 Detailed Description

Initialization routines for the motherboard.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "init.h"
```

Definition in file init.h.

6.19.2 Function Documentation

6.19.2.1 void init_ports (void)

Set the direction of the ports.

Set the direction of the ports

Set the direction of the ports

Set the direction of the ports

Set the direction of the ports

Definition at line 56 of file init.c.

6.19.2.2 void init_timer_0 (void)

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Initialize timer0 to use the main crystal clock and the output compare interrupt feature to generate an interrupt approximately once per millisecond to use as a general purpose time base.

Definition at line 35 of file init.c.

References OCR0_1MS.

6.19.2.3 void init_usart (void)

Initializes the USART for the communication bus.

Initializes the USART for the communication bus

Definition at line 41 of file init.c.

References usart0_init(), usart0_receive_loopback(), and usart0_transmit().

Referenced by main().

6.20 driver_unit/main.c File Reference

Main file of the driver unit.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "main.h"
#include "board.h"
#include "init.h"
#include "../i2c.h"
#include "../delay.h"
#include "../wmv_bus/bus.h"
#include "../wmv_bus/bus_rx_queue.h"
#include "../wmv_bus/bus_tx_queue.h"
#include "../wmv_bus/bus_commands.h"
```

Defines

- #define LM76_ADDR 0x90

  The address of the LM76 temperature sensor connected to the I2C bus.

Functions

- void activate_output (unsigned char from_addr, unsigned char index, unsigned char type)

  Activate a driver output This function is used to activate an output on the driver unit. It will remember which device that sent the request for an activation so that the driver_unit will remember it when that device goes offline so it can shut the outputs off.

- void deactivate_output (unsigned char from_addr, unsigned char index)

  Deactivate a driver output This function is used to deactivate an output on the driver unit. It will remember which device that sent the request for an deactivation so that the driver_unit will remember it when that device goes offline so it can shut the outputs off.

- unsigned int lm76_get_temp (void)

  Retrieve the temperature from the LM76 sensor This function is used to retrieve the temperature from the LM76 sensor that does exist on the driver_unit.

- void bus_parse_message (void)

  Parse a message and exectute the proper commands This function is used to parse a message that was receieved on the bus that is located in the RX queue.

- unsigned char read_ext_addr (void)

Read the external DIP-switch. This function is used to read the external offset address on the driver_unit.

- int main (void)
- ISR (SIG_OUTPUT_COMPARE0)

Variables

- driver_status_struct driver_status

  A status structure of the driver unit outputs.
- unsigned int counter_compare0 = 0

  Counter to keep track of the numbers of ticks from timer0.
- unsigned int counter_sync = 0

  Counter to keep track of the time elapsed since the last sync message was sent.
- unsigned int counter_ping_interval = 0

  Counter to keep track of when to send a ping out on the bus.

6.20.1 Detailed Description

Main file of the driver unit.

Author:

Mikael Larsmark, SM2WMV

Date:

2008-04-06

```
#include "driver_unit/main.c"
```

Definition in file main.c.

6.20.2 Function Documentation

6.20.2.1 void activate_output (unsigned char *from_addr*, unsigned char *index*, unsigned char *type*)

Activate a driver output This function is used to activate an output on the driver unit. It will remember which device that sent the request for an activation so that the driver_unit will remember it when that device goes offline so it can shut the outputs off.

Parameters:

from_addr The device that sent the request of activating an output

index The index of which output to activate

type Which type of output this is, usually is the BUS command

Definition at line 60 of file main.c.

References DRIVER_OUTPUT_1, DRIVER_OUTPUT_10, DRIVER_OUTPUT_11, DRIVER_OUTPUT_12, DRIVER_OUTPUT_13, DRIVER_OUTPUT_14, DRIVER_-OUTPUT_15, DRIVER_OUTPUT_16, DRIVER_OUTPUT_17, DRIVER_OUTPUT_18, DRIVER_OUTPUT_19, DRIVER_OUTPUT_2, DRIVER_OUTPUT_20, DRIVER_-OUTPUT_3, DRIVER_OUTPUT_4, DRIVER_OUTPUT_5, DRIVER_OUTPUT_6, DRIVER_OUTPUT_7, DRIVER_OUTPUT_8, DRIVER_OUTPUT_9, driver_status_-struct::driver_output_owner, driver_status_struct::driver_output_state, and driver_status_-struct::driver_output_type.

Referenced by bus_parse_message(), and parse_internal_comm_message().

6.20.2.2 void deactivate_output (unsigned char *from_addr*, unsigned char *index*)

Deactivate a driver output This function is used to deactivate an output on the driver unit. It will remember which device that sent the request for an deactivation so that the driver_unit will remember it when that device goes offline so it can shut the outputs off.

Parameters:

from_addr The device that sent the request of deactivating the output

index The index of which output to deactivate

Definition at line 118 of file main.c.

References DRIVER_OUTPUT_1, DRIVER_OUTPUT_10, DRIVER_OUTPUT_11, DRIVER_OUTPUT_12, DRIVER_OUTPUT_13, DRIVER_OUTPUT_14, DRIVER_-OUTPUT_15, DRIVER_OUTPUT_16, DRIVER_OUTPUT_17, DRIVER_OUTPUT_18, DRIVER_OUTPUT_19, DRIVER_OUTPUT_2, DRIVER_OUTPUT_20, DRIVER_-OUTPUT_3, DRIVER_OUTPUT_4, DRIVER_OUTPUT_5, DRIVER_OUTPUT_6, DRIVER_OUTPUT_7, DRIVER_OUTPUT_8, DRIVER_OUTPUT_9, driver_status_-struct::driver_output_owner, driver_status_struct::driver_output_state, and driver_status_-struct::driver_output_type.

Referenced by bus_parse_message(), main(), and parse_internal_comm_message().

6.20.2.3 ISR (SIG_OUTPUT_COMPARE0)

Output compare 0 interrupt - "called" with 1ms intervals

Definition at line 326 of file main.c.

References bus_add_tx_message(), bus_allowed_to_send(), BUS_BROADCAST_ADDR, BUS_CMD_PING, BUS_DEVICE_STATUS_MESSAGE_INTERVAL, bus_get_address(), counter_compare0, counter_ping_interval, and DEVICE_ID_DRIVER_POS.

6.20.2.4 unsigned int lm76_get_temp (void)

Retrieve the temperature from the LM76 sensor This function is used to retrieve the temperature from the LM76 sensor that does exist on the driver_unit.

Returns:

The temperature but not in float format

Definition at line 173 of file main.c.

References i2cMasterReceiveNI(), and LM76_ADDR.

Referenced by bus_parse_message().

6.20.2.5 int main (void)

Main function of the driver unit

Definition at line 277 of file main.c.

References bus_check_tx_status(), bus_get_address(), bus_init(), bus_parse_message(), bus_set_address(), bus_set_is_master(), deactivate_output(), driver_status_struct::driver_-output_state, init_ports(), init_timer_0(), init_timer_2(), read_ext_addr(), rx_queue_is_-empty(), and tx_queue_is_empty().

6.20.2.6 unsigned char read_ext_addr (void)

Read the external DIP-switch. This function is used to read the external offset address on the driver_unit.

Returns:

The address of the external DIP-switch

Definition at line 272 of file main.c.

Referenced by main().

6.21 driver_unit_v2/main.c File Reference

Main file of the driver unit.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "main.h"
#include "board.h"
#include "init.h"
#include "../i2c.h"
#include "../delay.h"
#include "../wmv_bus/bus.h"
#include "../wmv_bus/bus_ping.h"
#include "../wmv_bus/bus_rx_queue.h"
#include "../wmv_bus/bus_tx_queue.h"
#include "../wmv_bus/bus_commands.h"
```

Defines

- #define LM76_ADDR 0x90

  The address of the LM76 temperature sensor connected to the I2C bus.

Functions

- void activate_output (unsigned char from_addr, unsigned char index, unsigned char type)

  Activate a driver output This function is used to activate an output on the driver unit. It will remember which device that sent the request for an activation so that the driver_unit will remember it when that device goes offline so it can shut the outputs off.

- void deactivate_output (unsigned char from_addr, unsigned char index)

  Deactivate a driver output This function is used to deactivate an output on the driver unit. It will remember which device that sent the request for an deactivation so that the driver_unit will remember it when that device goes offline so it can shut the outputs off.

- unsigned int lm76_get_temp (void)

  Retrieve the temperature from the LM76 sensor This function is used to retrieve the temperature from the LM76 sensor that does exist on the driver_unit.

- void bus_parse_message (void)

  Parse a message and exectute the proper commands This function is used to parse a message that was receieved on the bus that is located in the RX queue.

- unsigned char read_ext_addr (void)

Read the external DIP-switch. This function is used to read the external offset address on the driver_unit.

- void set_ptt_led_status (unsigned char state)

  Set the PTT led status.

- unsigned char get_ptt_status (void)

  Check the status of the external PTT lines.

- int main (void)
- ISR (SIG_OUTPUT_COMPARE0)

  Output compare 0 interrupt - "called" with 1ms intervals.

Variables

- unsigned char device_id

  Contains info if the module is a positive or negative driver.

- driver_status_struct driver_status

  A status structure of the driver unit outputs.

- unsigned int counter_compare0 = 0

  Counter to keep track of the numbers of ticks from timer0.

- unsigned int counter_sync = 0

  Counter to keep track of the time elapsed since the last sync message was sent.

- unsigned int counter_ping_interval = 0

  Counter to keep track of when to send a ping out on the bus.

- unsigned char check_ptt_status = 0

  Flag which is set when we wish to poll the PTT status.

6.21.1 Detailed Description

Main file of the driver unit.

Author:

Mikael Larsmark, SM2WMV

Date:

2009-03-16

```
#include "driver_unit_v2/main.c"
```

Definition in file main.c.

6.21.2 Function Documentation

6.21.2.1 void activate_output (unsigned char *from_addr*, unsigned char *index*, unsigned char *type*)

Activate a driver output This function is used to activate an output on the driver unit. It will remember which device that sent the request for an activation so that the driver_unit will remember it when that device goes offline so it can shut the outputs off.

Parameters:

from_addr The device that sent the request of activating an output

index The index of which output to activate

type The type of output this is, usually is the bus command

Definition at line 67 of file main.c.

References driver_status_struct::driver_output_owner, driver_status_struct::driver_output_-state, and driver_status_struct::driver_output_type.

6.21.2.2 void deactivate_output (unsigned char *from_addr*, unsigned char *index*)

Deactivate a driver output This function is used to deactivate an output on the driver unit. It will remember which device that sent the request for an deactivation so that the driver_unit will remember it when that device goes offline so it can shut the outputs off.

Parameters:

from_addr The device that sent the request of deactivating the output

index The index of which output to deactivate

Definition at line 125 of file main.c.

References driver_status_struct::driver_output_owner, driver_status_struct::driver_output_-state, and driver_status_struct::driver_output_type.

6.21.2.3 unsigned char get_ptt_status (void)

Check the status of the external PTT lines.

Returns:

A byte which contains info of the state of the PTT lines. 0 = R1, 1 = R2 etc

Definition at line 364 of file main.c.

References status.

Referenced by main().

6.21.2.4 unsigned int lm76_get_temp (void)

Retrieve the temperature from the LM76 sensor This function is used to retrieve the temperature from the LM76 sensor that does exist on the driver_unit.

Returns:

The temperature but not in float format

Definition at line 180 of file main.c.

References i2cMasterReceiveNI(), and LM76_ADDR.

6.21.2.5 int main (void)

Main function of the driver unit

Definition at line 380 of file main.c.

References bus_add_tx_message(), bus_allowed_to_send(), BUS_BROADCAST_ADDR, bus_check_tx_status(), BUS_CMD_PING, BUS_CMD_SYNC, BUS_DEVICE_STATUS_-MESSAGE_INTERVAL, bus_get_address(), bus_get_device_count(), bus_init(), bus_-is_master(), BUS_MASTER_SYNC_INTERVAL, bus_parse_message(), bus_set_address(), bus_set_is_master(), check_ptt_status, counter_ping_interval, counter_sync, deactivate_-output(), DEF_NR_DEVICES, device_count, device_id, DEVICE_ID_DRIVER_NEG, DEVICE_ID_DRIVER_POS, driver_status_struct::driver_output_state, get_ptt_status(), init_ports(), init_timer_0(), init_timer_2(), read_ext_addr(), rx_queue_is_empty(), set_-ptt_led_status(), and tx_queue_is_empty().

6.21.2.6 unsigned char read_ext_addr (void)

Read the external DIP-switch. This function is used to read the external offset address on the driver_unit.

Returns:

The address of the external DIP-switch

Definition at line 352 of file main.c.

6.22 front_panel/main.c File Reference

Main file of the front panel.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include <avr/wdt.h>
#include "main.h"
#include "board.h"
#include "usart.h"
#include "init.h"
#include "display.h"
#include "glcd.h"
#include "ks0108.h"
#include "led_control.h"
#include "../delay.h"
#include "../i2c.h"
#include "../global.h"
#include "../event_queue.h"
#include "interrupt_handler.h"
#include "eeprom.h"
#include "ds1307.h"
#include "computer_interface.h"
#include "antenna_ctrl.h"
#include "eeprom_m24.h"
#include "radio_interface.h"
#include "menu.h"
#include "rotary_encoder.h"
#include "event_handler.h"
#include "powermeter.h"
#include "../wmv_bus/bus.h"
#include "../wmv_bus/bus_ping.h"
#include "../wmv_bus/bus_rx_queue.h"
#include "../wmv_bus/bus_tx_queue.h"
#include "../wmv_bus/bus_commands.h"
```

```
#include "../internal_comm.h"
#include "../internal_comm_commands.h"
```

Functions

- void clear_screensaver_timer (void)

  Clear the screensaver timer.

- void event_add_message (void(*func), unsigned int offset, unsigned char id)
- unsigned char ext_key_get_assignment (unsigned char index)
- void ext_key_set_assignment (unsigned char index, unsigned char func)
- void event_run (void)

  Run the first function in the event queue.

- void main_update_display (void)

  Sets the flag that the display should be updated.

- void shutdown_device (void)

  Send a message to the motherboard that the openASC box should be shut off. Will deactivate the power supply relay.

- void set_tx_ant_leds (void)

  Set the TX antenna leds according to the status of status.selected_ant.

- void set_knob_function (unsigned char function)

  Set the rotary knob function.

- void main_save_settings (void)

  Save runtime settings etc to the EEPROM.

- void load_settings (void)

  Load all settings from the EEPROM.

- void main_update_ptt_status (void)

  Function which updates the status of the PTT This function will check various sources if it is for example OK to transmit or not. This function also updates the color of the PTT led. It does also set the main_set_inhibit_state() status which is used at various places to make the sequencing etc safe.

- void main_set_inhibit_state (enum enum_inhibit_state state)

  Set the inhibit state This function is used to set the inhibit state, which is used to check at various places if it is safe for example to transmit, change band or change antennas.

- enum enum_inhibit_state main_get_inhibit_state (void)

  Get the inhibit state This function is used to get the inhibit state, which is used to check at various places if it is safe for example to transmit, change band or change antennas.

- void send_ping (void)

  Send a ping message out on the bus.

- int main (void)
- ISR (SIG_OUTPUT_COMPARE0A)
- ISR (SIG_OVERFLOW0)

Variables

- struct_setting settings

  Settings struct.

- unsigned char radio_rx_data_counter = 0

  Counter to keep track of when a character for the CAT was last received.

- unsigned int counter_compare0 = 0

  Counter which counts up each time a compare0 interrupt has occured.

- unsigned int counter_sync = 32000

  Counter which is used to keep track of when we last received a sync message from the bus.

- unsigned char counter_poll_buttons = 0

  Counter which keeps track of when we should poll the buttons.

- unsigned char counter_poll_ext_devices = 0

  Counter which keeps track of when we should poll the external inputs.

- unsigned int counter_screensaver_timeout = 0

  Counter which keeps track of the screensaver timeout.

- unsigned int counter_ping_interval = 0

  Counter which keeps track of when we should send out a ping to the communication bus.

- unsigned int counter_ms = 0

  Counter which counts up each millisecond.

- unsigned char counter_poll_rotary_encoder = 0

  Counter which keeps track when we should poll the rotary encoder.

- unsigned int counter_poll_radio = 0

  Counter which keeps track of when we should poll the radio.

- unsigned int counter_last_pulse_event = 0

  Counter which keeps track of when the last pulse event did occur. This is used to sense if we should change rx antennas.

- unsigned int counter_event_timer = 0

  After the counter reaches half of it's limit we remove that number from it by calling the function event_queue_wrap().

- unsigned char device_count = 0

  The number of devices on the bus.

- unsigned int main_flags = 0

    Different flags, description is found in main.h.

- unsigned char ping_message [3]

    Ping message of the openASC device.

- unsigned char device_started = 0

    Variable to check if the device has actually gone through all init steps.

6.22.1 Detailed Description

Main file of the front panel.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/main.c"
```

Definition in file main.c.

6.22.2 Function Documentation

6.22.2.1 void event_add_message (void ∗ *func*, unsigned int *offset*, unsigned char *id*)

Add a message to the event queue which will be run at the correct time

Parameters:

func A function pointer to the function we want to run

offset the time in ms when we want our function to be run

id Which type of event this is

Definition at line 115 of file main.c.

References counter_event_timer, event_in_queue(), and event_queue_add().

Referenced by display_update_screensaver(), event_internal_comm_parse_message(), sequencer_computer_rts_activated(), sequencer_computer_rts_deactivated(), sequencer_-footsw_pressed(), and sequencer_footsw_released().

6.22.2.2 unsigned char ext_key_get_assignment (unsigned char *index*)

Get the key assignment index

Parameters:

index The index of which task we wish to check

Returns:

The current task index, can be found in event_handler.h

Definition at line 131 of file main.c.

References struct_setting::ext_key_assignments.

Referenced by event_handler_process_ps2().

6.22.2.3 void ext_key_set_assignment (unsigned char *index*, unsigned char *func*)

Set the key assignment task

Parameters:

index The index of which task we wish to set

func The function we wish to assign to the assignment index

Definition at line 138 of file main.c.

References struct_setting::ext_key_assignments.

6.22.2.4 ISR (SIG_OVERFLOW0)

Output overflow 0 interrupt

Definition at line 748 of file main.c.

6.22.2.5 ISR (SIG_OUTPUT_COMPARE0A)

Output compare 0 interrupt - "called" with 1ms intervals

Definition at line 620 of file main.c.

References antenna_ctrl_get_rotatable(), antenna_ctrl_get_rotator_flags(), struct_runtime_-settings::band_change_mode, BAND_CHANGE_MODE_AUTO, bus_is_master(), bus_-ping_tick(), counter_event_timer, counter_last_pulse_event, counter_ms, counter_ping_-interval, counter_poll_buttons, counter_poll_ext_devices, counter_poll_radio, counter_-poll_rotary_encoder, counter_screensaver_timeout, counter_sync, device_started, event_in_-queue(), event_queue_get(), event_queue_wrap(), FLAG_BLINK_BAND_LED, FLAG_-CHANGE_RX_ANT, FLAG_CHANGE_SUBMENU, FLAG_LAST_ANTENNA_BLINK, FLAG_NO_ROTATION, FLAG_POLL_BUTTONS, FLAG_POLL_EXT_DEVICES, FLAG_POLL_PULSE_SENSOR, FLAG_POLL_RADIO, FLAG_PROCESS_RX_ANT_-CHANGE, FLAG_PROCESS_SUBMENU_CHANGE, FLAG_RUN_EVENT_QUEUE, FUNC_STATUS_SELECT_ANT_ROTATE, struct_status::function_status, internal_-comm_1ms_timer(), INTERVAL_POLL_BUTTONS, INTERVAL_POLL_EXT_DEVICES, INTERVAL_POLL_ROTARY_ENCODER, led_set_rotation_active(), led_set_tx_ant(), LED_STATE_OFF, LED_STATE_ON, main_flags, powermeter_1ms_tick(), struct_-setting::powermeter_address, PULSE_SENSOR_RX_ANT_CHANGE_LIMIT, PULSE_-SENSOR_SUBMENU_CHANGE_LIMIT, radio_communicaton_timeout(), radio_interface_-get_poll_interval(), radio_rx_data_counter, RADIO_RX_DATA_TIMEOUT, runtime_-settings, and status.

6.22.2.6 int main (void)

Main function of the front panel

Definition at line 293 of file main.c.

References struct_runtime_settings::amplifier_ptt_output, antenna_ctrl_change_rx_-ant(), struct_runtime_settings::band_change_mode, BAND_CHANGE_MODE_AUTO, BAND_CHANGE_MODE_MANUAL, band_ctrl_change_band(), band_ctrl_change_-band_portion(), band_ctrl_load_band_limits(), bus_add_tx_message(), bus_allowed_-to_send(), BUS_BROADCAST_ADDR, bus_check_tx_status(), BUS_CMD_SYNC, BUS_DEVICE_STATUS_MESSAGE_INTERVAL, bus_get_address(), bus_init(), bus_-is_master(), BUS_MASTER_SYNC_INTERVAL, bus_set_address(), bus_set_is_master(), struct_status::buttons_current_state, struct_status::buttons_last_state, computer_-interface_activate_setup(), computer_interface_deactivate_setup(), computer_interface_-init(), computer_interface_is_active(), computer_interface_parse_data(), computer_-interface_send_data(), counter_last_pulse_event, counter_ping_interval, counter_sync, struct_status::current_band_portion, device_count, DEVICE_ID_MAINBOX, device_-started, display_set_backlight(), display_setup_view(), ds1307_init(), eeprom_create_table(), eeprom_read_startup_byte(), eeprom_read_table(), eeprom_save_runtime_settings(), eeprom_write_startup_byte(), event_bus_parse_message(), event_internal_comm_parse_-message(), event_poll_buttons(), event_poll_ext_device(), event_pulse_sensor_down(), event_pulse_sensor_up(), event_run(), event_update_display(), struct_status::ext_devices_-current_state, struct_status::ext_devices_last_state, FLAG_BLINK_BAND_LED, FLAG_-CHANGE_RX_ANT, FLAG_CHANGE_SUBMENU, FLAG_LAST_BAND_BLINK, FLAG_POLL_BUTTONS, FLAG_POLL_EXT_DEVICES, FLAG_POLL_PULSE_-SENSOR, FLAG_POLL_RADIO, FLAG_PROCESS_RX_ANT_CHANGE, FLAG_-PROCESS_SUBMENU_CHANGE, FLAG_RUN_EVENT_QUEUE, FLAG_UPDATE_-DISPLAY, glcd_init(), i2c_init(), ih_poll_ext_devices(), INHIBIT_NOT_OK_TO_-SEND, struct_runtime_settings::inhibit_state, init_backlight(), init_ports(), init_timer_0(), init_timer_2(), init_usart(), init_usart_computer(), internal_comm_init(), internal_-comm_poll_rx_queue(), internal_comm_poll_tx_queue(), KNOB_FUNCTION_AUTO, struct_runtime_settings::lcd_backlight_value, led_set_all(), led_set_band(), led_set_-band_none(), led_set_ptt(), LED_STATE_OFF, LED_STATE_ON, LED_STATE_PTT_-INHIBIT, load_settings(), main_flags, main_set_inhibit_state(), menu_init(), struct_-setting::network_address, struct_setting::network_device_count, struct_setting::network_-device_is_master, struct_status::new_band, struct_status::new_band_portion, ping_-message, struct_setting::powermeter_address, powermeter_init(), powermeter_process_-tasks(), struct_setting::powermeter_update_rate_bargraph, struct_setting::powermeter_-update_rate_text, struct_setting::powermeter_vswr_limit, radio_get_band_portion(), radio_get_current_band(), RADIO_INTERFACE_BCD, radio_interface_get_interface(), RADIO_INTERFACE_MANUAL, radio_poll_status(), radio_process_tasks(), struct_-runtime_settings::radio_ptt_output, rotary_encoder_poll(), struct_status::rotator_step_-resolution, runtime_settings, rx_queue_is_empty(), struct_status::selected_band, struct_-status::selected_rx_antenna, send_ping(), set_knob_function(), status, struct_status::sub_-menu_antenna_index, sub_menu_get_current_pos(), sub_menu_send_data_to_bus(), tx_queue_dropall(), tx_queue_is_empty(), and usart0_transmit().

6.22.2.7 enum enum_inhibit_state main_get_inhibit_state (void)

Get the inhibit state This function is used to get the inhibit state, which is used to check at various places if it is safe for example to transmit, change band or change antennas.

Returns:

The current inhibit status

Definition at line 280 of file main.c.

References struct_runtime_settings::inhibit_state, and runtime_settings.

Referenced by band_ctrl_change_band(), event_pulse_sensor_down(), event_pulse_sensor_-up(), event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_button3_-pressed(), event_tx_button4_pressed(), radio_poll_status(), sequencer_computer_rts_-activated(), and sequencer_footsw_pressed().

6.22.2.8 void main_set_inhibit_state (enum enum_inhibit_state *state*)

Set the inhibit state This function is used to set the inhibit state, which is used to check at various places if it is safe for example to transmit, change band or change antennas.

Parameters:

state The state we wish to set the inhibit status to

Definition at line 272 of file main.c.

References struct_runtime_settings::inhibit_state, and runtime_settings.

Referenced by main(), and main_update_ptt_status().

6.22.2.9 void set_knob_function (unsigned char *function*)

Set the rotary knob function.

Parameters:

function Which type of action should occur when the knob is turned

Definition at line 178 of file main.c.

References struct_runtime_settings::band_change_mode, BAND_CHANGE_MODE_-MANUAL, struct_status::knob_function, KNOB_FUNCTION_AUTO, KNOB_-FUNCTION_NONE, KNOB_FUNCTION_SELECT_BAND, runtime_settings, and status.

Referenced by band_ctrl_change_band(), event_poll_buttons(), event_rotate_button_-pressed(), event_rxant_button_pressed(), event_sub_button_pressed(), main(), and menu_-action().

6.22.3 Variable Documentation

6.22.3.1 unsigned char radio_rx_data_counter = 0

Counter to keep track of when a character for the CAT was last received.

External variable of the radio rx data counter used for a timeout.

Definition at line 69 of file main.c.

Referenced by ISR().

6.23 general_io/main.c File Reference

Main file of the General I/O card.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "main.h"
#include "board.h"
#include "init.h"
#include "../i2c.h"
#include "../delay.h"
#include "../wmv_bus/bus.h"
#include "../wmv_bus/bus_ping.h"
#include "../wmv_bus/bus_rx_queue.h"
#include "../wmv_bus/bus_tx_queue.h"
#include "../wmv_bus/bus_commands.h"
```

Functions

- void bus_parse_message (void)

  Parse a message and exectute the proper commands This function is used to parse a message that was receieved on the bus that is located in the RX queue.

- unsigned char read_ext_addr (void)

  Read the external DIP-switch. This function is used to read the external offset address on the General I/O card.

- int main (void)
- ISR (SIG_OUTPUT_COMPARE0)

  Output compare 0 interrupt - "called" with 1ms intervals.

Variables

- unsigned char device_id

  Contains info of the driver type.

- unsigned int counter_compare0 = 0

  Counter to keep track of the numbers of ticks from timer0.

- unsigned int counter_sync = 0

  Counter to keep track of the time elapsed since the last sync message was sent.

- unsigned int counter_ping_interval = 0

  Counter to keep track of when to send a ping out on the bus.

6.23.1 Detailed Description

Main file of the General I/O card.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-05-18

```
#include "general_io/main.c"
```

Definition in file main.c.

6.23.2 Function Documentation

6.23.2.1 int main (void)

Main function of the General I/O

Definition at line 77 of file main.c.

References bus_add_tx_message(), bus_allowed_to_send(), BUS_BROADCAST_ADDR, bus_check_tx_status(), BUS_CMD_PING, BUS_CMD_SYNC, BUS_DEVICE_STATUS_-MESSAGE_INTERVAL, bus_get_address(), bus_get_device_count(), bus_init(), bus_-is_master(), BUS_MASTER_SYNC_INTERVAL, bus_parse_message(), bus_set_-address(), bus_set_is_master(), counter_ping_interval, counter_sync, DEF_NR_DEVICES, DEFAULT_STARTUP_DELAY, device_count, device_id, DEVICE_ID_GENERAL_IO, init_ports(), init_timer_0(), init_timer_2(), read_ext_addr(), rx_queue_is_empty(), and tx_queue_is_empty().

6.23.2.2 unsigned char read_ext_addr (void)

Read the external DIP-switch. This function is used to read the external offset address on the General I/O card.

Returns:

The address of the external DIP-switch

Definition at line 72 of file main.c.

6.24 motherboard/main.c File Reference

Main file of the motherboard.

```
#include <avr/parity.h>
#include "main.h"
#include "board.h"
#include "usart.h"
#include "init.h"
#include "../delay.h"
#include "../internal_comm.h"
#include "../internal_comm_commands.h"
#include "../wmv_bus/bus_commands.h"
#include "computer_interface.h"
```

Defines

- #define PS2_CLK_LOW PORTE &= ~(1<<6)
  Macro to put PS2 CLK output LOW.
- #define PS2_CLK_HIGH PORTE |= (1<<6)
  Macro to put PS2 CLK output HIGH.
- #define PS2_DATA_LOW PORTA &= ~(1<<3)
  Macro to put PS2 DATA output LOW.
- #define PS2_DATA_HIGH PORTA |= (1<<3)
  Macro to put PS2 DATA output HIGH.

Functions

- void __inline__ tiny_delay (void)
  Tiny delay function.
- void activate_output (unsigned char index, unsigned char type)
  Activate a driver output This function is used to activate an output on the relay driver output in the openASC It controls both the sink and source output at the same time.
- void deactivate_output (unsigned char index)
  Deactivate a driver output This function is used to deactivate an output on the relay outputs It controls both the sink and source output at the same time.
- void parse_internal_comm_message (UC_MESSAGE message)
  Parse an internal communication message.
- void ps2_keyboard_send (unsigned char cmd)

Send a command to the PS/2 keyboard output/input.

- void ps2_process_key (unsigned char key_code)
  Process a keystroke.
- int main (void)
  Main function of the motherboard.
- ISR (SIG_OUTPUT_COMPARE0)
  Output compare 0 interrupt - "called" with 1ms intervals.
- ISR (SIG_OVERFLOW0)
  Output overflow 0 interrupt.
- **ISR** (SIG_INTERRUPT6)

Variables

- unsigned char temp_count = 0
  Counter used for the PS/2 decoding.
- unsigned int driver_output_state = 0
  The driver output state.
- unsigned int driver_output_type [12]
  The type of driver output.
- unsigned char btn_on_off_last_state = 1
  Variable used to keep track of the last state of the ON/OFF button so we can see when it has been pressed and released.
- unsigned int counter_time_start = 0
- unsigned int counter_ps2 = 0
  Counter which keeps track of the PS/2 decoding.
- PS2_STRUCT ps2
  PS/2 struct.

6.24.1 Detailed Description

Main file of the motherboard.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "motherboard/main.c"
```

Definition in file main.c.

6.24.2 Function Documentation

6.24.2.1 void activate_output (unsigned char *index*, unsigned char *type*)

Activate a driver output This function is used to activate an output on the relay driver output in the openASC It controls both the sink and source output at the same time.

Parameters:

index The index of which output to activate (1-12)

type The command that did activate this output

Definition at line 79 of file main.c.

References DRIVER_OUTPUT_1, DRIVER_OUTPUT_10, DRIVER_OUTPUT_11, DRIVER_OUTPUT_12, DRIVER_OUTPUT_2, DRIVER_OUTPUT_3, DRIVER_-OUTPUT_4, DRIVER_OUTPUT_5, DRIVER_OUTPUT_6, DRIVER_OUTPUT_7, DRIVER_OUTPUT_8, DRIVER_OUTPUT_9, driver_output_state, and driver_output_-type.

6.24.2.2 void deactivate_output (unsigned char *index*)

Deactivate a driver output This function is used to deactivate an output on the relay outputs It controls both the sink and source output at the same time.

Parameters:

index The index of which output to activate (1-12)

Definition at line 118 of file main.c.

References DRIVER_OUTPUT_1, DRIVER_OUTPUT_10, DRIVER_OUTPUT_11, DRIVER_OUTPUT_12, DRIVER_OUTPUT_2, DRIVER_OUTPUT_3, DRIVER_-OUTPUT_4, DRIVER_OUTPUT_5, DRIVER_OUTPUT_6, DRIVER_OUTPUT_7, DRIVER_OUTPUT_8, DRIVER_OUTPUT_9, driver_output_state, and driver_output_-type.

6.24.2.3 void parse_internal_comm_message (UC_MESSAGE *message*)

Parse an internal communication message.

Parameters:

message The message that we wish to parse

Definition at line 154 of file main.c.

References activate_output(), AUX_X11_PIN3, AUX_X11_PIN4, AUX_X11_PIN5, AUX_-X11_PIN8, AUX_X11_PIN9, BUS_CMD_DRIVER_ACTIVATE_ANT_OUTPUT, BUS_-CMD_DRIVER_ACTIVATE_BAND_OUTPUT, BUS_CMD_DRIVER_ACTIVATE_-RX_ANT_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_ALL_ANT_OUTPUTS, BUS_CMD_DRIVER_DEACTIVATE_ALL_BAND_OUTPUTS, BUS_CMD_DRIVER_-DEACTIVATE_ALL_OUTPUTS, BUS_CMD_DRIVER_DEACTIVATE_ANT_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_RX_ANT_OUTPUT, UC_MESSAGE::cmd, computer_interface_send(), UC_MESSAGE::data, deactivate_output(), driver_output_-type, INT_COMM_AUX_CHANGE_OUTPUT_PIN, INT_COMM_GET_BAND_BCD_-STATUS, INT_COMM_PULL_THE_PLUG, INT_COMM_REDIRECT_DATA, internal_-comm_add_tx_message(), and UC_MESSAGE::length.

Referenced by main().

6.24.2.4 void ps2_keyboard_send (unsigned char *cmd*)

Send a command to the PS/2 keyboard output/input.

Parameters:

cmd The command we wish to send

Definition at line 279 of file main.c.

References PS2_STRUCT::bit_count, PS2_STRUCT::parity, PS2_CLK_HIGH, PS2_CLK_-LOW, PS2_DATA_HIGH, PS2_DATA_LOW, tiny_delay(), PS2_STRUCT::transmit, and PS2_STRUCT::tx_data.

6.24.2.5 void ps2_process_key (unsigned char *key_code*)

Process a keystroke.

Parameters:

key_code The key code which was received

Definition at line 301 of file main.c.

References INT_COMM_PS2_KEYPRESSED, and internal_comm_add_tx_message().

6.24.3 Variable Documentation

6.24.3.1 unsigned int counter_time_start = 0

Counter which keeps track of how long time ago it was since we started the box. This is used at startup so that we ignore button actions on the ON/OFF button for a certain time at startup

Definition at line 58 of file main.c.

Referenced by ISR().

6.25 driver_unit/main.h File Reference

Defines

- #define BUS_RX_QUEUE_SIZE 10

  The size of the RX queue in buffers.

- #define BUS_TX_QUEUE_SIZE 10

  The size of the TX queue in buffers.

6.25.1 Detailed Description

Definition in file main.h.

6.26 driver_unit_v2/main.h File Reference

Classes

- struct driver_status_struct

Defines

- #define BUS_RX_QUEUE_SIZE 10

  The size of the RX queue in buffers.

- #define BUS_TX_QUEUE_SIZE 10

  The size of the TX queue in buffers.

- #define FLAG_TXRX_MODE_ENABLED 0
- #define DRIVER_STATUS_OFF 0

  Driver status for output OFF.

- #define DRIVER_STATUS_ON 1

  Driver status for output ON.

6.26.1 Detailed Description

Definition in file main.h.

6.26.2 Define Documentation

6.26.2.1 #define FLAG_TXRX_MODE_ENABLED 0

Flag to indicate if the TX/RX mode is enabled

Definition at line 54 of file main.h.

6.27 front_panel/main.h File Reference

Main file of the front panel.

Classes

- struct struct_setting

  Settings struct.

- struct struct_status

  This struct only contains information that is temporary.

- struct struct_runtime_settings

  Settings like status but which should be saved into the EEPROM.

Defines

- #define FIRMWARE_REV "0.1b\0"

  The current firmware revision nr.

- #define ENABLE_TIMER0_INT() TIMSK0 |= (1<<OCIE0A);

  Macro to enable timer 0 interrupt.

- #define DISABLE_TIMER0_INT() TIMSK0 &= ~(1<<OCIE0A);

  Macro to disable timer 0 interrupt.

- #define BUS_STATUS_ALLOWED_TO_SEND_BIT 0

  Flag to indicate that the bus is allowed to transmit.

- #define BUS_STATUS_PREAMBLE_FOUND_BIT 1

  Flag to indicate that a preamble has been found in the bus.

- #define BUS_RX_QUEUE_SIZE 10

  The size of the RX queue in buffers.

- #define BUS_TX_QUEUE_SIZE 25

  The size of the TX queue in buffers.

- #define ANTENNA_EXIST_FLAG 0

  This flag is to indicate that the antenna exist.

- #define ANTENNA_ROTATOR_FLAG 1

  Flag if there is a rotator that can be controlled.

- #define ANTENNA_IS_MULTIBAND 2

  Flag if the antenna is a multiband antenna - Not implemented.

- #define ANTENNA_IN_USE_FLAG 3

Flag that shows if an antenna is occupied, used for multiband antennas primary - Not implemented.

- #define DISPLAY_SCREENSAVER_TIMEOUT 5000

  Screensaver timeout.

- #define RADIO_RX_DATA_TIMEOUT 10
- #define PTT_RADIO_BIT 0

  Indicate that radio is enabled or disabled.

- #define PTT_AMP_BIT 1

  Indicate that amp is enabled or disabled.

- #define INHIBIT_ENABLED_BIT 2

  Indicate that inhibit is enabled or disabled.

- #define BAND_CHANGE_MODE_MANUAL 0

  Band changes are done manually.

- #define BAND_CHANGE_MODE_AUTO 1

  Band changes are done automatically.

- #define FLAG_POLL_BUTTONS 0

  POLL BUTTONS, is set when a poll on the front panel buttons should occur.

- #define FLAG_POLL_EXT_DEVICES 1

  EXT DEVICES flag is set when a poll for external devices should occur.

- #define FLAG_RUN_EVENT_QUEUE 2

  Run the event first in the event queue.

- #define FLAG_UPDATE_DISPLAY 3

  FLAG to indicate that the display should be updated.

- #define FLAG_POLL_PULSE_SENSOR 4

  Poll the pulse sensor.

- #define FLAG_LAST_BAND_BLINK 5

  This flag indicates the state of the last BAND blink event, used to blink the LED when a new band change is in process.

- #define FLAG_LAST_ANTENNA_BLINK 6

  This flag is used to blink the antennas which can be rotated.

- #define FLAG_CHANGE_RX_ANT 7

  This flag is used to trigger an RX antenna change, after a certain amount of time which is set with the flag below.

- #define FLAG_PROCESS_RX_ANT_CHANGE 8

  Works together with the above flag, but this is set when the actual antenna change should occur.

- #define FLAG_BLINK_BAND_LED 9

  Blink the band led.

- #define FLAG_POLL_RADIO 10

  Indicate that we should poll the radio.

- #define FLAG_CHANGE_SUBMENU 11

  This flag is set to indicate that we have changed the sub menu.

- #define FLAG_PROCESS_SUBMENU_CHANGE 12

  This flag is to indicate that a sub menu change should occur, ie sent out on the bus.

- #define INTERVAL_POLL_BUTTONS 50

  The poll interval of the front panel buttons (unit = ms).

- #define INTERVAL_POLL_ROTARY_ENCODER 5

  The poll interval of the rotary encoder (unit = ms).

- #define INTERVAL_POLL_EXT_DEVICES 1

  The poll interval of the external devices as shown in board.h (unit = ms).

- #define KNOB_FUNCTION_NONE 0

  Knob function is to select RX antenna.

- #define KNOB_FUNCTION_RX_ANT 1

  Knob function is to select RX antenna.

- #define KNOB_FUNCTION_SELECT_BAND 2

  Knob function is to select band.

- #define KNOB_FUNCTION_SET_HEADING 3

  Knob function is to set the heading of a rotator.

- #define KNOB_FUNCTION_AUTO 4

  Auto select, pick the one which is most likely to be used.

- #define KNOB_FUNCTION_SET_SUBMENU 5

  Knob function set submenu option.

- #define RX_ANTENNA_NAME_LENGTH 15

  RX antenna name length.

- #define RX_ANTENNA_OUTPUT_STR_LENGTH 10

  RX antenna output str length.

- #define RX_ANTENNA_BAND_OUTPUT_STR_LENGTH 10

  RX antenna band output str length.

- #define ANTENNA_TEXT_SIZE 10

The max size of the antenna output str length.

- #define ANTENNA_OUTPUT_COMB_SIZE 10

  The max size of the output combination length.
- #define BAND_OUTPUT_STR_SIZE 10

  The max size of the band output str.
- #define SUB_MENU_ARRAY_STR_SIZE 10

  The max size of the sub menu array output str size.
- #define SUB_MENU_ARRAY_NAME_SIZE 3

  The size of the name of a 4-SQ.
- #define OUTPUT_ADDR_DELIMITER 0xFF

  The delimiter that seperates the outputs from which address they should be sent to.
- #define FUNC_STATUS_RXANT 0

  Define for function status.
- #define FUNC_STATUS_ROTATE 1

  Define for function status, that rotation is active.
- #define FUNC_STATUS_SELECT_ANT_ROTATE 2

  Define for function status, to select which antenna that should be rotated.
- #define FUNC_STATUS_SUBMENU 3

  Define for function status, to select sub menu.
- #define DISPLAY_LEVEL_LOGO 0

  Display level openASC logo.
- #define DISPLAY_LEVEL_BAND 1

  Display level current band.
- #define DISPLAY_LEVEL_SUBMENU 2

  Display level sub menu.
- #define CURRENT_DISPLAY_LOGO 0

  Current display is the openASC logo.
- #define CURRENT_DISPLAY_ANTENNA_INFO 1

  Current display is the antenna information.
- #define CURRENT_DISPLAY_MENU_SYSTEM 2

  Current display is the menu system.
- #define CURRENT_DISPLAY_SHUTDOWN_VIEW 3

  Current display is the shutdown in progress view.

- #define CURRENT_DISPLAY_POWERMETER_VIEW 4

    Current display power meter view.

- #define PULSE_SENSOR_RX_ANT_CHANGE_LIMIT 250

    The time from when a pulse sensor change occured to the actual change does happen, in ms.

- #define PULSE_SENSOR_SUBMENU_CHANGE_LIMIT 250

    The time from when a pulse sensor change occured to the actual change does happen, in ms.

- #define SUBMENU_NONE 0

    Sub menu type NONE.

- #define SUBMENU_VERT_ARRAY 1

    Sub menu type 4-SQ.

- #define SUBMENU_STACK 2

    Sub menu type stack.

- #define VIEW_ANTENNAS 0

    Flag used if we wish to view antennas.

Enumerations

- enum enum_inhibit_state { INHIBIT_OK_TO_SEND, INHIBIT_NOT_OK_TO_-SEND, INHIBIT_NOT_OK_TO_SEND_RADIO_TX }

    Different inhibit states.

Functions

- void main_update_ptt_status (void)

    Function which updates the status of the PTT This function will check various sources if it is for example OK to transmit or not. This function also updates the color of the PTT led. It does also set the main_set_inhibit_state() status which is used at various places to make the sequencing etc safe.

- void main_save_settings (void)

    Save runtime settings etc to the EEPROM.

- void event_add_message (void(*func), unsigned int offset, unsigned char id)
- unsigned char ext_key_get_assignment (unsigned char index)
- void ext_key_set_assignment (unsigned char index, unsigned char func)
- void main_update_display (void)

    Sets the flag that the display should be updated.

- void **check_knob_function** (void)
- void set_tx_ant_leds (void)

    Set the TX antenna leds according to the status of status.selected_ant.

- void set_knob_function (unsigned char function)

  Set the rotary knob function.

- void shutdown_device (void)

  Send a message to the motherboard that the openASC box should be shut off. Will deactivate the power supply relay.

- enum enum_inhibit_state main_get_inhibit_state (void)

  Get the inhibit state This function is used to get the inhibit state, which is used to check at various places if it is safe for example to transmit, change band or change antennas.

- void main_set_inhibit_state (enum enum_inhibit_state state)

  Set the inhibit state This function is used to set the inhibit state, which is used to check at various places if it is safe for example to transmit, change band or change antennas.

- void send_ping (void)

  Send a ping message out on the bus.

Variables

- struct_status status

  Contains different statuses of buttons etc.

- struct_runtime_settings runtime_settings

  Contains settings which will be saved and restored each time the box is turned on/off.

6.27.1 Detailed Description

Main file of the front panel.

Definition in file main.h.

6.27.2 Define Documentation

6.27.2.1 #define RADIO_RX_DATA_TIMEOUT 10

The limit (in ms) of the radio communication timeout. If this limit is reached the radio rx buffers will be cleared

Definition at line 58 of file main.h.

Referenced by ISR().

6.27.3 Enumeration Type Documentation

6.27.3.1 enum enum_inhibit_state

Different inhibit states.

Enumerator:

INHIBIT_OK_TO_SEND Inhibit state, OK to start a transmission.

INHIBIT_NOT_OK_TO_SEND Inhibit state, NOT OK to start a transmission.

INHIBIT_NOT_OK_TO_SEND_RADIO_TX Inhibit state, NOT OK to start a transmission, Radio is in TX.

Definition at line 187 of file main.h.

6.27.4 Function Documentation

6.27.4.1 void event_add_message (void ∗ *func*, unsigned int *offset*, unsigned char *id*)

Add a message to the event queue which will be run at the correct time

Parameters:

func A function pointer to the function we want to run

offset the time in ms when we want our function to be run

id Which type of event this is

Definition at line 115 of file main.c.

References counter_event_timer, event_in_queue(), and event_queue_add().

Referenced by display_update_screensaver(), event_internal_comm_parse_message(), sequencer_computer_rts_activated(), sequencer_computer_rts_deactivated(), sequencer_-footsw_pressed(), and sequencer_footsw_released().

6.27.4.2 unsigned char ext_key_get_assignment (unsigned char *index*)

Get the key assignment index

Parameters:

index The index of which task we wish to check

Returns:

The current task index, can be found in event_handler.h

Definition at line 131 of file main.c.

References struct_setting::ext_key_assignments.

Referenced by event_handler_process_ps2().

6.27.4.3 void ext_key_set_assignment (unsigned char *index*, unsigned char *func*)

Set the key assignment task

Parameters:

index The index of which task we wish to set *func* The function we wish to assign to the assignment index

Definition at line 138 of file main.c.

References struct_setting::ext_key_assignments.

6.27.4.4 enum enum_inhibit_state main_get_inhibit_state (void)

Get the inhibit state This function is used to get the inhibit state, which is used to check at various places if it is safe for example to transmit, change band or change antennas.

Returns:

The current inhibit status

Definition at line 280 of file main.c.

References struct_runtime_settings::inhibit_state, and runtime_settings.

Referenced by band_ctrl_change_band(), event_pulse_sensor_down(), event_pulse_sensor_-up(), event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_button3_-pressed(), event_tx_button4_pressed(), radio_poll_status(), sequencer_computer_rts_-activated(), and sequencer_footsw_pressed().

6.27.4.5 void main_set_inhibit_state (enum enum_inhibit_state *state*)

Set the inhibit state This function is used to set the inhibit state, which is used to check at various places if it is safe for example to transmit, change band or change antennas.

Parameters:

state The state we wish to set the inhibit status to

Definition at line 272 of file main.c.

References struct_runtime_settings::inhibit_state, and runtime_settings.

Referenced by main(), and main_update_ptt_status().

6.27.4.6 void set_knob_function (unsigned char *function*)

Set the rotary knob function.

Parameters:

function Which type of action should occur when the knob is turned

Definition at line 178 of file main.c.

References struct_runtime_settings::band_change_mode, BAND_CHANGE_MODE_-MANUAL, struct_status::knob_function, KNOB_FUNCTION_AUTO, KNOB_-FUNCTION_NONE, KNOB_FUNCTION_SELECT_BAND, runtime_settings, and status.

Referenced by band_ctrl_change_band(), event_poll_buttons(), event_rotate_button_-pressed(), event_rxant_button_pressed(), event_sub_button_pressed(), main(), and menu_-action().

6.28 general_io/main.h File Reference

Defines

- #define BUS_RX_QUEUE_SIZE 10

  The size of the RX queue in buffers.

- #define BUS_TX_QUEUE_SIZE 10

  The size of the TX queue in buffers.

6.28.1 Detailed Description

Definition in file main.h.

6.29 motherboard/main.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
```

Classes

- struct PS2_STRUCT

  Struct of the PS/2 interface status.

Defines

- #define INT_COMM_REDIRECT_DATA 0x10

  Internal communication command to redirect data.

6.29.1 Detailed Description

Definition in file main.h.

6.30 event_queue.c File Reference

Event queue.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "event_queue.h"
```

Functions

- void event_queue_init (void)

  Initialize the event queue.

- char event_queue_add (EVENT_MESSAGE event)

  Insert a message into the event queue. It will end up on a position based on it's time_target.

- void event_queue_wrap (unsigned int remove_val)

  Removes a certain amount of numbers from the time_target.

- EVENT_MESSAGE event_queue_get ()

  Retrieve the first message from the event queue.

- unsigned char event_in_queue (void)

  Checks if there is any event in the queue.

- void event_queue_drop (void)

  Drops the first message in the queue.

- unsigned char event_queue_count (void)

  Retrieve the number of items in the event queue.

- void event_queue_dropall (void)

  Erase all content in the event queue.

- int event_queue_drop_id (unsigned char id)

  Drops all messages in the queue with a certain ID.

- unsigned char event_queue_check_id (unsigned char id)

  Check if a specific ID exist in the event queue.

6.30.1 Detailed Description

Event queue.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "event_queue.c"
```

Definition in file event_queue.c.

6.30.2 Function Documentation

6.30.2.1 unsigned char event_in_queue (void)

Checks if there is any event in the queue.

Returns:

1 if there is an event in the queue and 0 otherwise

Definition at line 85 of file event_queue.c.

References event_list.

Referenced by event_add_message(), event_run(), and ISR().

6.30.2.2 char event_queue_add (EVENT_MESSAGE *event*)

Insert a message into the event queue. It will end up on a position based on it's time_target.

Parameters:

event - The event that should be inserted into the queue

Returns:

The position the event was inserted into, -1 means queue was full

Definition at line 43 of file event_queue.c.

References event_list, EVENT_LIST_SIZE, and EVENT_MESSAGE::time_target.

Referenced by event_add_message().

6.30.2.3 unsigned char event_queue_check_id (unsigned char *id*)

Check if a specific ID exist in the event queue.

Parameters:

id The id we which to check for

Returns:

1 if it exist, 0 if it doesn't

Definition at line 155 of file event_queue.c.

References event_list, and EVENT_LIST_SIZE.

6.30.2.4 unsigned char event_queue_count (void)

Retrieve the number of items in the event queue.

Returns:

Numbers of items in the queue

Definition at line 104 of file event_queue.c.

References event_list, and EVENT_LIST_SIZE.

6.30.2.5 int event_queue_drop_id (unsigned char *id*)

Drops all messages in the queue with a certain ID.

Returns:

the number of events that were dropped

Definition at line 126 of file event_queue.c.

References event_list, EVENT_LIST_SIZE, EVENT_MESSAGE::func, EVENT_-MESSAGE::id, and EVENT_MESSAGE::time_target.

Referenced by sequencer_computer_rts_deactivated(), and sequencer_footsw_released().

6.30.2.6 EVENT_MESSAGE event_queue_get (void)

Retrieve the first message from the event queue.

Returns:

The first message in the queue

Definition at line 78 of file event_queue.c.

References event_list.

Referenced by event_run(), and ISR().

6.30.2.7 void event_queue_wrap (unsigned int *remove_val*)

Removes a certain amount of numbers from the time_target.

Parameters:

remove_val The number we want to remove from all time targets

Definition at line 69 of file event_queue.c.

References event_list, EVENT_LIST_SIZE, and EVENT_MESSAGE::time_target.

Referenced by ISR().

6.31 event_queue.h File Reference

Event queue.

Classes

- struct EVENT_MESSAGE

  Event message used for timing of events.

Defines

- #define EVENT_LIST_SIZE 10

  The size of the event list.

Functions

- void event_queue_init (void)

  Initialize the event queue.

- char event_queue_add (EVENT_MESSAGE event)

  Insert a message into the event queue. It will end up on a position based on it's time_target.

- EVENT_MESSAGE event_queue_get (void)

  Retrieve the first message from the event queue.

- void event_queue_drop (void)

  Drops the first message in the queue.

- unsigned char event_queue_count (void)

  Retrieve the number of items in the event queue.

- void event_queue_dropall (void)

  Erase all content in the event queue.

- unsigned char event_in_queue (void)

  Checks if there is any event in the queue.

- void event_queue_wrap (unsigned int remove_val)

  Removes a certain amount of numbers from the time_target.

- int event_queue_drop_id (unsigned char id)

  Drops all messages in the queue with a certain ID.

- unsigned char event_queue_check_id (unsigned char id)

  Check if a specific ID exist in the event queue.

Variables

- EVENT_MESSAGE event_list [EVENT_LIST_SIZE]

  Event list with size EVENT_LIST_SIZE.

6.31.1 Detailed Description

Event queue.

Definition in file event_queue.h.

6.31.2 Function Documentation

6.31.2.1 unsigned char event_in_queue (void)

Checks if there is any event in the queue.

Returns:

1 if there is an event in the queue and 0 otherwise

Definition at line 85 of file event_queue.c.

References event_list.

Referenced by event_add_message(), event_run(), and ISR().

6.31.2.2 char event_queue_add (EVENT_MESSAGE *event*)

Insert a message into the event queue. It will end up on a position based on it's time_target.

Parameters:

event - The event that should be inserted into the queue

Returns:

The position the event was inserted into, -1 means queue was full

Definition at line 43 of file event_queue.c.

References event_list, EVENT_LIST_SIZE, and EVENT_MESSAGE::time_target.

Referenced by event_add_message().

6.31.2.3 unsigned char event_queue_check_id (unsigned char *id*)

Check if a specific ID exist in the event queue.

Parameters:

id The id we which to check for

Returns:

1 if it exist, 0 if it doesn't

Definition at line 155 of file event_queue.c.

References event_list, and EVENT_LIST_SIZE.

6.31.2.4 unsigned char event_queue_count (void)

Retrieve the number of items in the event queue.

Returns:

Numbers of items in the queue

Definition at line 104 of file event_queue.c.

References event_list, and EVENT_LIST_SIZE.

6.31.2.5 int event_queue_drop_id (unsigned char *id*)

Drops all messages in the queue with a certain ID.

Returns:

the number of events that were dropped

Definition at line 126 of file event_queue.c.

References event_list, EVENT_LIST_SIZE, EVENT_MESSAGE::func, EVENT_-MESSAGE::id, and EVENT_MESSAGE::time_target.

Referenced by sequencer_computer_rts_deactivated(), and sequencer_footsw_released().

6.31.2.6 EVENT_MESSAGE event_queue_get (void)

Retrieve the first message from the event queue.

Returns:

The first message in the queue

Definition at line 78 of file event_queue.c.

References event_list.

Referenced by event_run(), and ISR().

6.31.2.7 void event_queue_wrap (unsigned int *remove_val*)

Removes a certain amount of numbers from the time_target.

Parameters:

remove_val The number we want to remove from all time targets

Definition at line 69 of file event_queue.c.

References event_list, EVENT_LIST_SIZE, and EVENT_MESSAGE::time_target.

Referenced by ISR().

6.32 front_panel/antenna_ctrl.c File Reference

Antenna control functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include "antenna_ctrl.h"
#include "main.h"
#include "eeprom.h"
#include "led_control.h"
#include "band_ctrl.h"
#include "event_handler.h"
#include "../global.h"
#include "../wmv_bus/bus.h"
#include "../wmv_bus/bus_rx_queue.h"
#include "../wmv_bus/bus_tx_queue.h"
#include "../wmv_bus/bus_commands.h"
```

Functions

- unsigned char **antenna_ctrl_get_comb_value** (unsigned char antenna_comb)
- unsigned char antenna_ctrl_comb_allowed (unsigned char antenna_comb)

  Retrieve if a certain antenna combination is allowed.
- void antenna_ctrl_deactivate_outputs (unsigned char *addresses, unsigned char length, unsigned char cmd)

  This function will go through a parameter with addresses and send a command to it.
- unsigned char antenna_ctrl_antenna_selected (void)

  This function returns the selected antenna combination.
- void antenna_ctrl_send_ant_data_to_bus (void)

  Send the output string for the current antenna to the bus.
- void antenna_ctrl_send_rx_ant_data_to_bus (unsigned char antenna_index)

  Send the output string for the rx antenna to the bus.
- void antenna_ctrl_send_rx_ant_band_data_to_bus (char index)

  Send the output string for the rx antenna to the bus.
- void **antenna_ctrl_rotate** (unsigned char ant_index, unsigned int heading)
- void antenna_ctrl_change_rx_ant (unsigned char ant_index)

Function used to change an rx antenna.

- unsigned char antenna_ctrl_get_rotatable (void)

  Get which antennas can be rotated.

- void antenna_ctrl_deactivate_all (void)

  Function which will deactivate all activated antenna ctrl outputs, using type BUS_CMD_-DRIVER_DEACTIVATE_ALL_ANT_OUTPUTS.

- void antenna_ctrl_deactivate_all_rx_band (void)

  Function which will deactivate all activated rx antenna ctrl band outputs, using type BUS_-CMD_DRIVER_DEACTIVATE_ALL_RX_BAND_OUTPUTS.

- void antenna_ctrl_set_antenna_text (char *str, unsigned char index)

  Set the antenna text.

- char * antenna_ctrl_get_antenna_text (unsigned char index)
- unsigned char antenna_ctrl_get_antenna_text_length (unsigned char index)

  Get the antenna text length.

- void antenna_ctrl_set_output_comb (unsigned char *data, unsigned char index, unsigned char length)

  Set the output combination string.

- unsigned char * antenna_ctrl_get_output_comb (unsigned char index)

  Retrieve the output combination string.

- unsigned char antenna_ctrl_get_output_comb_length (unsigned char index)

  Retrieve the length of the output combination string.

- void antenna_ctrl_set_direction (unsigned int dir, unsigned char index)

  Set the direction of a specific antenna.

- unsigned int antenna_ctrl_get_direction (unsigned char index)

  Get the direction of a specific antenna.

- unsigned char antenna_ctrl_get_rotator_addr (unsigned char ant_index)

  Get the address of the rotator at a certain antenna index.

- void antenna_ctrl_set_flags (unsigned char flags, unsigned char index)

  Set the antenna flags.

- unsigned char antenna_ctrl_get_flags (unsigned char index)

  Get the antenna flags.

- void antenna_ctrl_set_comb_allowed (unsigned int comb)

  Set the value of combination allowed.

- void antenna_ctrl_set_rotator_flags (unsigned char ant_index, unsigned char flags)

  Set the flags of the rotator, see antenna_ctrl.h for defines.

- unsigned char antenna_ctrl_get_rotator_flags (unsigned char ant_index)

    Get the flags of the rotator, see antenna_ctrl.h for defines.

- unsigned int antenna_ctrl_get_comb_allowed (void)

    Get the value of combination allowed.

- void antenna_ctrl_set_antenna_data (struct_antenna *data)

    Set the antenna data.

- void antenna_ctrl_set_rx_antenna_data (struct_rx_antennas *data)

    Set the antenna rx data.

- unsigned char antenna_ctrl_get_rx_antenna_count (void)

    Retrieve the number of rx antennas.

- char * antenna_ctrl_get_rx_antenna_name (unsigned char ant_index)

    Retrieve the rx antenna name.

- char * antenna_ctrl_get_rx_antenna_output_str (unsigned char ant_index)

    Retrieve the rx antenna output str.

- void antenna_ctrl_select_default_ant (void)

    Function which will select the default antenna for this band if it is configured.

- void antenna_ctrl_ant_read_eeprom (unsigned char band_index)

    Read the eeprom for the antenna settings.

- void antenna_ctrl_rx_ant_read_eeprom (void)

    Read the eeprom for the rx antenna settings.

- unsigned int antenna_ctrl_get_start_heading (unsigned char ant_index)

    Function returns the start heading for a certain antenna.

- unsigned int antenna_ctrl_get_max_rotation (unsigned char ant_index)

    Function returns the maximal number of degrees we can rotate an antenna.

- unsigned char antenna_ctrl_get_sub_menu_type (unsigned char ant_index)

    Get which kind of sub meny type an antenna has got.

Variables

- struct_antenna current_antennas

    Contains the current antenna information.

- struct_rx_antennas rx_antennas

    Contains the rx antenna information.

- unsigned int main_flags

Different flags, description is found in main.h.

- unsigned char current_activated_ant_outputs [ANTENNA_OUTPUT_COMB_SIZE]

  Array which we store the current devices which we have activated antenna outputs on.

- unsigned char current_activated_ant_outputs_length = 0

  How many devices we have activated antenna outputs on.

- unsigned char current_activated_rx_ant_outputs [RX_ANTENNA_OUTPUT_STR_-LENGTH]

  Array which we store the current devices which we have rx antenna activated outputs on.

- unsigned char current_activated_rx_ant_outputs_length = 0

  How many devices we have activated rx antenna outputs on.

- unsigned char current_band_activated_outputs_rx [RX_ANTENNA_BAND_-OUTPUT_STR_LENGTH]

  Array which we store the current devices which we have activated rx antenna band outputs on.

- unsigned char current_band_activated_outputs_rx_length = 0

  How many devices we have activated rx antenna band outputs on.

6.32.1 Detailed Description

Antenna control functions.

Antenna control functions

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/antenna_ctrl.c"
```

Definition in file antenna_ctrl.c.

6.32.2 Function Documentation

6.32.2.1 void antenna_ctrl_ant_read_eeprom (unsigned char *band_index*)

Read the eeprom for the antenna settings.

Parameters:

band_index The band index

Definition at line 493 of file antenna_ctrl.c.

References eeprom_get_antenna_data().

Referenced by band_ctrl_load_band().

6.32.2.2 unsigned char antenna_ctrl_antenna_selected (void)

This function returns the selected antenna combination.

Returns:

The selected antenna combination, for example 1 means antenna 1, 3 means antenna 1 and 2 (binary representation)

Definition at line 134 of file antenna_ctrl.c.

References struct_status::selected_ant, and status.

6.32.2.3 void antenna_ctrl_change_rx_ant (unsigned char *ant_index*)

Function used to change an rx antenna.

Parameters:

ant_index Which RX antenna we wish to chose. If ant_index = 0 the rx antenna outputs are disabled

Definition at line 294 of file antenna_ctrl.c.

References antenna_ctrl_deactivate_outputs(), antenna_ctrl_send_rx_ant_data_to_bus(), BUS_CMD_DRIVER_DEACTIVATE_ALL_RX_ANTENNA_OUTPUTS, current_activated_rx_ant_outputs, and current_activated_rx_ant_outputs_length.

Referenced by band_ctrl_change_band(), event_rxant_button_pressed(), event_set_rx_antenna(), and main().

6.32.2.4 unsigned char antenna_ctrl_comb_allowed (unsigned char *antenna_comb*)

Retrieve if a certain antenna combination is allowed.

Parameters:

antenna_comb The antenna configuration you wish to check

Returns:

1 if the combination is allowed, 0 if it is not allowed

Definition at line 107 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_allowed.

Referenced by event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_button3_pressed(), and event_tx_button4_pressed().

6.32.2.5 void antenna_ctrl_deactivate_outputs (unsigned char ∗ *addresses*, unsigned char *length*, unsigned char *cmd*)

This function will go through a parameter with addresses and send a command to it.

Parameters:

addresses The list of addresses to send CMD to

length The length of the address list

cmd The command we wish to send to the boards in the address list

Definition at line 121 of file antenna_ctrl.c.

References bus_add_tx_message(), bus_get_address(), BUS_MESSAGE_FLAGS_NEED_-ACK, and internal_comm_add_tx_message().

Referenced by antenna_ctrl_change_rx_ant(), antenna_ctrl_deactivate_all(), antenna_ctrl_-deactivate_all_rx_band(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_send_-rx_ant_band_data_to_bus(), antenna_ctrl_send_rx_ant_data_to_bus(), band_ctrl_-deactivate_all(), sub_menu_deactivate_all(), and sub_menu_send_data_to_bus().

6.32.2.6 char∗ antenna_ctrl_get_antenna_text (unsigned char *index*)

Get the antenna text

Parameters:

index The index of the antenna

Returns:

A pointer to the string

Definition at line 342 of file antenna_ctrl.c.

References struct_antenna::antenna_text.

Referenced by display_antennas(), and display_show_sub_menu().

6.32.2.7 unsigned char antenna_ctrl_get_antenna_text_length (unsigned char *index*)

Get the antenna text length.

Parameters:

index The index of the antenna

Returns:

the length of the text

Definition at line 349 of file antenna_ctrl.c.

References struct_antenna::antenna_text_length.

Referenced by display_antennas(), and display_invert_antenna().

6.32.2.8 unsigned int antenna_ctrl_get_comb_allowed (void)

Get the value of combination allowed.

Returns:

The combination allowed value

Definition at line 437 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_allowed.

6.32.2.9 unsigned int antenna_ctrl_get_direction (unsigned char *index*)

Get the direction of a specific antenna.

Parameters:

index The index of the antenna

Returns:

The direction of the antenna

Definition at line 388 of file antenna_ctrl.c.

References struct_antenna::antenna_direction.

Referenced by display_rotator_directions(), event_rotate_button_pressed(), event_tx_-button1_pressed(), event_tx_button2_pressed(), event_tx_button3_pressed(), and event_-tx_button4_pressed().

6.32.2.10 unsigned char antenna_ctrl_get_flags (unsigned char *index*)

Get the antenna flags.

Parameters:

index The index of which antenna you wish to get the flag content from

Returns:

The flags

Definition at line 411 of file antenna_ctrl.c.

References struct_antenna::antenna_flag.

Referenced by display_antennas(), display_invert_antenna(), display_rotator_directions(), event_rotate_button_pressed(), event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_button3_pressed(), event_tx_button4_pressed(), and main_update_ptt_status().

6.32.2.11 unsigned int antenna_ctrl_get_max_rotation (unsigned char *ant_index*)

Function returns the maximal number of degrees we can rotate an antenna.

Parameters:

ant_index The antenna index we wish to retrieve the information from

Returns:

The number of degrees the antenna can be rotated

Definition at line 512 of file antenna_ctrl.c.

References struct_antenna::rotator_max_rotation.

Referenced by event_pulse_sensor_down(), and event_pulse_sensor_up().

6.32.2.12 unsigned char∗ antenna_ctrl_get_output_comb (unsigned char *index*)

Retrieve the output combination string.

Parameters:

index Which of the bands you wish to get the output string for

Returns:

pointer to the string beginning

Definition at line 367 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_output_str.

6.32.2.13 unsigned char antenna_ctrl_get_output_comb_length (unsigned char *index*)

Retrieve the length of the output combination string.

Parameters:

index Which of the combinations you wish to retrieve the length of

Returns:

The length of the output string

Definition at line 374 of file antenna_ctrl.c.

References struct_antenna::antenna_output_length.

6.32.2.14 unsigned char antenna_ctrl_get_rotatable (void)

Get which antennas can be rotated.

Returns:

Which antennas can be rotated, in binary form starting with ant 0 from byte 0

Definition at line 306 of file antenna_ctrl.c.

References struct_antenna::antenna_flag, and ANTENNA_ROTATOR_FLAG.

Referenced by ISR().

6.32.2.15 unsigned char antenna_ctrl_get_rotator_addr (unsigned char *ant_index*)

Get the address of the rotator at a certain antenna index.

Parameters:

ant_index The index of the antenna

Returns:

The address of the rotator

Definition at line 395 of file antenna_ctrl.c.

References struct_antenna::rotator_addr.

6.32.2.16 unsigned char antenna_ctrl_get_rotator_flags (unsigned char *ant_index*)

Get the flags of the rotator, see antenna_ctrl.h for defines.

Returns:

The rotator flags of the antenna

Definition at line 430 of file antenna_ctrl.c.

References struct_antenna::rotator_flags.

Referenced by ISR().

6.32.2.17 unsigned char antenna_ctrl_get_rx_antenna_count (void)

Retrieve the number of rx antennas.

Returns:

The number of rx antenna count

Definition at line 455 of file antenna_ctrl.c.

References struct_rx_antennas::name.

Referenced by display_show_rx_ant(), event_pulse_sensor_down(), event_pulse_sensor_up(), and event_rxant_button_pressed().

6.32.2.18 char∗ antenna_ctrl_get_rx_antenna_name (unsigned char *ant_index*)

Retrieve the rx antenna name.

Parameters:

ant_index The index of the antenna

Returns:

The name of the RX antenna

Definition at line 468 of file antenna_ctrl.c.

References struct_rx_antennas::name.

Referenced by display_show_rx_ant().

6.32.2.19 char* antenna_ctrl_get_rx_antenna_output_str (unsigned char *ant_index*)

Retrieve the rx antenna output str.

Parameters:

ant_index The index of the antenna

Returns:

The output str of the rx antenna sent in

Definition at line 475 of file antenna_ctrl.c.

References struct_rx_antennas::output_str.

6.32.2.20 unsigned int antenna_ctrl_get_start_heading (unsigned char *ant_index*)

Function returns the start heading for a certain antenna.

Parameters:

ant_index The index of the antenna we wish to retrieve the heading from

Returns:

The start heading of this antenna

Definition at line 505 of file antenna_ctrl.c.

References struct_antenna::rotator_min_heading.

Referenced by event_pulse_sensor_down(), and event_pulse_sensor_up().

6.32.2.21 unsigned char antenna_ctrl_get_sub_menu_type (unsigned char *ant_index*)

Get which kind of sub meny type an antenna has got.

Parameters:

ant_index Which antenna index we wish to show the sub menu type for

Returns:

The sub meny type

Definition at line 519 of file antenna_ctrl.c.

References struct_antenna::sub_menu_type.

Referenced by display_rotator_directions(), event_sub_button_pressed(), sub_menu_-activate_all(), sub_menu_get_count(), sub_menu_get_text(), sub_menu_get_type(), and sub_menu_load().

6.32.2.22 void antenna_ctrl_send_rx_ant_band_data_to_bus (char *index*)

Send the output string for the rx antenna to the bus.

Parameters:

index The index of the antenna you wish to send the string of

Definition at line 231 of file antenna_ctrl.c.

References antenna_ctrl_deactivate_outputs(), struct_rx_antennas::band_output_length, struct_rx_antennas::band_output_str, bus_add_tx_message(), BUS_CMD_DRIVER_-ACTIVATE_RX_BAND_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_ALL_RX_-BAND_OUTPUTS, bus_get_address(), BUS_MESSAGE_FLAGS_NEED_ACK, current_-band_activated_outputs_rx, current_band_activated_outputs_rx_length, internal_comm_-add_tx_message(), and OUTPUT_ADDR_DELIMITER.

Referenced by band_ctrl_change_band().

6.32.2.23 void antenna_ctrl_send_rx_ant_data_to_bus (unsigned char *antenna_index*)

Send the output string for the rx antenna to the bus.

Parameters:

antenna_index The index of the antenna you wish to send the string of

Definition at line 189 of file antenna_ctrl.c.

References antenna_ctrl_deactivate_outputs(), bus_add_tx_message(), BUS_CMD_-DRIVER_ACTIVATE_RX_ANT_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_ALL_-RX_ANTENNA_OUTPUTS, bus_get_address(), BUS_MESSAGE_FLAGS_NEED_ACK, current_activated_rx_ant_outputs, current_activated_rx_ant_outputs_length, internal_-comm_add_tx_message(), OUTPUT_ADDR_DELIMITER, struct_rx_antennas::output_-length, and struct_rx_antennas::output_str.

Referenced by antenna_ctrl_change_rx_ant().

6.32.2.24 void antenna_ctrl_set_antenna_data (struct_antenna ∗ *data*)

Set the antenna data.

Parameters:

data The data we wish to use as antenna data

Definition at line 443 of file antenna_ctrl.c.

6.32.2.25 void antenna_ctrl_set_antenna_text (char * *str*, unsigned char *index*)

Set the antenna text.

Parameters:

str Which data should be saved

index The index of the antenna

Definition at line 335 of file antenna_ctrl.c.

References struct_antenna::antenna_text.

6.32.2.26 void antenna_ctrl_set_comb_allowed (unsigned int *comb*)

Set the value of combination allowed.

Parameters:

comb The combination that is allowed

Definition at line 417 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_allowed.

6.32.2.27 void antenna_ctrl_set_direction (unsigned int *dir*, unsigned char *index*)

Set the direction of a specific antenna.

Parameters:

dir The direction of the antenna

index The index of the antenna

Definition at line 381 of file antenna_ctrl.c.

References struct_antenna::antenna_direction.

6.32.2.28 void antenna_ctrl_set_flags (unsigned char *flags*, unsigned char *index*)

Set the antenna flags.

Parameters:

flags The flags you wish to be enabled for this antenna

index The index of the antenna which the flags should be set to

Definition at line 403 of file antenna_ctrl.c.

References struct_antenna::antenna_flag.

6.32.2.29 void antenna_ctrl_set_output_comb (unsigned char * *data*, unsigned char *index*, unsigned char *length*)

Set the output combination string.

Parameters:

data The string you wish to save

index The index of the output combination

length The length of the output string

Definition at line 358 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_output_str, and struct_antenna::antenna_output_-length.

6.32.2.30 void antenna_ctrl_set_rotator_flags (unsigned char *ant_index*, unsigned char *flags*)

Set the flags of the rotator, see antenna_ctrl.h for defines.

Parameters:

ant_index The antenna index

flags Flags from the rotator

Definition at line 424 of file antenna_ctrl.c.

References struct_antenna::rotator_flags.

6.32.2.31 void antenna_ctrl_set_rx_antenna_data (struct_rx_antennas * *data*)

Set the antenna rx data.

Parameters:

data The data we wish to use as rx antenna data

Definition at line 449 of file antenna_ctrl.c.

6.33 front_panel/antenna_ctrl.h File Reference

Antenna control functions.

`#include "main.h"`

Classes

- struct struct_rx_antennas

  Struct which contains information of the rx antennas.

- struct struct_antenna

  Structure of an antenna.

Defines

- #define FLAG_NO_ROTATION 1

  The rotator is currently standing still.

- #define FLAG_ROTATION_ALLOWED 2

  The rotator is allowed to be rotated.

- #define FLAG_ROTATION_CW 3

  The rotator is being rotated CW.

- #define FLAG_ROTATION_CCW 4

  The rotator is being rotated CCW.

Functions

- void antenna_ctrl_deactivate_all_rx_band (void)

  Function which will deactivate all activated rx antenna ctrl band outputs, using type BUS_-CMD_DRIVER_DEACTIVATE_ALL_RX_BAND_OUTPUTS.

- void antenna_ctrl_send_ant_data_to_bus (void)

  Send the output string for the current antenna to the bus.

- void antenna_ctrl_send_rx_ant_data_to_bus (unsigned char antenna_index)

  Send the output string for the rx antenna to the bus.

- void antenna_ctrl_send_rx_ant_band_data_to_bus (char index)

  Send the output string for the rx antenna to the bus.

- unsigned char antenna_ctrl_comb_allowed (unsigned char antenna_comb)

  Retrieve if a certain antenna combination is allowed.

- unsigned char **antenna_ctrl_get_comb_value** (unsigned char antenna_comb)

- void antenna_ctrl_set_antenna_text (char ∗str, unsigned char index)

  Set the antenna text.

- char ∗ antenna_ctrl_get_antenna_text (unsigned char index)
- unsigned char antenna_ctrl_get_antenna_text_length (unsigned char index)

  Get the antenna text length.

- void antenna_ctrl_set_output_comb (unsigned char ∗data, unsigned char index, unsigned char length)

  Set the output combination string.

- unsigned char ∗ antenna_ctrl_get_output_comb (unsigned char index)

  Retrieve the output combination string.

- unsigned char antenna_ctrl_get_output_comb_length (unsigned char index)

  Retrieve the length of the output combination string.

- void antenna_ctrl_set_direction (unsigned int dir, unsigned char index)

  Set the direction of a specific antenna.

- unsigned int antenna_ctrl_get_direction (unsigned char index)

  Get the direction of a specific antenna.

- void antenna_ctrl_set_flags (unsigned char flags, unsigned char index)

  Set the antenna flags.

- unsigned char antenna_ctrl_get_flags (unsigned char index)

  Get the antenna flags.

- void antenna_ctrl_set_comb_allowed (unsigned int comb)

  Set the value of combination allowed.

- unsigned int antenna_ctrl_get_comb_allowed (void)

  Get the value of combination allowed.

- void antenna_ctrl_change_rx_ant (unsigned char ant_index)

  Function used to change an rx antenna.

- void antenna_ctrl_set_antenna_data (struct_antenna ∗data)

  Set the antenna data.

- void antenna_ctrl_set_rx_antenna_data (struct_rx_antennas ∗data)

  Set the antenna rx data.

- unsigned char antenna_ctrl_get_rx_antenna_count (void)

  Retrieve the number of rx antennas.

- char ∗ antenna_ctrl_get_rx_antenna_name (unsigned char ant_index)

  Retrieve the rx antenna name.

- char ∗ antenna_ctrl_get_rx_antenna_output_str (unsigned char ant_index)

  Retrieve the rx antenna output str.

- void antenna_ctrl_ant_read_eeprom (unsigned char band_index)

  Read the eeprom for the antenna settings.

- void antenna_ctrl_rx_ant_read_eeprom (void)

  Read the eeprom for the rx antenna settings.

- void antenna_ctrl_deactivate_outputs (unsigned char ∗addresses, unsigned char length, unsigned char cmd)

  This function will go through a parameter with addresses and send a command to it.

- unsigned char antenna_ctrl_get_sub_menu_type (unsigned char ant_index)

  Get which kind of sub meny type an antenna has got.

- void antenna_ctrl_deactivate_all (void)

  Function which will deactivate all activated antenna ctrl outputs, using type BUS_CMD_-DRIVER_DEACTIVATE_ALL_ANT_OUTPUTS.

- void **antenna_ctrl_rotate** (unsigned char ant_index, unsigned int heading)
- unsigned char antenna_ctrl_antenna_selected (void)

  This function returns the selected antenna combination.

- unsigned char antenna_ctrl_get_rotatable (void)

  Get which antennas can be rotated.

- unsigned int antenna_ctrl_get_start_heading (unsigned char ant_index)

  Function returns the start heading for a certain antenna.

- unsigned int antenna_ctrl_get_max_rotation (unsigned char ant_index)

  Function returns the maximal number of degrees we can rotate an antenna.

- unsigned char antenna_ctrl_get_rotator_addr (unsigned char ant_index)

  Get the address of the rotator at a certain antenna index.

- void antenna_ctrl_set_rotator_flags (unsigned char ant_index, unsigned char flags)

  Set the flags of the rotator, see antenna_ctrl.h for defines.

- unsigned char antenna_ctrl_get_rotator_flags (unsigned char ant_index)

  Get the flags of the rotator, see antenna_ctrl.h for defines.

- void antenna_ctrl_select_default_ant (void)

  Function which will select the default antenna for this band if it is configured.

6.33.1 Detailed Description

Antenna control functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/antenna_ctrl.h"
```

Definition in file antenna_ctrl.h.

6.33.2 Function Documentation

6.33.2.1 void antenna_ctrl_ant_read_eeprom (unsigned char *band_index*)

Read the eeprom for the antenna settings.

Parameters:

band_index The band index

Definition at line 493 of file antenna_ctrl.c.

References eeprom_get_antenna_data().

Referenced by band_ctrl_load_band().

6.33.2.2 unsigned char antenna_ctrl_antenna_selected (void)

This function returns the selected antenna combination.

Returns:

The selected antenna combination, for example 1 means antenna 1, 3 means antenna 1 and 2 (binary representation)

Definition at line 134 of file antenna_ctrl.c.

References struct_status::selected_ant, and status.

6.33.2.3 void antenna_ctrl_change_rx_ant (unsigned char *ant_index*)

Function used to change an rx antenna.

Parameters:

ant_index Which RX antenna we wish to chose. If ant_index = 0 the rx antenna outputs are disabled

Definition at line 294 of file antenna_ctrl.c.

References antenna_ctrl_deactivate_outputs(), antenna_ctrl_send_rx_ant_data_to_-bus(), BUS_CMD_DRIVER_DEACTIVATE_ALL_RX_ANTENNA_OUTPUTS, current_-activated_rx_ant_outputs, and current_activated_rx_ant_outputs_length.

Referenced by band_ctrl_change_band(), event_rxant_button_pressed(), event_set_rx_-antenna(), and main().

6.33.2.4 unsigned char antenna_ctrl_comb_allowed (unsigned char *antenna_comb*)

Retrieve if a certain antenna combination is allowed.

Parameters:

antenna_comb The antenna configuration you wish to check

Returns:

1 if the combination is allowed, 0 if it is not allowed

Definition at line 107 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_allowed.

Referenced by event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_-button3_pressed(), and event_tx_button4_pressed().

6.33.2.5 void antenna_ctrl_deactivate_outputs (unsigned char * *addresses*, unsigned char *length*, unsigned char *cmd*)

This function will go through a parameter with addresses and send a command to it.

Parameters:

addresses The list of addresses to send CMD to

length The length of the address list

cmd The command we wish to send to the boards in the address list

Definition at line 121 of file antenna_ctrl.c.

References bus_add_tx_message(), bus_get_address(), BUS_MESSAGE_FLAGS_NEED_-ACK, and internal_comm_add_tx_message().

Referenced by antenna_ctrl_change_rx_ant(), antenna_ctrl_deactivate_all(), antenna_ctrl_-deactivate_all_rx_band(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_send_-rx_ant_band_data_to_bus(), antenna_ctrl_send_rx_ant_data_to_bus(), band_ctrl_-deactivate_all(), sub_menu_deactivate_all(), and sub_menu_send_data_to_bus().

6.33.2.6 char* antenna_ctrl_get_antenna_text (unsigned char *index*)

Get the antenna text

Parameters:

index The index of the antenna

Returns:

A pointer to the string

Definition at line 342 of file antenna_ctrl.c.

References struct_antenna::antenna_text.

Referenced by display_antennas(), and display_show_sub_menu().

6.33.2.7 unsigned char antenna_ctrl_get_antenna_text_length (unsigned char *index*)

Get the antenna text length.

Parameters:

index The index of the antenna

Returns:

the length of the text

Definition at line 349 of file antenna_ctrl.c.

References struct_antenna::antenna_text_length.

Referenced by display_antennas(), and display_invert_antenna().

6.33.2.8 unsigned int antenna_ctrl_get_comb_allowed (void)

Get the value of combination allowed.

Returns:

The combination allowed value

Definition at line 437 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_allowed.

6.33.2.9 unsigned int antenna_ctrl_get_direction (unsigned char *index*)

Get the direction of a specific antenna.

Parameters:

index The index of the antenna

Returns:

The direction of the antenna

Definition at line 388 of file antenna_ctrl.c.

References struct_antenna::antenna_direction.

Referenced by display_rotator_directions(), event_rotate_button_pressed(), event_tx_-button1_pressed(), event_tx_button2_pressed(), event_tx_button3_pressed(), and event_-tx_button4_pressed().

6.33.2.10 unsigned char antenna_ctrl_get_flags (unsigned char *index*)

Get the antenna flags.

Parameters:

index The index of which antenna you wish to get the flag content from

Returns:

The flags

Definition at line 411 of file antenna_ctrl.c.

References struct_antenna::antenna_flag.

Referenced by display_antennas(), display_invert_antenna(), display_rotator_directions(), event_rotate_button_pressed(), event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_button3_pressed(), event_tx_button4_pressed(), and main_update_ptt_status().

6.33.2.11 unsigned int antenna_ctrl_get_max_rotation (unsigned char *ant_index*)

Function returns the maximal number of degrees we can rotate an antenna.

Parameters:

ant_index The antenna index we wish to retrieve the information from

Returns:

The number of degrees the antenna can be rotated

Definition at line 512 of file antenna_ctrl.c.

References struct_antenna::rotator_max_rotation.

Referenced by event_pulse_sensor_down(), and event_pulse_sensor_up().

6.33.2.12 unsigned char∗ antenna_ctrl_get_output_comb (unsigned char *index*)

Retrieve the output combination string.

Parameters:

index Which of the bands you wish to get the output string for

Returns:

pointer to the string beginning

Definition at line 367 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_output_str.

6.33.2.13 unsigned char antenna_ctrl_get_output_comb_length (unsigned char *index*)

Retrieve the length of the output combination string.

Parameters:

index Which of the combinations you wish to retrieve the length of

Returns:

The length of the output string

Definition at line 374 of file antenna_ctrl.c.

References struct_antenna::antenna_output_length.

6.33.2.14 unsigned char antenna_ctrl_get_rotatable (void)

Get which antennas can be rotated.

Returns:

Which antennas can be rotated, in binary form starting with ant 0 from byte 0

Definition at line 306 of file antenna_ctrl.c.

References struct_antenna::antenna_flag, and ANTENNA_ROTATOR_FLAG.

Referenced by ISR().

6.33.2.15 unsigned char antenna_ctrl_get_rotator_addr (unsigned char *ant_index*)

Get the address of the rotator at a certain antenna index.

Parameters:

ant_index The index of the antenna

Returns:

The address of the rotator

Definition at line 395 of file antenna_ctrl.c.

References struct_antenna::rotator_addr.

6.33.2.16 unsigned char antenna_ctrl_get_rotator_flags (unsigned char *ant_index*)

Get the flags of the rotator, see antenna_ctrl.h for defines.

Returns:

The rotator flags of the antenna

Definition at line 430 of file antenna_ctrl.c.

References struct_antenna::rotator_flags.

Referenced by ISR().

6.33.2.17 unsigned char antenna_ctrl_get_rx_antenna_count (void)

Retrieve the number of rx antennas.

Returns:

The number of rx antenna count

Definition at line 455 of file antenna_ctrl.c.

References struct_rx_antennas::name.

Referenced by display_show_rx_ant(), event_pulse_sensor_down(), event_pulse_sensor_up(), and event_rxant_button_pressed().

6.33.2.18 char∗ antenna_ctrl_get_rx_antenna_name (unsigned char *ant_index*)

Retrieve the rx antenna name.

Parameters:

ant_index The index of the antenna

Returns:

The name of the RX antenna

Definition at line 468 of file antenna_ctrl.c.

References struct_rx_antennas::name.

Referenced by display_show_rx_ant().

6.33.2.19 char∗ antenna_ctrl_get_rx_antenna_output_str (unsigned char *ant_index*)

Retrieve the rx antenna output str.

Parameters:

ant_index The index of the antenna

Returns:

The output str of the rx antenna sent in

Definition at line 475 of file antenna_ctrl.c.

References struct_rx_antennas::output_str.

6.33.2.20 unsigned int antenna_ctrl_get_start_heading (unsigned char *ant_index*)

Function returns the start heading for a certain antenna.

Parameters:

ant_index The index of the antenna we wish to retrieve the heading from

Returns:

The start heading of this antenna

Definition at line 505 of file antenna_ctrl.c.

References struct_antenna::rotator_min_heading.

Referenced by event_pulse_sensor_down(), and event_pulse_sensor_up().

6.33.2.21 unsigned char antenna_ctrl_get_sub_menu_type (unsigned char *ant_index*)

Get which kind of sub meny type an antenna has got.

Parameters:

ant_index Which antenna index we wish to show the sub menu type for

Returns:

The sub meny type

Definition at line 519 of file antenna_ctrl.c.

References struct_antenna::sub_menu_type.

Referenced by display_rotator_directions(), event_sub_button_pressed(), sub_menu_-activate_all(), sub_menu_get_count(), sub_menu_get_text(), sub_menu_get_type(), and sub_menu_load().

6.33.2.22 void antenna_ctrl_send_rx_ant_band_data_to_bus (char *index*)

Send the output string for the rx antenna to the bus.

Parameters:

index The index of the antenna you wish to send the string of

Definition at line 231 of file antenna_ctrl.c.

References antenna_ctrl_deactivate_outputs(), struct_rx_antennas::band_output_length, struct_rx_antennas::band_output_str, bus_add_tx_message(), BUS_CMD_DRIVER_-ACTIVATE_RX_BAND_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_ALL_RX_-BAND_OUTPUTS, bus_get_address(), BUS_MESSAGE_FLAGS_NEED_ACK, current_-band_activated_outputs_rx, current_band_activated_outputs_rx_length, internal_comm_-add_tx_message(), and OUTPUT_ADDR_DELIMITER.

Referenced by band_ctrl_change_band().

6.33.2.23 void antenna_ctrl_send_rx_ant_data_to_bus (unsigned char *antenna_index*)

Send the output string for the rx antenna to the bus.

Parameters:

antenna_index The index of the antenna you wish to send the string of

Definition at line 189 of file antenna_ctrl.c.

References antenna_ctrl_deactivate_outputs(), bus_add_tx_message(), BUS_CMD_-DRIVER_ACTIVATE_RX_ANT_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_ALL_-RX_ANTENNA_OUTPUTS, bus_get_address(), BUS_MESSAGE_FLAGS_NEED_ACK, current_activated_rx_ant_outputs, current_activated_rx_ant_outputs_length, internal_-comm_add_tx_message(), OUTPUT_ADDR_DELIMITER, struct_rx_antennas::output_-length, and struct_rx_antennas::output_str.

Referenced by antenna_ctrl_change_rx_ant().

6.33.2.24 void antenna_ctrl_set_antenna_data (struct_antenna ∗ *data*)

Set the antenna data.

Parameters:

data The data we wish to use as antenna data

Definition at line 443 of file antenna_ctrl.c.

6.33.2.25 void antenna_ctrl_set_antenna_text (char ∗ *str*, unsigned char *index*)

Set the antenna text.

Parameters:

str Which data should be saved

index The index of the antenna

Definition at line 335 of file antenna_ctrl.c.

References struct_antenna::antenna_text.

6.33.2.26 void antenna_ctrl_set_comb_allowed (unsigned int *comb*)

Set the value of combination allowed.

Parameters:

comb The combination that is allowed

Definition at line 417 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_allowed.

6.33.2.27 void antenna_ctrl_set_direction (unsigned int *dir*, unsigned char *index*)

Set the direction of a specific antenna.

Parameters:

dir The direction of the antenna
index The index of the antenna

Definition at line 381 of file antenna_ctrl.c.

References struct_antenna::antenna_direction.

6.33.2.28 void antenna_ctrl_set_flags (unsigned char *flags*, unsigned char *index*)

Set the antenna flags.

Parameters:

flags The flags you wish to be enabled for this antenna
index The index of the antenna which the flags should be set to

Definition at line 403 of file antenna_ctrl.c.

References struct_antenna::antenna_flag.

6.33.2.29 void antenna_ctrl_set_output_comb (unsigned char ∗ *data*, unsigned char *index*, unsigned char *length*)

Set the output combination string.

Parameters:

data The string you wish to save
index The index of the output combination
length The length of the output string

Definition at line 358 of file antenna_ctrl.c.

References struct_antenna::antenna_comb_output_str, and struct_antenna::antenna_output_-length.

6.33.2.30 void antenna_ctrl_set_rotator_flags (unsigned char *ant_index*, unsigned char *flags*)

Set the flags of the rotator, see antenna_ctrl.h for defines.

Parameters:

ant_index The antenna index
flags Flags from the rotator

Definition at line 424 of file antenna_ctrl.c.

References struct_antenna::rotator_flags.

6.33.2.31 void antenna_ctrl_set_rx_antenna_data (struct_rx_antennas * *data*)

Set the antenna rx data.

Parameters:

data The data we wish to use as rx antenna data

Definition at line 449 of file antenna_ctrl.c.

6.34 front_panel/band_ctrl.c File Reference

Band control functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "band_ctrl.h"
#include "main.h"
#include "eeprom.h"
#include "led_control.h"
#include "radio_interface.h"
#include "sub_menu.h"
#include "../global.h"
#include "../internal_comm.h"
#include "../wmv_bus/bus.h"
#include "../wmv_bus/bus_rx_queue.h"
#include "../wmv_bus/bus_tx_queue.h"
#include "../wmv_bus/bus_commands.h"
```

Functions

- void band_ctrl_send_band_data_to_bus (unsigned char band_portion)

  Send the output string for the current band to the bus.

- void band_ctrl_load_band (unsigned char band)

  Function will load a band from the EEPROM into the current_band struct.

- void band_ctrl_change_band_portion (unsigned char band_portion)

  Function will send out new band portion settings for the current selected band ∗.

- void band_ctrl_change_band (unsigned char band)

  Function used to change band.

- void band_ctrl_deactivate_all (void)

  Function which will deactiavate all band outptus, BUS_CMD_DRIVER_DEACTIVATE_-ALL_BAND_OUTPUTS.

- void band_ctrl_load_band_limits (void)

  Loads the band limits into the band limits struct.

- unsigned int band_ctrl_get_low_portion_low (unsigned char band)

  Retrieve the lower frequency limit of the low band limit.

- unsigned int band_ctrl_get_low_portion_high (unsigned char band)

  Retrieve the higher frequency limit of the low band limit.

- unsigned int band_ctrl_get_high_portion_low (unsigned char band)

  Retrieve the lower frequency limit of the high band limit.

- unsigned int band_ctrl_get_high_portion_high (unsigned char band)

  Retrieve the higher frequency limit of the high band limit.

- unsigned char ∗ band_ctrl_get_high_output_str (void)

  Retrieve the higher frequency output string, of max length BAND_OUTPUT_STR_SIZE.

- unsigned char ∗ band_ctrl_get_low_output_str (void)

  Retrieve the lower frequency output string, of max length BAND_OUTPUT_STR_SIZE.

- unsigned char band_ctrl_get_portion (void)

  Retrieve which band portion we are currently at.

Variables

- struct_band current_band

  Contains the current band information.

- unsigned char current_band_activated_outputs [BAND_OUTPUT_STR_SIZE]

  Array which we use to keep track of which devices we have been activating outputs on.

- unsigned char current_band_activated_outputs_length = 0

  The number of devices we have activated outputs on.

6.34.1 Detailed Description

Band control functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/band_ctrl.c"
```

Definition in file band_ctrl.c.

6.34.2 Function Documentation

6.34.2.1 void band_ctrl_change_band (unsigned char *band*)

Function used to change band.

Parameters:

band The band we wish to change to

Definition at line 145 of file band_ctrl.c.

References antenna_ctrl_change_rx_ant(), antenna_ctrl_deactivate_all(), antenna_ctrl_-deactivate_all_rx_band(), antenna_ctrl_select_default_ant(), antenna_ctrl_send_rx_ant_-band_data_to_bus(), band_ctrl_deactivate_all(), band_ctrl_load_band(), band_ctrl_-send_band_data_to_bus(), struct_status::current_band_portion, struct_status::current_-display, CURRENT_DISPLAY_ANTENNA_INFO, struct_status::current_display_level, CURRENT_DISPLAY_LOGO, CURRENT_DISPLAY_MENU_SYSTEM, CURRENT_-DISPLAY_SHUTDOWN_VIEW, DISPLAY_LEVEL_BAND, FUNC_STATUS_RXANT, struct_status::function_status, INHIBIT_NOT_OK_TO_SEND_RADIO_TX, KNOB_-FUNCTION_AUTO, led_set_band(), led_set_band_none(), led_set_rx_ant(), led_set_-rxant(), led_set_tx_ant(), LED_STATE_OFF, main_get_inhibit_state(), main_update_-display(), main_update_ptt_status(), struct_status::new_band, struct_status::selected_ant, struct_status::selected_band, struct_status::selected_rx_antenna, set_knob_function(), and status.

Referenced by event_internal_comm_parse_message(), event_poll_buttons(), and main().

6.34.2.2 void band_ctrl_change_band_portion (unsigned char *band_portion*)

Function will send out new band portion settings for the current selected band ∗.

Parameters:

band_portion The current band portion

Definition at line 139 of file band_ctrl.c.

References band_ctrl_send_band_data_to_bus().

Referenced by event_aux2_button_pressed(), and main().

6.34.2.3 unsigned char∗ band_ctrl_get_high_output_str (void)

Retrieve the higher frequency output string, of max length BAND_OUTPUT_STR_SIZE.

Returns:

The output string

Definition at line 253 of file band_ctrl.c.

References struct_band::band_high_output_str.

6.34.2.4 unsigned int band_ctrl_get_high_portion_high (unsigned char *band*)

Retrieve the higher frequency limit of the high band limit.

Returns:

The frequency in kHz

Definition at line 247 of file band_ctrl.c.

References band_limits.

Referenced by radio_freq_to_band(), and radio_get_band_portion().

6.34.2.5 unsigned int band_ctrl_get_high_portion_low (unsigned char *band*)

Retrieve the lower frequency limit of the high band limit.

Returns:

The frequency in kHz

Definition at line 241 of file band_ctrl.c.

References band_limits.

Referenced by radio_get_band_portion().

6.34.2.6 unsigned char* band_ctrl_get_low_output_str (void)

Retrieve the lower frequency output string, of max length BAND_OUTPUT_STR_SIZE.

Returns:

The output string

Definition at line 259 of file band_ctrl.c.

References struct_band::band_low_output_str.

6.34.2.7 unsigned int band_ctrl_get_low_portion_high (unsigned char *band*)

Retrieve the higher frequency limit of the low band limit.

Returns:

The frequency in kHz

Definition at line 235 of file band_ctrl.c.

References band_limits.

Referenced by radio_get_band_portion().

6.34.2.8 unsigned int band_ctrl_get_low_portion_low (unsigned char *band*)

Retrieve the lower frequency limit of the low band limit.

Returns:

The frequency in kHz

Definition at line 229 of file band_ctrl.c.

References band_limits.

Referenced by radio_freq_to_band(), and radio_get_band_portion().

6.34.2.9 unsigned char band_ctrl_get_portion (void)

Retrieve which band portion we are currently at.

Returns:

BAND_LOW, BAND_HIGH or BAND_UNDEFINED

Definition at line 265 of file band_ctrl.c.

References struct_runtime_settings::band_change_mode, BAND_CHANGE_MODE_AUTO, BAND_CHANGE_MODE_MANUAL, struct_status::current_band_portion, radio_get_-band_portion(), RADIO_INTERFACE_BCD, radio_interface_get_interface(), runtime_-settings, and status.

6.34.2.10 void band_ctrl_load_band (unsigned char *band*)

Function will load a band from the EEPROM into the current_band struct.

Parameters:

band The index of the band we wish to load from the EEPROM

Definition at line 126 of file band_ctrl.c.

References antenna_ctrl_ant_read_eeprom(), eeprom_get_band_data(), and sub_menu_-load().

Referenced by band_ctrl_change_band().

6.35 front_panel/band_ctrl.h File Reference

Band control functions.

`#include "main.h"`

Classes

- struct struct_band

  Struct of band data.

- struct struct_band_limits

  Struct of the band limits.

Functions

- void band_ctrl_load_band_limits (void)

  Loads the band limits into the band limits struct.

- unsigned int band_ctrl_get_low_portion_low (unsigned char band)

  Retrieve the lower frequency limit of the low band limit.

- unsigned int band_ctrl_get_low_portion_high (unsigned char band)

  Retrieve the higher frequency limit of the low band limit.

- unsigned int band_ctrl_get_high_portion_low (unsigned char band)

  Retrieve the lower frequency limit of the high band limit.

- unsigned int band_ctrl_get_high_portion_high (unsigned char band)

  Retrieve the higher frequency limit of the high band limit.

- unsigned char * band_ctrl_get_high_output_str (void)

  Retrieve the higher frequency output string, of max length BAND_OUTPUT_STR_SIZE.

- unsigned char * band_ctrl_get_low_output_str (void)

  Retrieve the lower frequency output string, of max length BAND_OUTPUT_STR_SIZE.

- void band_ctrl_deactivate_all (void)

  Function which will deactiavate all band outptus, BUS_CMD_DRIVER_DEACTIVATE_-ALL_BAND_OUTPUTS.

- void band_ctrl_change_band_portion (unsigned char band_portion)

  *Function will send out new band portion settings for the current selected band *.*

- void band_ctrl_change_band (unsigned char band)

  Function used to change band.

Variables

- struct_band_limits band_limits [9]

  The band limits, an array with size 9.

6.35.1 Detailed Description

Band control functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/band_ctrl.h"
```

Definition in file band_ctrl.h.

6.35.2 Function Documentation

6.35.2.1 void band_ctrl_change_band (unsigned char *band*)

Function used to change band.

Parameters:

band The band we wish to change to

Definition at line 145 of file band_ctrl.c.

References antenna_ctrl_change_rx_ant(), antenna_ctrl_deactivate_all(), antenna_ctrl_deactivate_all_rx_band(), antenna_ctrl_select_default_ant(), antenna_ctrl_send_rx_ant_band_data_to_bus(), band_ctrl_deactivate_all(), band_ctrl_load_band(), band_ctrl_send_band_data_to_bus(), struct_status::current_band_portion, struct_status::current_display, CURRENT_DISPLAY_ANTENNA_INFO, struct_status::current_display_level, CURRENT_DISPLAY_LOGO, CURRENT_DISPLAY_MENU_SYSTEM, CURRENT_DISPLAY_SHUTDOWN_VIEW, DISPLAY_LEVEL_BAND, FUNC_STATUS_RXANT, struct_status::function_status, INHIBIT_NOT_OK_TO_SEND_RADIO_TX, KNOB_FUNCTION_AUTO, led_set_band(), led_set_band_none(), led_set_rx_ant(), led_set_rxant(), led_set_tx_ant(), LED_STATE_OFF, main_get_inhibit_state(), main_update_display(), main_update_ptt_status(), struct_status::new_band, struct_status::selected_ant, struct_status::selected_band, struct_status::selected_rx_antenna, set_knob_function(), and status.

Referenced by event_internal_comm_parse_message(), event_poll_buttons(), and main().

6.35.2.2 void band_ctrl_change_band_portion (unsigned char *band_portion*)

Function will send out new band portion settings for the current selected band ∗.

Parameters:

band_portion The current band portion

Definition at line 139 of file band_ctrl.c.

References band_ctrl_send_band_data_to_bus().

Referenced by event_aux2_button_pressed(), and main().

6.35.2.3 unsigned char* band_ctrl_get_high_output_str (void)

Retrieve the higher frequency output string, of max length BAND_OUTPUT_STR_SIZE.

Returns:

The output string

Definition at line 253 of file band_ctrl.c.

References struct_band::band_high_output_str.

6.35.2.4 unsigned int band_ctrl_get_high_portion_high (unsigned char *band*)

Retrieve the higher frequency limit of the high band limit.

Returns:

The frequency in kHz

Definition at line 247 of file band_ctrl.c.

References band_limits.

Referenced by radio_freq_to_band(), and radio_get_band_portion().

6.35.2.5 unsigned int band_ctrl_get_high_portion_low (unsigned char *band*)

Retrieve the lower frequency limit of the high band limit.

Returns:

The frequency in kHz

Definition at line 241 of file band_ctrl.c.

References band_limits.

Referenced by radio_get_band_portion().

6.35.2.6 unsigned char* band_ctrl_get_low_output_str (void)

Retrieve the lower frequency output string, of max length BAND_OUTPUT_STR_SIZE.

Returns:

The output string

Definition at line 259 of file band_ctrl.c.

References struct_band::band_low_output_str.

6.35.2.7 unsigned int band_ctrl_get_low_portion_high (unsigned char *band*)

Retrieve the higher frequency limit of the low band limit.

Returns:

The frequency in kHz

Definition at line 235 of file band_ctrl.c.

References band_limits.

Referenced by radio_get_band_portion().

6.35.2.8 unsigned int band_ctrl_get_low_portion_low (unsigned char *band*)

Retrieve the lower frequency limit of the low band limit.

Returns:

The frequency in kHz

Definition at line 229 of file band_ctrl.c.

References band_limits.

Referenced by radio_freq_to_band(), and radio_get_band_portion().

6.36 front_panel/computer_interface.c File Reference

Interface towards the computer.

```
#include <avr/wdt.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <string.h>
#include "computer_interface.h"
#include "radio_interface.h"
#include "usart.h"
#include "ds1307.h"
#include "antenna_ctrl.h"
#include "eeprom.h"
#include "led_control.h"
#include "sequencer.h"
```

Classes

- struct computer_comm_struct

  Computer interface communication struct.

Defines

- #define COMPUTER_RX_BUFFER_LENGTH 128

  The length of the computer RX BUFFER.
- #define COMPUTER_TX_BUFFER_LENGTH 20

  The length of the computer RX BUFFER.
- #define COMPUTER_INTERFACE_FIXED_SIZE 5

  The fixed size of the computer interface structure (PREAMBLE, POSTAMBLE etc).
- #define COMPUTER_COMM_PREAMBLE 0xFE

  The preamble of the computer communication protocol.
- #define COMPUTER_COMM_POSTAMBLE 0xFD

  The postamble of the computer communication protocol.
- #define COMPUTER_COMM_ACK 0xFA

The serial acknowledge of the computer communication protocol.

- #define COMPUTER_COMM_NACK 0xFB

  The serial NOT acknowledge of the computer communication protocol.

- #define COMPUTER_COMM_FLAG_SETUP_MODE 0

  Flag to see if the setup mode is activated.

- #define COMPUTER_COMM_FLAG_FOUND_PREAMBLE 1

  Flag to see if the preamble was found.

- #define COMPUTER_COMM_FLAG_DATA_IN_RX_BUF 2

  Flag to see that there is data in the rx buffer.

- #define COMPUTER_COMM_ENTER_BOOTLOADER 0x01

  Command to force the openASC box into bootloader mode.

- #define CTRL_REBOOT 0x02

  CTRL command: Reboot the device.

- #define CTRL_GET_FIRMWARE_REV 0x03

  CTRL command: Retrieve the firmware revision.

- #define CTRL_DONE 0x04

  This function just replies with the same command, this is so we can see when something has been finished.

- #define CTRL_SET_TIME 0x10

  CTRL section: Set the time of the realtime clock.

- #define CTRL_SET_ANT_DATA 0x11

  CTRL section: Set the TX antenna settings.

- #define CTRL_CREATE_EEPROM_TABLE 0x12

  CTRL section: Create an EEPROM table.

- #define CTRL_SET_RX_ANT_DATA 0x13

  CTRL section: Set the RX antenna settings.

- #define CTRL_SET_RADIO_SETTINGS 0x14

  CTRL section: Set the radio settings.

- #define CTRL_SET_DEVICE_SETTINGS 0x15

  CTRL section: Set the device settings.

- #define CTRL_SET_BAND_DATA 0x16

  CTRL section: Set the band data settings.

- #define CTRL_SET_EXT_INPUT 0x17

  CTRL section: Set the external input settings.

- #define CTRL_SET_SEQUENCER_SETTINGS 0x18

  CTRL section: Set the sequencer settings.
- #define CTRL_SET_RADIO_SETTINGS_SAVE 0x01

  CTRL command: Save the radio settings.
- #define CTRL_SET_RADIO_SETTINGS_ALL 0x02

  CTRL command: Set all antenna settings.
- #define CTRL_SET_ANT_DATA_SAVE 0x01

  CTRL command: Save the antenna information data to the EEPROM.
- #define CTRL_SET_ANT_DATA_TEXT 0x02

  CTRL command: Set the antenna text.
- #define CTRL_SET_ANT_DATA_SUB_MENU_TYPE 0x03

  CTRL command: Set the antenna sub menu type.
- #define CTRL_SET_ANT_DATA_ANT_FLAGS 0x04

  CTRL command: Set the antenna flags.
- #define CTRL_SET_ANT_DATA_COMB_ALLOWED 0x05

  CTRL command: Set the output combination allowed.
- #define CTRL_SET_ANT_DATA_ANT_OUT_STR 0x06

  CTRL command: Set the antenna output str.
- #define CTRL_SET_ANT_ROTATOR_DATA 0x07

  CTRL command: Set the rotator information.
- #define CTRL_SET_ANT_DEFAULT_INDEX 0x08

  CTRL command: Set the default antenna index.
- #define CTRL_SET_ANT_SUB_MENU_DATA 0x09

  CTRL command: Set the sub menu data.
- #define CTRL_SET_ANT_SUB_MENU_TEXT 0x0A

  CTRL command: Set the sub menu data, text.
- #define CTRL_SET_ANT_SUB_MENU_OUTPUT_STR 0x0B

  CTRL command: Set the sub menu data, output str.
- #define CTRL_SET_BAND_DATA_LIMITS 0x01

  CTRL command: Set the band data limits.
- #define CTRL_SET_BAND_DATA_LOW_OUT_STR 0x02

  CTRL command: Set the band low portion output str.
- #define CTRL_SET_BAND_DATA_HIGH_OUT_STR 0x03

CTRL command: Set the band high portion output str.

- #define CTRL_SET_BAND_DATA_SAVE 0x07

  CTRL command: Save the band data settings.

- #define CTRL_SET_RX_ANT_DATA_TEXT 0x01

  CTRL command: Set the RX antenna text.

- #define CTRL_SET_RX_ANT_DATA_ANT_OUT_STR 0x02

  CTRL command: Set the RX antenna output str.

- #define CTRL_SET_RX_ANT_DATA_BAND_OUT_STR 0x03

  CTRL command: Set the RX antenna band output str.

- #define CTRL_SET_RX_ANT_DATA_SAVE 0x07

  CTRL command: Save the RX antenna settings.

- #define CTRL_SET_DEVICE_SETTINGS_NETWORK 0x01

  CTRL command: Network settings.

- #define CTRL_SET_POWERMETER_SETTINGS 0x02

  CTRL command: Powermeter settings.

- #define CTRL_SET_DEVICE_SETTINGS_OTHER 0x03

  CTRL command: Various settings.

- #define CTRL_SET_DEVICE_SETTINGS_EXT_INPUTS 0x04

  CTRL command: External input settings.

- #define CTRL_SET_DEVICE_SETTINGS_SAVE 0x07

  CTRL command: Save data to eeprom.

- #define CTRL_SET_SEQUENCER_SAVE 0x01

  CTRL command: Save the sequencer settings.

- #define CTRL_SET_SEQUENCER_FOOTSWITCH 0x02

  CTRL command: Set the sequencer footswitch input values.

- #define CTRL_SET_SEQUENCER_COMPUTER 0x03

  CTRL command: Set the sequencer computer input values.

- #define CTRL_SET_SEQUENCER_RADIO_SENSE 0x04

  CTRL command: Set the sequencer radio sense input values.

- #define CTRL_SET_DEVICE_SETTINGS_EXT_INPUTS 0x05

  CTRL command: External input settings.

- #define CTRL_SET_DEVICE_SETTINGS_SAVE 0x07

  CTRL command: Save data to eeprom.

- #define CTRL_SET_EXT_KEYPAD_FUNCTIONS 0x01

Functions

- void computer_interface_init (void)

  Initialize the communication interface towards the computer. Will initialize buffers etc.

- void computer_interface_send_data (void)

  Function which will send data from the tx_buffer to the uart.

- void computer_interface_send (unsigned char command, unsigned int length, char *data)

  Function which will add data to the tx_buffer. Function also sets the flag indicating that the data should be sent.

- void computer_interface_send_ack (void)

  Function which will add an ACK message to the tx_buffer. Also sets a flag that indicates data ready to be sent.

- void computer_interface_send_nack (void)

  Function which will add an NACK message to the tx_buffer. Also sets a flag that indicates data ready to be sent.

- void computer_interface_parse_data (void)

  Function which will parse the data in the rx_buffer and process the command.

- unsigned char computer_interface_is_active (void)

  Retrieve the status if the computer interface is active.

- void computer_interface_activate_setup (void)

  Activate the setup mode of the device. Will mainly just create various buffers needed to store settings.

- void computer_interface_deactivate_setup (void)

  Function which will deactivate the computer setup mode, this will clear up memory space of the allocated buffers in the computer_interface_activate_setup() function.

- **ISR** (SIG_USART1_DATA)
- ISR (SIG_USART1_RECV)

  Interrupt when a character is received over the UART. If computer setup mode is active it will parse the incoming data, otherwise it is used for CAT control.

Variables

- computer_comm_struct computer_comm

  Computer communication structure.

- struct_antenna * antenna_ptr

  Pointer to an area which we create space when configuring the antenna data.

- struct_rx_antennas * rx_antenna_ptr

  Pointer to an area which we create space when configuring the rx antenna data.

- struct_band * band_ptr

  Pointer to an area which we create space when configuring the band data.

- struct_setting * settings_ptr

  Pointer to an area which we create space when configuring the settings.

- struct_ptt * ptt_sequencer_ptr

  Pointer to an area which we create space when configuring the ptt_sequencer.

- struct_radio_settings * radio_settings_ptr

  Pointer to an area which we crate space when configuring the radio settings.

- struct_sub_menu_array * sub_menu_array_ptr [4]

  Pointer to an area which we crate space when configuring the sub menu (array).

- void(* bootloader_start)(void) = (void *)0x1FE00

  Address which we call when we wish to reboot the device (jumps to the bootloader area).

6.36.1 Detailed Description

Interface towards the computer.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/computer_interface.c"
```

Definition in file computer_interface.c.

6.36.2 Define Documentation

6.36.2.1 #define CTRL_SET_DEVICE_SETTINGS_EXT_INPUTS 0x05

CTRL command: External input settings.

CTRL command: Set the external inputs.

Definition at line 176 of file computer_interface.c.

6.36.2.2 #define CTRL_SET_DEVICE_SETTINGS_EXT_INPUTS 0x04

CTRL command: External input settings.

CTRL command: Set the external inputs.

Definition at line 176 of file computer_interface.c.

Referenced by computer_interface_parse_data().

6.36.2.3 #define CTRL_SET_DEVICE_SETTINGS_SAVE 0x07

CTRL command: Save data to eeprom.

CTRL command: Save the external input settings.

Definition at line 178 of file computer_interface.c.

6.36.2.4 #define CTRL_SET_DEVICE_SETTINGS_SAVE 0x07

CTRL command: Save data to eeprom.

CTRL command: Save the external input settings.

Definition at line 178 of file computer_interface.c.

Referenced by computer_interface_parse_data().

6.36.2.5 #define CTRL_SET_EXT_KEYPAD_FUNCTIONS 0x01

CTRL command: Set the external keypad function

Definition at line 181 of file computer_interface.c.

6.36.3 Function Documentation

6.36.3.1 unsigned char computer_interface_is_active (void)

Retrieve the status if the computer interface is active.

Returns:

1 if it is active, 0 otherwise

Definition at line 686 of file computer_interface.c.

References COMPUTER_COMM_FLAG_SETUP_MODE, and computer_comm_-struct::flags.

Referenced by event_internal_comm_parse_message(), ISR(), and main().

6.36.3.2 void computer_interface_parse_data (void)

Function which will parse the data in the rx_buffer and process the command.

Bit 0 = Footswitch Bit 1 = Radio sense lower floor Bit 2 = Radio sense upper floor Bit 3 = Computer RTS Bit 4 = Inverted radio sense Bit 5 = Inverted Computer RTS Bit 6 = Inhibit polarity (0=active low, 1=active high)

unsigned char ptt_input;

Definition at line 304 of file computer_interface.c.

Referenced by main().

6.36.3.3 void computer_interface_send (unsigned char *command*, unsigned int *length*, char * *data*)

Function which will add data to the tx_buffer. Function also sets the flag indicating that the data should be sent.

Parameters:

command The command we wish to sendchar(

length Number of bytes of data to be sent (only size of the data variable)

data The data we wish to send

Definition at line 263 of file computer_interface.c.

References COMPUTER_COMM_POSTAMBLE, COMPUTER_COMM_PREAMBLE, COMPUTER_INTERFACE_FIXED_SIZE, computer_comm_struct::data_in_tx_buffer, computer_comm_struct::tx_buffer, and computer_comm_struct::tx_buffer_length.

Referenced by computer_interface_parse_data(), and parse_internal_comm_message().

6.37 motherboard/computer_interface.c File Reference

Interface towards the computer.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <string.h>
#include <avr/wdt.h>
#include "computer_interface.h"
#include "usart.h"
#include "main.h"
#include "../internal_comm.h"
#include "../internal_comm_commands.h"
```

Classes

- struct computer_comm_struct
  Computer interface communication struct.

Defines

- #define COMPUTER_RX_BUFFER_LENGTH 128
  The length of the computer RX BUFFER.
- #define COMPUTER_TX_BUFFER_LENGTH 20
  The length of the computer RX BUFFER.
- #define COMPUTER_INTERFACE_FIXED_SIZE 5
  The number of bytes the fixed part of the data structure takes up.
- #define COMPUTER_COMM_PREAMBLE 0xFE
  The preamble of the computer communication protocol.
- #define COMPUTER_COMM_POSTAMBLE 0xFD
  The postamble of the computer communication protocol.
- #define COMPUTER_COMM_ACK 0xFA
  The serial acknowledge of the computer communication protocol.
- #define COMPUTER_COMM_NACK 0xFB
  The serial NOT acknowledge of the computer communication protocol.

- #define COMPUTER_COMM_REDIRECT_DATA 0x10

  Command which is used just to redirect data from the USB to the front panel.

- #define COMPUTER_COMM_FLAG_FOUND_PREAMBLE 1

  Flag to see if the preamble was found.

- #define COMPUTER_COMM_FLAG_DATA_IN_RX_BUF 2

  Flag to see that there is data in the rx buffer.

Functions

- void computer_interface_init (void)

  Initialize the communication interface towards the computer.

- void computer_interface_send_data (void)

  Send data to the computer.

- void computer_interface_send (unsigned char command, unsigned int length, char ∗data)

  Send data to the computer.

- void computer_interface_send_ack (void)

  Send an ACK message.

- void computer_interface_send_nack (void)

  Send a NACK message.

- void computer_interface_parse_data (void)

  Parse the data in the rx_buffer and execute the proper functions.

- **ISR** (SIG_USART1_DATA)
- ISR (SIG_USART1_RECV)

Variables

- computer_comm_struct computer_comm

  Computer communication data.

- void(∗ bootloader_start)(void) = (void ∗)0x1FE00

  The bootloader start address.

6.37.1 Detailed Description

Interface towards the computer.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "computer_interface.c"
```

Definition in file computer_interface.c.

6.37.2 Function Documentation

6.37.2.1 void computer_interface_init (void)

Initialize the communication interface towards the computer.

Initialize the communication interface towards the computer. Will initialize buffers etc.

Definition at line 106 of file computer_interface.c.

References COMPUTER_RX_BUFFER_LENGTH, COMPUTER_TX_BUFFER_LENGTH, computer_comm_struct::data_in_tx_buffer, computer_comm_struct::rx_buffer, computer_-comm_struct::rx_buffer_start, computer_comm_struct::tx_buffer, and computer_comm_-struct::tx_buffer_start.

6.37.2.2 void computer_interface_parse_data (void)

Parse the data in the rx_buffer and execute the proper functions.

Function which will parse the data in the rx_buffer and process the command.

Bit 0 = Footswitch Bit 1 = Radio sense lower floor Bit 2 = Radio sense upper floor Bit 3 = Computer RTS Bit 4 = Inverted radio sense Bit 5 = Inverted Computer RTS Bit 6 = Inhibit polarity (0=active low, 1=active high)

unsigned char ptt_input;

Definition at line 175 of file computer_interface.c.

References struct_ptt_sequencer::active, struct_ptt_sequencer::amp_post_delay, struct_-ptt_sequencer::amp_pre_delay, struct_antenna::antenna_comb_allowed, struct_-antenna::antenna_comb_output_str, struct_antenna::antenna_flag, struct_antenna::antenna_-output_length, struct_antenna::antenna_text, struct_antenna::antenna_text_length, struct_ptt_sequencer::antennas_post_delay, struct_band::band_high_output_str, struct_-band::band_high_output_str_length, struct_band::band_low_output_str, struct_-band::band_low_output_str_length, struct_rx_antennas::band_output_length, struct_rx_-antennas::band_output_str, struct_radio_settings::baudrate, bootloader_start, struct_radio_-settings::cat_enabled, struct_radio_settings::civ_addr, computer_comm_struct::command, struct_ptt::computer, COMPUTER_COMM_ENTER_BOOTLOADER, COMPUTER_-COMM_FLAG_DATA_IN_RX_BUF, computer_interface_deactivate_setup(), computer_-interface_send(), computer_interface_send_ack(), computer_interface_send_nack(), CTRL_CREATE_EEPROM_TABLE, CTRL_DONE, CTRL_GET_FIRMWARE_-REV, CTRL_REBOOT, CTRL_SET_ANT_DATA, CTRL_SET_ANT_DATA_ANT_-FLAGS, CTRL_SET_ANT_DATA_ANT_OUT_STR, CTRL_SET_ANT_DATA_-COMB_ALLOWED, CTRL_SET_ANT_DATA_SAVE, CTRL_SET_ANT_DATA_-SUB_MENU_TYPE, CTRL_SET_ANT_DATA_TEXT, CTRL_SET_ANT_DEFAULT_-INDEX, CTRL_SET_ANT_ROTATOR_DATA, CTRL_SET_ANT_SUB_MENU_DATA, CTRL_SET_ANT_SUB_MENU_OUTPUT_STR, CTRL_SET_ANT_SUB_MENU_-TEXT, CTRL_SET_BAND_DATA, CTRL_SET_BAND_DATA_HIGH_OUT_STR, CTRL_SET_BAND_DATA_LIMITS, CTRL_SET_BAND_DATA_LOW_OUT_STR, CTRL_SET_BAND_DATA_SAVE, CTRL_SET_DEVICE_SETTINGS, CTRL_SET_-DEVICE_SETTINGS_EXT_INPUTS, CTRL_SET_DEVICE_SETTINGS_NETWORK, CTRL_SET_DEVICE_SETTINGS_OTHER, CTRL_SET_DEVICE_SETTINGS_SAVE, CTRL_SET_POWERMETER_SETTINGS, CTRL_SET_RADIO_SETTINGS, CTRL_-SET_RADIO_SETTINGS_ALL, CTRL_SET_RADIO_SETTINGS_SAVE, CTRL_-SET_RX_ANT_DATA, CTRL_SET_RX_ANT_DATA_ANT_OUT_STR, CTRL_-SET_RX_ANT_DATA_BAND_OUT_STR, CTRL_SET_RX_ANT_DATA_SAVE, CTRL_SET_RX_ANT_DATA_TEXT, CTRL_SET_SEQUENCER_COMPUTER, CTRL_SET_SEQUENCER_FOOTSWITCH, CTRL_SET_SEQUENCER_RADIO_-SENSE, CTRL_SET_SEQUENCER_SAVE, CTRL_SET_SEQUENCER_SETTINGS, CTRL_SET_TIME, struct_antenna::default_antenna, struct_sub_menu_array::direction_-count, struct_sub_menu_array::direction_name, ds1307_set_time(), eeprom_create_table(), eeprom_save_ant_structure(), eeprom_save_ant_sub_menu_array_structure(), eeprom_-save_band_data(), eeprom_save_ptt_data(), eeprom_save_radio_settings_structure(), eeprom_save_rx_ant_structure(), eeprom_save_settings_structure(), struct_setting::ext_-key_assignments, FIRMWARE_REV, computer_comm_struct::flags, struct_ptt::footswitch, struct_band::high_portion_high_limit, struct_band::high_portion_low_limit, struct_ptt_-sequencer::inhibit_post_delay, struct_ptt_sequencer::inhibit_pre_delay, INT_COMM_-PC_CTRL, INT_COMM_REDIRECT_DATA, struct_radio_settings::interface_type, internal_comm_add_tx_message(), computer_comm_struct::length, struct_band::low_-portion_high_limit, struct_band::low_portion_low_limit, struct_rx_antennas::name, struct_rx_antennas::name_length, struct_setting::network_address, struct_setting::network_-device_count, struct_setting::network_device_is_master, struct_rx_antennas::output_-length, struct_rx_antennas::output_str, struct_sub_menu_array::output_str_dir, struct_-sub_menu_array::output_str_dir_length, struct_radio_settings::poll_interval, struct_-setting::powermeter_address, struct_setting::powermeter_update_rate_bargraph, struct_-setting::powermeter_update_rate_text, struct_setting::powermeter_vswr_limit, struct_-ptt::ptt_input, struct_radio_settings::ptt_input, struct_setting::ptt_interlock_input, struct_radio_settings::radio_model, struct_ptt_sequencer::radio_post_delay, struct_ptt_-sequencer::radio_pre_delay, struct_ptt::radio_sense, struct_antenna::rotator_addr, struct_-antenna::rotator_delay, struct_antenna::rotator_max_rotation, struct_antenna::rotator_-min_heading, struct_antenna::rotator_sub_addr, struct_antenna::rotator_view_360_deg, computer_comm_struct::rx_buffer, computer_comm_struct::rx_buffer_start, struct_radio_-settings::stopbits, struct_antenna::sub_menu_type, SUBMENU_STACK, and SUBMENU_-VERT_ARRAY.

6.37.2.3 void computer_interface_send (unsigned char *command*, unsigned int *length*, char * *data*)

Send data to the computer.

Parameters:

command The command we wish to send

length The length of the data

data The data we wish to send

Definition at line 134 of file computer_interface.c.

References COMPUTER_COMM_POSTAMBLE, COMPUTER_COMM_PREAMBLE, COMPUTER_INTERFACE_FIXED_SIZE, computer_comm_struct::data_in_tx_buffer, computer_comm_struct::tx_buffer, and computer_comm_struct::tx_buffer_length.

6.37.2.4 void computer_interface_send_data (void)

Send data to the computer.

Function which will send data from the tx_buffer to the uart.

Definition at line 118 of file computer_interface.c.

References computer_comm_struct::data_in_tx_buffer, computer_comm_struct::tx_buffer, computer_comm_struct::tx_buffer_length, computer_comm_struct::tx_buffer_start, and usart1_transmit().

6.37.2.5 ISR (SIG_USART1_RECV)

Interrupt which is called when a byte has been received

Definition at line 197 of file computer_interface.c.

References computer_comm_struct::command, COMPUTER_COMM_FLAG_DATA_-IN_RX_BUF, COMPUTER_COMM_FLAG_FOUND_PREAMBLE, COMPUTER_-COMM_POSTAMBLE, COMPUTER_COMM_PREAMBLE, COMPUTER_RX_BUFFER_-LENGTH, computer_comm_struct::count, computer_comm_struct::flags, computer_comm_-struct::length, computer_comm_struct::rx_buffer, and computer_comm_struct::rx_buffer_-start.

6.38 front_panel/computer_interface.h File Reference

Interface towards the computer.

Functions

- void computer_interface_init (void)

  Initialize the communication interface towards the computer. Will initialize buffers etc.
- void computer_interface_send_data (void)

  Function which will send data from the tx_buffer to the uart.
- void computer_interface_parse_data (void)

  Function which will parse the data in the rx_buffer and process the command.
- void computer_interface_activate_setup (void)

  Activate the setup mode of the device. Will mainly just create various buffers needed to store settings.
- void computer_interface_deactivate_setup (void)

  Function which will deactivate the computer setup mode, this will clear up memory space of the allocated buffers in the computer_interface_activate_setup() function.
- unsigned char computer_interface_is_active (void)

  Retrieve the status if the computer interface is active.

6.38.1 Detailed Description

Interface towards the computer.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/computer_interface.h"
```

Definition in file computer_interface.h.

6.38.2 Function Documentation

6.38.2.1 unsigned char computer_interface_is_active (void)

Retrieve the status if the computer interface is active.

Returns:

1 if it is active, 0 otherwise

Definition at line 686 of file computer_interface.c.

References COMPUTER_COMM_FLAG_SETUP_MODE, and computer_comm_-struct::flags.

Referenced by event_internal_comm_parse_message(), ISR(), and main().

6.38.2.2 void computer_interface_parse_data (void)

Function which will parse the data in the rx_buffer and process the command.

Bit 0 = Footswitch Bit 1 = Radio sense lower floor Bit 2 = Radio sense upper floor Bit 3 = Computer RTS Bit 4 = Inverted radio sense Bit 5 = Inverted Computer RTS Bit 6 = Inhibit polarity (0=active low, 1=active high)

unsigned char ptt_input;

Definition at line 304 of file computer_interface.c.

6.39 motherboard/computer_interface.h File Reference

Interface towards the computer.

Functions

- void computer_interface_init (void)

  Initialize the communication interface towards the computer. Will initialize buffers etc.

- void computer_interface_send_data (void)

  Function which will send data from the tx_buffer to the uart.

- void computer_interface_parse_data (void)

  Function which will parse the data in the rx_buffer and process the command.

6.39.1 Detailed Description

Interface towards the computer.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "computer_interface.h"
```

Definition in file computer_interface.h.

6.39.2 Function Documentation

6.39.2.1 void computer_interface_parse_data (void)

Function which will parse the data in the rx_buffer and process the command.

Bit 0 = Footswitch Bit 1 = Radio sense lower floor Bit 2 = Radio sense upper floor Bit 3 = Computer RTS Bit 4 = Inverted radio sense Bit 5 = Inverted Computer RTS Bit 6 = Inhibit polarity (0=active low, 1=active high)

unsigned char ptt_input;

Bit 0 = Footswitch Bit 1 = Radio sense lower floor Bit 2 = Radio sense upper floor Bit 3 = Computer RTS Bit 4 = Inverted radio sense Bit 5 = Inverted Computer RTS Bit 6 = Inhibit polarity (0=active low, 1=active high)

unsigned char ptt_input;

Definition at line 304 of file computer_interface.c.

References struct_ptt_sequencer::active, struct_ptt_sequencer::amp_post_delay, struct_-ptt_sequencer::amp_pre_delay, struct_antenna::antenna_comb_allowed, struct_-antenna::antenna_comb_output_str, struct_antenna::antenna_flag, struct_antenna::antenna_-output_length, struct_antenna::antenna_text, struct_antenna::antenna_text_length, struct_ptt_sequencer::antennas_post_delay, struct_band::band_high_output_str, struct_-band::band_high_output_str_length, struct_band::band_low_output_str, struct_-band::band_low_output_str_length, struct_rx_antennas::band_output_length, struct_rx_-antennas::band_output_str, struct_radio_settings::baudrate, bootloader_start, struct_radio_-settings::cat_enabled, struct_radio_settings::civ_addr, computer_comm_struct::command, struct_ptt::computer, COMPUTER_COMM_ENTER_BOOTLOADER, COMPUTER_-COMM_FLAG_DATA_IN_RX_BUF, computer_interface_deactivate_setup(), computer_-interface_send(), computer_interface_send_ack(), computer_interface_send_nack(), CTRL_CREATE_EEPROM_TABLE, CTRL_DONE, CTRL_GET_FIRMWARE_-REV, CTRL_REBOOT, CTRL_SET_ANT_DATA, CTRL_SET_ANT_DATA_ANT_-FLAGS, CTRL_SET_ANT_DATA_ANT_OUT_STR, CTRL_SET_ANT_DATA_-COMB_ALLOWED, CTRL_SET_ANT_DATA_SAVE, CTRL_SET_ANT_DATA_-SUB_MENU_TYPE, CTRL_SET_ANT_DATA_TEXT, CTRL_SET_ANT_DEFAULT_-INDEX, CTRL_SET_ANT_ROTATOR_DATA, CTRL_SET_ANT_SUB_MENU_DATA, CTRL_SET_ANT_SUB_MENU_OUTPUT_STR, CTRL_SET_ANT_SUB_MENU_-TEXT, CTRL_SET_BAND_DATA, CTRL_SET_BAND_DATA_HIGH_OUT_STR, CTRL_SET_BAND_DATA_LIMITS, CTRL_SET_BAND_DATA_LOW_OUT_STR, CTRL_SET_BAND_DATA_SAVE, CTRL_SET_DEVICE_SETTINGS, CTRL_SET_-DEVICE_SETTINGS_EXT_INPUTS, CTRL_SET_DEVICE_SETTINGS_NETWORK, CTRL_SET_DEVICE_SETTINGS_OTHER, CTRL_SET_DEVICE_SETTINGS_SAVE, CTRL_SET_POWERMETER_SETTINGS, CTRL_SET_RADIO_SETTINGS, CTRL_-SET_RADIO_SETTINGS_ALL, CTRL_SET_RADIO_SETTINGS_SAVE, CTRL_-SET_RX_ANT_DATA, CTRL_SET_RX_ANT_DATA_ANT_OUT_STR, CTRL_-SET_RX_ANT_DATA_BAND_OUT_STR, CTRL_SET_RX_ANT_DATA_SAVE, CTRL_SET_RX_ANT_DATA_TEXT, CTRL_SET_SEQUENCER_COMPUTER, CTRL_SET_SEQUENCER_FOOTSWITCH, CTRL_SET_SEQUENCER_RADIO_-SENSE, CTRL_SET_SEQUENCER_SAVE, CTRL_SET_SEQUENCER_SETTINGS, CTRL_SET_TIME, struct_antenna::default_antenna, struct_sub_menu_array::direction_-count, struct_sub_menu_array::direction_name, ds1307_set_time(), eeprom_create_table(), eeprom_save_ant_structure(), eeprom_save_ant_sub_menu_array_structure(), eeprom_-save_band_data(), eeprom_save_ptt_data(), eeprom_save_radio_settings_structure(), eeprom_save_rx_ant_structure(), eeprom_save_settings_structure(), struct_setting::ext_-key_assignments, FIRMWARE_REV, computer_comm_struct::flags, struct_ptt::footswitch, struct_band::high_portion_high_limit, struct_band::high_portion_low_limit, struct_ptt_-sequencer::inhibit_post_delay, struct_ptt_sequencer::inhibit_pre_delay, INT_COMM_-PC_CTRL, INT_COMM_REDIRECT_DATA, struct_radio_settings::interface_type, internal_comm_add_tx_message(), computer_comm_struct::length, struct_band::low_-portion_high_limit, struct_band::low_portion_low_limit, struct_rx_antennas::name, struct_rx_antennas::name_length, struct_setting::network_address, struct_setting::network_-device_count, struct_setting::network_device_is_master, struct_rx_antennas::output_-length, struct_rx_antennas::output_str, struct_sub_menu_array::output_str_dir, struct_-sub_menu_array::output_str_dir_length, struct_radio_settings::poll_interval, struct_-setting::powermeter_address, struct_setting::powermeter_update_rate_bargraph, struct_-setting::powermeter_update_rate_text, struct_setting::powermeter_vswr_limit, struct_-ptt::ptt_input, struct_radio_settings::ptt_input, struct_setting::ptt_interlock_input, struct_radio_settings::radio_model, struct_ptt_sequencer::radio_post_delay, struct_ptt_-sequencer::radio_pre_delay, struct_ptt::radio_sense, struct_antenna::rotator_addr, struct_-antenna::rotator_delay, struct_antenna::rotator_max_rotation, struct_antenna::rotator_-min_heading, struct_antenna::rotator_sub_addr, struct_antenna::rotator_view_360_deg, computer_comm_struct::rx_buffer, computer_comm_struct::rx_buffer_start, struct_radio_-settings::stopbits, struct_antenna::sub_menu_type, SUBMENU_STACK, and SUBMENU_-VERT_ARRAY.

Referenced by main().

6.40 front_panel/display.c File Reference

The serial interface to configure the device and control it.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/io.h>
#include "display.h"
#include "glcd.h"
#include "fonts.h"
#include "../global.h"
#include "main.h"
#include "ds1307.h"
#include "antenna_ctrl.h"
#include "radio_interface.h"
#include "sub_menu.h"
```

Functions

- void display_shutdown_view (void)

  Display the shutdown in progress screen.
- void display_setup_view (void)

  Display the setup in progress screen.
- unsigned char display_screensaver_mode (void)

  Get the mode of the screensaver.
- void display_update_screensaver (void)

  Updates the screensaver which consist of a clock.
- void display_set_backlight (unsigned char value)

  Set the backlight level of the LCD.
- unsigned char display_calculate_width (char *str, unsigned char font, unsigned char length)

  Retrieve the width of a string in pixels.
- void display_text_center_adjust (unsigned char y, char *str, unsigned char length, unsigned char font)

  Displays a piece of text center adjusted on the display.
- void display_text_right_adjust (unsigned char x, unsigned char y, char *str, unsigned char length, unsigned char font)

  Displays a piece of text right adjusted on the display.

- void display_antennas (unsigned char band)

  Display a set of antennas on the display.
- void display_rotator_directions (unsigned char band)

  Display the current rotator directions If the rotator option has been enabled for a certain antenna it will be shown it's current direction on the LCD.
- void display_invert_antenna (unsigned char ant_index)

  Displays an antenna but inverted.
- void display_radio_freq (unsigned char length, char *freq)

  Displays the radios frequency Will display the radios frequency in the bottom right corner of the display.
- void display_view (unsigned char mode)

  Displays a specified view This is to display lines, icons etc depending on what kind of of "view" you wish to display.
- void display_show_rx_ant (unsigned char ant_index)

  Show the current selected RX ant.
- void display_show_set_heading (unsigned int rotator_heading, unsigned char view_360_-deg)

  Show SET rotator heading.
- void display_update (unsigned char band, unsigned char antenna)

  Updates the display.
- void display_update_radio_freq (void)

  Update the radio frequency area of the display.
- void display_show_sub_menu (unsigned char ant_index, unsigned char sub_menu_-type)

  Will show the sub menu of a certain antenna.
- void display_show_bargraph_fwd (unsigned char percent)

  Will display the forward bargraph.
- void display_show_bargraph_ref (unsigned char percent)

  Will display the reflected bargraph.
- void display_show_powermeter_bargraph (unsigned int fwd_power, unsigned int ref_-power)

  Show the power meter bargraphs.
- void display_show_powermeter_text (unsigned int fwd_power, unsigned int ref_power, unsigned int vswr)

  This function will print out the power meter text which shows FWD, REF power and VSWR.
- void display_show_powermeter (void)

  This function will show the power meter display.

Variables

- unsigned char screensaver_mode = 0

  Flag which indicates if the screensaver is activated or not.

- unsigned char last_fwd_val = 0

  The last forward value, used for the power meter bargraph update.

- unsigned char last_ref_val = 0

  The last reflected value, used for the power meter bargraph update.

- char ∗ temp_ptr = NULL

  Memory area used for printing variables to the display.

6.40.1 Detailed Description

The serial interface to configure the device and control it.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/display.c"
```

Definition in file display.c.

6.40.2 Function Documentation

6.40.2.1 void display_antennas (unsigned char *band*)

Display a set of antennas on the display.

Parameters:

band The band you wish to show the antennas from

Definition at line 149 of file display.c.

References antenna_ctrl_get_antenna_text(), antenna_ctrl_get_antenna_text_length(), antenna_ctrl_get_flags(), ANTENNA_IN_USE_FLAG, CLEAR_ANT_AREA, DISPLAY_-TEXT_ANT1_X_POS, DISPLAY_TEXT_ANT1_Y_POS, DISPLAY_TEXT_ANT2_X_-POS, DISPLAY_TEXT_ANT2_Y_POS, DISPLAY_TEXT_ANT3_X_POS, DISPLAY_-TEXT_ANT3_Y_POS, DISPLAY_TEXT_ANT4_X_POS, and DISPLAY_TEXT_ANT4_-Y_POS.

Referenced by display_update().

6.40.2.2 unsigned char display_calculate_width (char * *str*, unsigned char *font*, unsigned char *length*)

Retrieve the width of a string in pixels.

Parameters:

str The text string you wish to find out the length of

font Which font type the string is

length The length of the string (strlen)

Returns:

The actual graphical width of the text string sent in, in pixels

Definition at line 111 of file display.c.

Referenced by display_text_center_adjust(), display_text_right_adjust(), and menu_show_-text().

6.40.2.3 void display_invert_antenna (unsigned char *ant_index*)

Displays an antenna but inverted.

Parameters:

ant_index Which antenna you wish to invert

Definition at line 241 of file display.c.

References antenna_ctrl_get_antenna_text_length(), antenna_ctrl_get_flags(), ANTENNA_-IN_USE_FLAG, DISPLAY_TEXT_ANT1_Y_POS, DISPLAY_TEXT_ANT2_Y_POS, DISPLAY_TEXT_ANT3_Y_POS, DISPLAY_TEXT_ANT4_Y_POS, DISPLAY_-TEXT_ANT_HEIGHT, DISPLAY_TEXT_ANTENNA_IN_USE_ADDITION_WIDTH, and DISPLAY_TEXT_ANTENNA_WIDTH.

Referenced by display_update().

6.40.2.4 void display_radio_freq (unsigned char *length*, char * *freq*)

Displays the radios frequency Will display the radios frequency in the bottom right corner of the display.

Parameters:

freq the frequency you want to display

Definition at line 286 of file display.c.

References CLEAR_RADIO_FREQ_AREA, DISPLAY_RADIO_FREQ_X_POS, DISPLAY_RADIO_FREQ_Y_POS, and display_text_right_adjust().

Referenced by display_update_radio_freq().

6.40.2.5 void display_rotator_directions (unsigned char *band*)

Display the current rotator directions If the rotator option has been enabled for a certain antenna it will be shown it's current direction on the LCD.

Parameters:

band The band you wish to show the rotators direction

Definition at line 192 of file display.c.

References antenna_ctrl_get_direction(), antenna_ctrl_get_flags(), antenna_ctrl_get_-sub_menu_type(), ANTENNA_ROTATOR_FLAG, CLEAR_ROTATOR_AREA, display_-text_right_adjust(), DISPLAY_TEXT_ROTATOR_ANT1_X_POS, DISPLAY_TEXT_-ROTATOR_ANT1_Y_POS, DISPLAY_TEXT_ROTATOR_ANT2_X_POS, DISPLAY_-TEXT_ROTATOR_ANT2_Y_POS, DISPLAY_TEXT_ROTATOR_ANT3_X_POS, DISPLAY_TEXT_ROTATOR_ANT3_Y_POS, DISPLAY_TEXT_ROTATOR_ANT4_-X_POS, DISPLAY_TEXT_ROTATOR_ANT4_Y_POS, sub_menu_get_current_pos(), sub_menu_get_text(), and SUBMENU_VERT_ARRAY.

Referenced by display_update().

6.40.2.6 unsigned char display_screensaver_mode (void)

Get the mode of the screensaver.

Returns:

0 if the screensaver is disabled, 1 otherwise

Definition at line 69 of file display.c.

References screensaver_mode.

6.40.2.7 void display_set_backlight (unsigned char *value*)

Set the backlight level of the LCD.

Parameters:

value What we wish to set the backlight level to to, 0-100%

Definition at line 97 of file display.c.

Referenced by main(), and menu_action().

6.40.2.8 void display_show_bargraph_fwd (unsigned char *percent*)

Will display the forward bargraph.

Parameters:

percent How much we wish to fill it

Definition at line 461 of file display.c.

References last_fwd_val.

Referenced by display_show_powermeter_bargraph().

6.40.2.9 void display_show_bargraph_ref (unsigned char *percent*)

Will display the reflected bargraph.

Parameters:

percent How much we wish to fill it

Definition at line 492 of file display.c.

References last_ref_val.

Referenced by display_show_powermeter_bargraph().

6.40.2.10 void display_show_powermeter_bargraph (unsigned int *fwd_power*, unsigned int *ref_power*)

Show the power meter bargraphs.

Parameters:

fwd_power The forward power in percent

ref_power The reflected power in percent

Definition at line 524 of file display.c.

References display_show_bargraph_fwd(), and display_show_bargraph_ref().

6.40.2.11 void display_show_powermeter_text (unsigned int *fwd_power*, unsigned int *ref_power*, unsigned int *vswr*)

This function will print out the power meter text which shows FWD, REF power and VSWR.

Parameters:

fwd_power Forward power in watts

ref_power Reflected power in watts

vswr The current VSWR, for example 151 means 1.51:1

Definition at line 533 of file display.c.

References display_text_right_adjust().

6.40.2.12 void display_show_rx_ant (unsigned char *ant_index*)

Show the current selected RX ant.

Parameters:

ant_index The antenna index of which antenna that is selected and should be shown

Definition at line 304 of file display.c.

References antenna_ctrl_get_rx_antenna_count(), antenna_ctrl_get_rx_antenna_name(), CLEAR_RX_ANTENNA_AREA, struct_status::current_display_level, DISPLAY_LEVEL_-BAND, DISPLAY_TEXT_RX_ANT_X_POS, DISPLAY_TEXT_RX_ANT_Y_POS, display_view(), status, and VIEW_ANTENNAS.

Referenced by event_update_display().

6.40.2.13 void display_show_set_heading (unsigned int *rotator_heading*, unsigned char *view_360_deg*)

Show SET rotator heading.

Parameters:

rotator_heading The current set rotator heading

view_360_deg The status of the view_360_deg option

Definition at line 326 of file display.c.

References struct_status::current_display_level, DISPLAY_LEVEL_BAND, display_text_-center_adjust(), and status.

Referenced by event_pulse_sensor_down(), event_pulse_sensor_up(), event_rotate_button_-pressed(), event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_button3_-pressed(), and event_tx_button4_pressed().

6.40.2.14 void display_show_sub_menu (unsigned char *ant_index*, unsigned char *sub_menu_type*)

Will show the sub menu of a certain antenna.

Parameters:

ant_index The antenna index (0-3)

sub_menu_type Which type of sub menu it is

Definition at line 438 of file display.c.

References antenna_ctrl_get_antenna_text(), CLEAR_SET_SUB_MENU_ARRAY_AREA, struct_status::current_display_level, DISPLAY_LEVEL_SUBMENU, display_text_center_-adjust(), status, sub_menu_get_current_pos(), sub_menu_get_text(), and SUBMENU_-VERT_ARRAY.

Referenced by event_pulse_sensor_down(), event_pulse_sensor_up(), and event_sub_-button_pressed().

6.40.2.15 void display_text_center_adjust (unsigned char *y*, char ∗ *str*, unsigned char *length*, unsigned char *font*)

Displays a piece of text center adjusted on the display.

Parameters:

y Where the text should be located in y-axis (pixels)

str The string we wish to center adjust to the display

length The length of the string (strlen)

font Which font you wish to show the string with

Definition at line 132 of file display.c.

References display_calculate_width().

Referenced by display_setup_view(), display_show_set_heading(), display_show_sub_menu(), and display_shutdown_view().

6.40.2.16 void display_text_right_adjust (unsigned char *x*, unsigned char *y*, char * *str*, unsigned char *length*, unsigned char *font*)

Displays a piece of text right adjusted on the display.

Parameters:

x Where the right adjust should start (pixels)

y Where the text should be located in y-axis (pixels)

length The length of the string (strlen)

font Which font you wish to show the string with

Definition at line 142 of file display.c.

References display_calculate_width().

Referenced by display_radio_freq(), display_rotator_directions(), display_show_powermeter(), and display_show_powermeter_text().

6.40.2.17 void display_update (unsigned char *band*, unsigned char *antenna*)

Updates the display.

Parameters:

band Which band to show the antenna information from

antenna The antenna combination that is currently selected

Definition at line 358 of file display.c.

References struct_status::current_display_level, display_antennas(), display_invert_antenna(), DISPLAY_LEVEL_BAND, display_rotator_directions(), display_update_radio_freq(), display_view(), status, and VIEW_ANTENNAS.

Referenced by event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_-button3_pressed(), event_tx_button4_pressed(), and event_update_display().

6.40.2.18 void display_view (unsigned char *mode*)

Displays a specified view This is to display lines, icons etc depending on what kind of of "view" you wish to display.

Parameters:

mode Which view mode you wish to display

Definition at line 296 of file display.c.

References glcd_line(), and VIEW_ANTENNAS.

Referenced by display_show_rx_ant(), display_update(), and display_update_radio_freq().

6.41 front_panel/display.h File Reference

The serial interface to configure the device and control it.

`#include "glcd.h"`

`#include "main.h"`

Defines

- #define CLEAR_ANT_AREA() glcd_clear_area(0,90,0,56)

  Macro that clears the antenna area of the LCD.

- #define CLEAR_ROTATOR_AREA() glcd_clear_area(90,128,0,56)

  Macro that clears the rotator area of the LCD.

- #define CLEAR_RADIO_FREQ_AREA() glcd_clear_area(90,128,58,64)

  Macro that clears the radio frequency area of the LCD.

- #define CLEAR_RX_ANTENNA_AREA() glcd_clear_area(0,90,58,64)

  Macro that clears the RX antenna area.

- #define CLEAR_SET_ROTATOR_AREA() glcd_clear_area(0,80,58,64)

  Macro that clears the rotator area.

- #define CLEAR_SET_SUB_MENU_ARRAY_AREA() glcd_clear_area(44,84,35,64)

  Macro that clears the sub menu array direction area.

- #define **DISPLAY_SCREENSAVER_DEF_CONTRAST** 20
- #define DISPLAY_RADIO_FREQ_X_POS 128

  The position of the frequency text X position.

- #define DISPLAY_RADIO_FREQ_Y_POS 58

  The position of the frequency text Y position.

- #define DISPLAY_TEXT_ROTATOR_ANT1_X_POS 128

  The position of the antenna1 rotator text X position.

- #define DISPLAY_TEXT_ROTATOR_ANT1_Y_POS 1

  The position of the antenna1 rotator text Y position.

- #define DISPLAY_TEXT_ROTATOR_ANT2_X_POS 128

  The position of the antenna2 rotator text X position.

- #define DISPLAY_TEXT_ROTATOR_ANT2_Y_POS 16

  The position of the antenna2 rotator text Y position.

- #define DISPLAY_TEXT_ROTATOR_ANT3_X_POS 128

  The position of the antenna3 rotator text X position.

- #define DISPLAY_TEXT_ROTATOR_ANT3_Y_POS 29

The position of the antenna3 rotator text Y position.

- #define DISPLAY_TEXT_ROTATOR_ANT4_X_POS 128

  The position of the antenna4 rotator text X position.

- #define DISPLAY_TEXT_ROTATOR_ANT4_Y_POS 43

  The position of the antenna4 rotator text Y position.

- #define DISPLAY_TEXT_ANT1_X_POS 0

  The position of the antenna1 text X position.

- #define DISPLAY_TEXT_ANT1_Y_POS 0

  The position of the antenna1 text Y position.

- #define DISPLAY_TEXT_ANT2_X_POS 0

  The position of the antenna2 text X position.

- #define DISPLAY_TEXT_ANT2_Y_POS 15

  The position of the antenna2 text Y position.

- #define DISPLAY_TEXT_ANT3_X_POS 0

  The position of the antenna3 text X position.

- #define DISPLAY_TEXT_ANT3_Y_POS 28

  The position of the antenna3 text Y position.

- #define DISPLAY_TEXT_ANT4_X_POS 0

  The position of the antenna4 text X position.

- #define DISPLAY_TEXT_ANT4_Y_POS 42

  The position of the antenna4 text Y position.

- #define DISPLAY_TEXT_RX_ANT_X_POS 0

  The position of the RX antenna X pos.

- #define DISPLAY_TEXT_RX_ANT_Y_POS 58

  The position of the RX antenna Y position.

- #define DISPLAY_TEXT_ROTATE_ANT_X_POS 0

  The position of the SET ROTATE X pos.

- #define DISPLAY_TEXT_ROTATE_ANT_Y_POS 58

  The position of the SET ROTATE Y position.

- #define DISPLAY_TEXT_ANT_HEIGHT 10

  The height of the antenna text inverting area.

- #define DISPLAY_TEXT_ANTENNA_WIDTH 8

  The width of the antenna text.

- #define DISPLAY_TEXT_ANTENNA_IN_USE_ADDITION_WIDTH 16

  The width addition of the inverted area when an antenna is in use.

Functions

- void display_setup_view (void)

  Display the setup in progress screen.

- void display_shutdown_view (void)

  Display the shutdown in progress screen.

- void display_antennas (unsigned char band)

  Display a set of antennas on the display.

- void display_rotator_directions (unsigned char band)

  Display the current rotator directions If the rotator option has been enabled for a certain antenna it will be shown it's current direction on the LCD.

- void display_radio_freq (unsigned char length, char *freq)

  Displays the radios frequency Will display the radios frequency in the bottom right corner of the display.

- void display_view (unsigned char mode)

  Displays a specified view This is to display lines, icons etc depending on what kind of of "view" you wish to display.

- void display_invert_antenna (unsigned char ant_index)

  Displays an antenna but inverted.

- void display_update_screensaver (void)

  Updates the screensaver which consist of a clock.

- unsigned char display_screensaver_mode (void)

  Get the mode of the screensaver.

- void display_set_backlight (unsigned char value)

  Set the backlight level of the LCD.

- void display_update (unsigned char band, unsigned char antenna)

  Updates the display.

- void display_show_rx_ant (unsigned char ant_index)

  Show the current selected RX ant.

- unsigned char display_calculate_width (char *str, unsigned char font, unsigned char length)

  Retrieve the width of a string in pixels.

- void display_show_set_heading (unsigned int rotator_heading, unsigned char view_360_-deg)

  Show SET rotator heading.

- void display_text_center_adjust (unsigned char y, char ∗str, unsigned char length, unsigned char font)

  Displays a piece of text center adjusted on the display.

- void display_update_radio_freq (void)

  Update the radio frequency area of the display.

- void display_show_sub_menu (unsigned char ant_index, unsigned char sub_menu_-type)

  Will show the sub menu of a certain antenna.

- void display_show_powermeter_bargraph (unsigned int fwd_power, unsigned int ref_-power)

  Show the power meter bargraphs.

- void display_show_powermeter_text (unsigned int fwd_power, unsigned int ref_power, unsigned int vswr)

  This function will print out the power meter text which shows FWD, REF power and VSWR.

- void display_show_powermeter (void)

  This function will show the power meter display.

6.41.1 Detailed Description

The serial interface to configure the device and control it.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/display.h"
```

Definition in file display.h.

6.41.2 Function Documentation

6.41.2.1 void display_antennas (unsigned char *band*)

Display a set of antennas on the display.

Parameters:

band The band you wish to show the antennas from

Definition at line 149 of file display.c.

References antenna_ctrl_get_antenna_text(), antenna_ctrl_get_antenna_text_length(), antenna_ctrl_get_flags(), ANTENNA_IN_USE_FLAG, CLEAR_ANT_AREA, DISPLAY_-TEXT_ANT1_X_POS, DISPLAY_TEXT_ANT1_Y_POS, DISPLAY_TEXT_ANT2_X_-POS, DISPLAY_TEXT_ANT2_Y_POS, DISPLAY_TEXT_ANT3_X_POS, DISPLAY_-TEXT_ANT3_Y_POS, DISPLAY_TEXT_ANT4_X_POS, and DISPLAY_TEXT_ANT4_-Y_POS.

Referenced by display_update().

6.41.2.2 unsigned char display_calculate_width (char * *str*, unsigned char *font*, unsigned char *length*)

Retrieve the width of a string in pixels.

Parameters:

str The text string you wish to find out the length of

font Which font type the string is

length The length of the string (strlen)

Returns:

The actual graphical width of the text string sent in, in pixels

Definition at line 111 of file display.c.

Referenced by display_text_center_adjust(), display_text_right_adjust(), and menu_show_-text().

6.41.2.3 void display_invert_antenna (unsigned char *ant_index*)

Displays an antenna but inverted.

Parameters:

ant_index Which antenna you wish to invert

Definition at line 241 of file display.c.

References antenna_ctrl_get_antenna_text_length(), antenna_ctrl_get_flags(), ANTENNA_-IN_USE_FLAG, DISPLAY_TEXT_ANT1_Y_POS, DISPLAY_TEXT_ANT2_Y_POS, DISPLAY_TEXT_ANT3_Y_POS, DISPLAY_TEXT_ANT4_Y_POS, DISPLAY_-TEXT_ANT_HEIGHT, DISPLAY_TEXT_ANTENNA_IN_USE_ADDITION_WIDTH, and DISPLAY_TEXT_ANTENNA_WIDTH.

Referenced by display_update().

6.41.2.4 void display_radio_freq (unsigned char *length*, char * *freq*)

Displays the radios frequency Will display the radios frequency in the bottom right corner of the display.

Parameters:

freq the frequency you want to display

Definition at line 286 of file display.c.

References CLEAR_RADIO_FREQ_AREA, DISPLAY_RADIO_FREQ_X_POS, DISPLAY_RADIO_FREQ_Y_POS, and display_text_right_adjust().

Referenced by display_update_radio_freq().

6.41.2.5 void display_rotator_directions (unsigned char *band*)

Display the current rotator directions If the rotator option has been enabled for a certain antenna it will be shown it's current direction on the LCD.

Parameters:

band The band you wish to show the rotators direction

Definition at line 192 of file display.c.

References antenna_ctrl_get_direction(), antenna_ctrl_get_flags(), antenna_ctrl_get_-sub_menu_type(), ANTENNA_ROTATOR_FLAG, CLEAR_ROTATOR_AREA, display_-text_right_adjust(), DISPLAY_TEXT_ROTATOR_ANT1_X_POS, DISPLAY_TEXT_-ROTATOR_ANT1_Y_POS, DISPLAY_TEXT_ROTATOR_ANT2_X_POS, DISPLAY_-TEXT_ROTATOR_ANT2_Y_POS, DISPLAY_TEXT_ROTATOR_ANT3_X_POS, DISPLAY_TEXT_ROTATOR_ANT3_Y_POS, DISPLAY_TEXT_ROTATOR_ANT4_-X_POS, DISPLAY_TEXT_ROTATOR_ANT4_Y_POS, sub_menu_get_current_pos(), sub_menu_get_text(), and SUBMENU_VERT_ARRAY.

Referenced by display_update().

6.41.2.6 unsigned char display_screensaver_mode (void)

Get the mode of the screensaver.

Returns:

0 if the screensaver is disabled, 1 otherwise

Definition at line 69 of file display.c.

References screensaver_mode.

6.41.2.7 void display_set_backlight (unsigned char *value*)

Set the backlight level of the LCD.

Parameters:

value What we wish to set the backlight level to to, 0-100%

Definition at line 97 of file display.c.

Referenced by main(), and menu_action().

6.41.2.8 void display_show_powermeter_bargraph (unsigned int *fwd_power*, unsigned int *ref_power*)

Show the power meter bargraphs.

Parameters:

fwd_power The forward power in percent

ref_power The reflected power in percent

Definition at line 524 of file display.c.

References display_show_bargraph_fwd(), and display_show_bargraph_ref().

6.41.2.9 void display_show_powermeter_text (unsigned int *fwd_power*, unsigned int *ref_power*, unsigned int *vswr*)

This function will print out the power meter text which shows FWD, REF power and VSWR.

Parameters:

fwd_power Forward power in watts

ref_power Reflected power in watts

vswr The current VSWR, for example 151 means 1.51:1

Definition at line 533 of file display.c.

References display_text_right_adjust().

6.41.2.10 void display_show_rx_ant (unsigned char *ant_index*)

Show the current selected RX ant.

Parameters:

ant_index The antenna index of which antenna that is selected and should be shown

Definition at line 304 of file display.c.

References antenna_ctrl_get_rx_antenna_count(), antenna_ctrl_get_rx_antenna_name(), CLEAR_RX_ANTENNA_AREA, struct_status::current_display_level, DISPLAY_LEVEL_-BAND, DISPLAY_TEXT_RX_ANT_X_POS, DISPLAY_TEXT_RX_ANT_Y_POS, display_view(), status, and VIEW_ANTENNAS.

Referenced by event_update_display().

6.41.2.11 void display_show_set_heading (unsigned int *rotator_heading*, unsigned char *view_360_deg*)

Show SET rotator heading.

Parameters:

rotator_heading The current set rotator heading

view_360_deg The status of the view_360_deg option

Definition at line 326 of file display.c.

References struct_status::current_display_level, DISPLAY_LEVEL_BAND, display_text_-center_adjust(), and status.

Referenced by event_pulse_sensor_down(), event_pulse_sensor_up(), event_rotate_button_-pressed(), event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_button3_-pressed(), and event_tx_button4_pressed().

6.41.2.12 void display_show_sub_menu (unsigned char *ant_index*, unsigned char *sub_menu_type*)

Will show the sub menu of a certain antenna.

Parameters:

ant_index The antenna index (0-3)

sub_menu_type Which type of sub menu it is

Definition at line 438 of file display.c.

References antenna_ctrl_get_antenna_text(), CLEAR_SET_SUB_MENU_ARRAY_AREA, struct_status::current_display_level, DISPLAY_LEVEL_SUBMENU, display_text_center_-adjust(), status, sub_menu_get_current_pos(), sub_menu_get_text(), and SUBMENU_-VERT_ARRAY.

Referenced by event_pulse_sensor_down(), event_pulse_sensor_up(), and event_sub_-button_pressed().

6.41.2.13 void display_text_center_adjust (unsigned char *y*, char * *str*, unsigned char *length*, unsigned char *font*)

Displays a piece of text center adjusted on the display.

Parameters:

y Where the text should be located in y-axis (pixels)

str The string we wish to center adjust to the display

length The length of the string (strlen)

font Which font you wish to show the string with

Definition at line 132 of file display.c.

References display_calculate_width().

Referenced by display_setup_view(), display_show_set_heading(), display_show_sub_menu(), and display_shutdown_view().

6.41.2.14 void display_update (unsigned char *band*, unsigned char *antenna*)

Updates the display.

Parameters:

band Which band to show the antenna information from

antenna The antenna combination that is currently selected

Definition at line 358 of file display.c.

References struct_status::current_display_level, display_antennas(), display_invert_antenna(), DISPLAY_LEVEL_BAND, display_rotator_directions(), display_update_radio_freq(), display_view(), status, and VIEW_ANTENNAS.

Referenced by event_tx_button1_pressed(), event_tx_button2_pressed(), event_tx_-button3_pressed(), event_tx_button4_pressed(), and event_update_display().

6.41.2.15 void display_view (unsigned char *mode*)

Displays a specified view This is to display lines, icons etc depending on what kind of of "view" you wish to display.

Parameters:

mode Which view mode you wish to display

Definition at line 296 of file display.c.

References glcd_line(), and VIEW_ANTENNAS.

Referenced by display_show_rx_ant(), display_update(), and display_update_radio_freq().

6.42 front_panel/ds1307.c File Reference

Main file of the front panel.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "ds1307.h"
#include "../i2c.h"
#include "../delay.h"
```

Functions

- void ds1307_init (void)

    Initialize the realtime clock on the front panel.
- void ds1307_set_time (char ∗data)

    Set the current time of the realtime clock.
- unsigned char ds1307_get_hours (void)

    Retrieve the hour part of the time from the realtime clock.
- unsigned char ds1307_get_minutes (void)

    Retrieve the minute part of the time from the realtime clock.
- unsigned char ds1307_get_seconds (void)

    Retrieve the seconds part of the time from the realtime clock.
- void ds1307_read (void)

    Read the current time/date from the realtime clock. Stores the data and can be retrieved with the get functions in this file.

Variables

- unsigned char allowed_to_read = 0

    Flag which is set to 1 if a read request is allowed to the ds1307.
- unsigned char ∗ time_data

    Variable which contains information of the current time/date.

6.42.1 Detailed Description

Main file of the front panel.

Realtime clock.

Author:

Mikael Larsmark, SM2WMV

Date:

2008-04-30 /*!

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/ds1307.c"
```

Definition in file ds1307.c.

6.42.2 Function Documentation

6.42.2.1 unsigned char ds1307_get_hours (void)

Retrieve the hour part of the time from the realtime clock.

Returns:

The current hour

Definition at line 85 of file ds1307.c.

Referenced by display_update_screensaver().

6.42.2.2 unsigned char ds1307_get_minutes (void)

Retrieve the minute part of the time from the realtime clock.

Returns:

The current minute

Definition at line 93 of file ds1307.c.

Referenced by display_update_screensaver().

6.42.2.3 unsigned char ds1307_get_seconds (void)

Retrieve the seconds part of the time from the realtime clock.

Returns:

The current seconds

Definition at line 101 of file ds1307.c.

Referenced by display_update_screensaver().

6.42.2.4 void ds1307_set_time (char * *data*)

Set the current time of the realtime clock.

Parameters:

data data[0] = seconds, data[1] = minutes, data[2] = hours, data[3] = Day, data[4] = Date, data[5] = month, data[6] = year

Definition at line 62 of file ds1307.c.

Referenced by computer_interface_parse_data().

6.43 front_panel/ds1307.h File Reference

Realtime clock.

Defines

- #define DS1307_ADDR 0x68

  The external I2C address of the DS1307 realtime clock.

Functions

- void ds1307_init (void)

  Initialize the realtime clock on the front panel.

- void ds1307_read (void)

  Read the current time/date from the realtime clock. Stores the data and can be retrieved with the get functions in this file.

- unsigned char ds1307_get_seconds (void)

  Retrieve the seconds part of the time from the realtime clock.

- unsigned char ds1307_get_minutes (void)

  Retrieve the minute part of the time from the realtime clock.

- unsigned char ds1307_get_hours (void)

  Retrieve the hour part of the time from the realtime clock.

- void ds1307_set_time (char *data)

  Set the current time of the realtime clock.

6.43.1 Detailed Description

Realtime clock.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/ds1307.h"
```

Definition in file ds1307.h.

6.43.2 Function Documentation

6.43.2.1 unsigned char ds1307_get_hours (void)

Retrieve the hour part of the time from the realtime clock.

Returns:

The current hour

Definition at line 85 of file ds1307.c.

Referenced by display_update_screensaver().

6.43.2.2 unsigned char ds1307_get_minutes (void)

Retrieve the minute part of the time from the realtime clock.

Returns:

The current minute

Definition at line 93 of file ds1307.c.

Referenced by display_update_screensaver().

6.43.2.3 unsigned char ds1307_get_seconds (void)

Retrieve the seconds part of the time from the realtime clock.

Returns:

The current seconds

Definition at line 101 of file ds1307.c.

Referenced by display_update_screensaver().

6.43.2.4 void ds1307_set_time (char * *data*)

Set the current time of the realtime clock.

Parameters:

data data[0] = seconds, data[1] = minutes, data[2] = hours, data[3] = Day, data[4] = Date, data[5] = month, data[6] = year

Definition at line 62 of file ds1307.c.

Referenced by computer_interface_parse_data().

6.44 front_panel/eeprom.c File Reference

EEPROM functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/io.h>
#include <avr/eeprom.h>
#include "eeprom_m24.h"
#include "eeprom.h"
#include "../global.h"
#include "antenna_ctrl.h"
#include "band_ctrl.h"
#include "main.h"
#include "radio_interface.h"
```

Functions

- unsigned char eeprom_read_startup_byte (void)
- void eeprom_write_startup_byte (unsigned char val)

  This function will write a byte in the EEPROM so we can keep track of if the unit has ever been started.

- void eeprom_print (void)

  *Temporary crap for debug */.*

- void eeprom_read_table (void)

  Read the map of the EEPROM.

- void eeprom_get_antenna_data (struct_antenna *data, unsigned char band)

  Returns the antenna struct for a specific band.

- void eeprom_get_rx_antenna_data (struct_rx_antennas *data)

  Returns the rx antenna data.

- void eeprom_get_band_data (unsigned char band, struct_band *data)

  Returns the band data.

- void eeprom_get_radio_settings_structure (struct_radio_settings *data)

  get the radio settings from the eeprom

- void eeprom_get_settings_structure (struct_setting *data)

  get the settings from the eeprom

- void eeprom_create_table (void)

Creates the eeprom table which is a map over the eeprom data.

- void eeprom_save_runtime_settings (struct_runtime_settings *content)
  Save the runtime_settings structure to the eeprom.
- void eeprom_get_ptt_data (struct_ptt *data)
  Get the ptt structure from the EEPROM.
- void eeprom_get_runtime_settings (struct_runtime_settings *data)
  Get the runtime_settings structure from the EEPROM.
- void eeprom_get_ant_sub_menu_array_structure (unsigned char band_index, unsigned char ant_index, struct_sub_menu_array *data)
  Get the struct_sub_menu_array structure from the EEPROM.
- void eeprom_save_ant_structure (unsigned char band_index, struct_antenna *content)
  Save the antenna structure to the eeprom.
- void eeprom_save_rx_ant_structure (struct_rx_antennas *data)
  Save the rx antenna structure to the eeprom.
- void eeprom_save_settings_structure (struct_setting *data)
  Save the device settings to the eeprom.
- void eeprom_save_radio_settings_structure (struct_radio_settings *data)
  Save the radio settings to the eeprom.
- void eeprom_save_band_data (unsigned char band, struct_band *data)
  Save the band data to the eeprom.
- void eeprom_save_ptt_data (struct_ptt *data)
  Save the band data to the eeprom.
- void eeprom_save_ant_sub_menu_array_structure (unsigned char band_index, unsigned char ant_index, struct_sub_menu_array *data)
  Save the sub menu array data to the EEPROM.

Variables

- struct_eeprom_table eeprom_table
  EEPROM table which is a description of the location of different structures in the eeprom.

6.44.1 Detailed Description

EEPROM functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/eeprom.c"
```

Definition in file eeprom.c.

6.44.2 Function Documentation

6.44.2.1 void eeprom_get_ant_sub_menu_array_structure (unsigned char *band_index*, unsigned char *ant_index*, struct_sub_menu_array ∗ *data*)

Get the struct_sub_menu_array structure from the EEPROM.

Parameters:

band_index Which band we wish to retrieve the sub menu from

ant_index Which antenna we wish to get the sub menu from

data Pointer to where we wish to store the data

Definition at line 205 of file eeprom.c.

References struct_eeprom_table::antenna1_sub_menu, struct_eeprom_table::antenna2_sub_-menu, struct_eeprom_table::antenna3_sub_menu, struct_eeprom_table::antenna4_sub_-menu, and eeprom_m24_read_byte().

Referenced by sub_menu_load().

6.44.2.2 void eeprom_get_antenna_data (struct_antenna ∗ *data*, unsigned char *band*)

Returns the antenna struct for a specific band.

Parameters:

data Where the data should be saved

band Which band you wish to get the pointer

Definition at line 77 of file eeprom.c.

References struct_eeprom_table::antenna, and eeprom_m24_read_byte().

Referenced by antenna_ctrl_ant_read_eeprom().

6.44.2.3 void eeprom_get_band_data (unsigned char *band*, struct_band ∗ *data*)

Returns the band data.

Parameters:

band Which band we wish to retrieve the data from

data Where the data should be saved

Definition at line 98 of file eeprom.c.

References struct_eeprom_table::band, and eeprom_m24_read_byte().

Referenced by band_ctrl_load_band(), and band_ctrl_load_band_limits().

6.44.2.4 void eeprom_get_ptt_data (struct_ptt ∗ *data*)

Get the ptt structure from the EEPROM.

Parameters:

data A pointer where to store the data

Definition at line 183 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::struct_ptt.

Referenced by sequencer_load_eeprom().

6.44.2.5 void eeprom_get_radio_settings_structure (struct_radio_settings ∗ *data*)

get the radio settings from the eeprom

Parameters:

data Where the data should be saved

Definition at line 108 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::radio_settings.

Referenced by radio_interface_load_eeprom().

6.44.2.6 void eeprom_get_runtime_settings (struct_runtime_settings ∗ *data*)

Get the runtime_settings structure from the EEPROM.

Parameters:

data A pointer where to store the data

Definition at line 193 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::runtime_settings.

Referenced by load_settings().

6.44.2.7 void eeprom_get_rx_antenna_data (struct_rx_antennas ∗ *data*)

Returns the rx antenna data.

Parameters:

data Where the data should be saved

Definition at line 87 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::rx_antennas.

Referenced by antenna_ctrl_rx_ant_read_eeprom().

6.44.2.8 void eeprom_get_settings_structure (struct_setting * *data*)

get the settings from the eeprom

Parameters:

data Where the data should be saved

Definition at line 118 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::settings.

Referenced by load_settings().

6.44.2.9 unsigned char eeprom_read_startup_byte (void)

Will read the startup byte from the EEPROM, which does indicate if the unit has been started before or not

Returns:

The status of the startup byte

Definition at line 45 of file eeprom.c.

References EEPROM_STARTUP_BYTE_ADDR.

Referenced by main().

6.44.2.10 void eeprom_save_ant_structure (unsigned char *band_index*, struct_antenna * *content*)

Save the antenna structure to the eeprom.

Parameters:

band_index Which band it is

content The data to be saved

Definition at line 230 of file eeprom.c.

References struct_eeprom_table::antenna, and eeprom_m24_write_block().

Referenced by computer_interface_parse_data().

6.44.2.11 void eeprom_save_ant_sub_menu_array_structure (unsigned char *band_index*, unsigned char *ant_index*, struct_sub_menu_array * *data*)

Save the sub menu array data to the EEPROM.

Parameters:

band_index The band we wish to save the settings for

ant_index The antenna index of the data

data The data to save to the EEPROM

Definition at line 269 of file eeprom.c.

References struct_eeprom_table::antenna1_sub_menu, struct_eeprom_table::antenna2_sub_-menu, struct_eeprom_table::antenna3_sub_menu, struct_eeprom_table::antenna4_sub_-menu, and eeprom_m24_write_block().

Referenced by computer_interface_parse_data().

6.44.2.12 void eeprom_save_band_data (unsigned char *band*, struct_band ∗ *data*)

Save the band data to the eeprom.

Parameters:

band Which band we wish to save the data to

data The data to save to the EEPROM

Definition at line 255 of file eeprom.c.

References struct_eeprom_table::band, and eeprom_m24_write_block().

Referenced by computer_interface_parse_data().

6.44.2.13 void eeprom_save_ptt_data (struct_ptt ∗ *data*)

Save the band data to the eeprom.

Parameters:

data The data to save to the EEPROM

Definition at line 261 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::struct_ptt.

Referenced by computer_interface_parse_data().

6.44.2.14 void eeprom_save_radio_settings_structure (struct_radio_settings ∗ *data*)

Save the radio settings to the eeprom.

Parameters:

data The data to save to the EEPROM

Definition at line 248 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::radio_settings.

Referenced by computer_interface_parse_data().

6.44.2.15 void eeprom_save_runtime_settings (struct_runtime_settings ∗ *content*)

Save the runtime_settings structure to the eeprom.

Parameters:

content The data to be saved

Definition at line 177 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::runtime_settings.

Referenced by main(), and main_save_settings().

6.44.2.16 void eeprom_save_rx_ant_structure (struct_rx_antennas ∗ *data*)

Save the rx antenna structure to the eeprom.

Parameters:

data The data to save to the EEPROM

Definition at line 236 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::rx_antennas.

Referenced by computer_interface_parse_data().

6.44.2.17 void eeprom_save_settings_structure (struct_setting ∗ *data*)

Save the device settings to the eeprom.

Parameters:

data The data to save to the EEPROM

Definition at line 242 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::settings.

Referenced by computer_interface_parse_data().

6.44.2.18 void eeprom_write_startup_byte (unsigned char *val*)

This function will write a byte in the EEPROM so we can keep track of if the unit has ever been started.

Parameters:

val What value we wish to write to the EEPROM

Definition at line 51 of file eeprom.c.

References EEPROM_STARTUP_BYTE_ADDR.

Referenced by main().

6.45 front_panel/eeprom.h File Reference

EEPROM functions.

```
#include "board.h"
#include "sequencer.h"
#include "main.h"
#include "antenna_ctrl.h"
#include "radio_interface.h"
#include "band_ctrl.h"
#include "sub_menu.h"
```

Classes

- struct struct_eeprom_table

  The EEPROM table.

Defines

- #define EEPROM_STARTUP_BYTE_ADDR 0x01

  Defines where the startup byte is located in the uC EEPROM. This is used to keep track of the device is started for the first time.

Functions

- unsigned char eeprom_read_startup_byte (void)
- void eeprom_write_startup_byte (unsigned char val)

  This function will write a byte in the EEPROM so we can keep track of if the unit has ever been started.

- void eeprom_read_table (void)

  Read the map of the EEPROM.

- void eeprom_create_table (void)

  Creates the eeprom table which is a map over the eeprom data.

- void eeprom_save_runtime_settings (struct_runtime_settings *content)

  Save the runtime_settings structure to the eeprom.

- void eeprom_get_runtime_settings (struct_runtime_settings *data)

  Get the runtime_settings structure from the EEPROM.

- void eeprom_get_ant_sub_menu_array_structure (unsigned char band_index, unsigned char ant_index, struct_sub_menu_array *data)

  Get the struct_sub_menu_array structure from the EEPROM.

- void eeprom_get_antenna_data (struct_antenna *data, unsigned char band)

  Returns the antenna struct for a specific band.
- void eeprom_get_band_data (unsigned char band, struct_band *data)

  Returns the band data.
- void eeprom_get_ptt_data (struct_ptt *data)

  Get the ptt structure from the EEPROM.
- void eeprom_save_ant_structure (unsigned char band_index, struct_antenna *content)

  Save the antenna structure to the eeprom.
- void eeprom_get_radio_settings_structure (struct_radio_settings *data)

  get the radio settings from the eeprom
- void eeprom_get_rx_antenna_data (struct_rx_antennas *data)

  Returns the rx antenna data.
- void eeprom_save_rx_ant_structure (struct_rx_antennas *data)

  Save the rx antenna structure to the eeprom.
- void eeprom_save_radio_settings_structure (struct_radio_settings *data)

  Save the radio settings to the eeprom.
- void eeprom_save_band_data (unsigned char band, struct_band *data)

  Save the band data to the eeprom.
- void eeprom_get_settings_structure (struct_setting *data)

  get the settings from the eeprom
- void eeprom_save_settings_structure (struct_setting *data)

  Save the device settings to the eeprom.
- void eeprom_save_ptt_data (struct_ptt *data)

  Save the band data to the eeprom.
- void eeprom_save_ant_sub_menu_array_structure (unsigned char band_index, unsigned char ant_index, struct_sub_menu_array *data)

  Save the sub menu array data to the EEPROM.

6.45.1 Detailed Description

EEPROM functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/eeprom.h"
```

Definition in file eeprom.h.

6.45.2 Function Documentation

6.45.2.1 void eeprom_get_ant_sub_menu_array_structure (unsigned char *band_index*, unsigned char *ant_index*, struct_sub_menu_array ∗ *data*)

Get the struct_sub_menu_array structure from the EEPROM.

Parameters:

band_index Which band we wish to retrieve the sub menu from
ant_index Which antenna we wish to get the sub menu from
data Pointer to where we wish to store the data

Definition at line 205 of file eeprom.c.

References struct_eeprom_table::antenna1_sub_menu, struct_eeprom_table::antenna2_sub_-menu, struct_eeprom_table::antenna3_sub_menu, struct_eeprom_table::antenna4_sub_-menu, and eeprom_m24_read_byte().

Referenced by sub_menu_load().

6.45.2.2 void eeprom_get_antenna_data (struct_antenna ∗ *data*, unsigned char *band*)

Returns the antenna struct for a specific band.

Parameters:

data Where the data should be saved
band Which band you wish to get the pointer

Definition at line 77 of file eeprom.c.

References struct_eeprom_table::antenna, and eeprom_m24_read_byte().

Referenced by antenna_ctrl_ant_read_eeprom().

6.45.2.3 void eeprom_get_band_data (unsigned char *band*, struct_band ∗ *data*)

Returns the band data.

Parameters:

band Which band we wish to retrieve the data from
data Where the data should be saved

Definition at line 98 of file eeprom.c.

References struct_eeprom_table::band, and eeprom_m24_read_byte().

Referenced by band_ctrl_load_band(), and band_ctrl_load_band_limits().

6.45.2.4 void eeprom_get_ptt_data (struct_ptt ∗ *data*)

Get the ptt structure from the EEPROM.

Parameters:

data A pointer where to store the data

Definition at line 183 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::struct_ptt.

Referenced by sequencer_load_eeprom().

6.45.2.5 void eeprom_get_radio_settings_structure (struct_radio_settings ∗ *data*)

get the radio settings from the eeprom

Parameters:

data Where the data should be saved

Definition at line 108 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::radio_settings.

Referenced by radio_interface_load_eeprom().

6.45.2.6 void eeprom_get_runtime_settings (struct_runtime_settings ∗ *data*)

Get the runtime_settings structure from the EEPROM.

Parameters:

data A pointer where to store the data

Definition at line 193 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::runtime_settings.

Referenced by load_settings().

6.45.2.7 void eeprom_get_rx_antenna_data (struct_rx_antennas ∗ *data*)

Returns the rx antenna data.

Parameters:

data Where the data should be saved

Definition at line 87 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::rx_antennas.

Referenced by antenna_ctrl_rx_ant_read_eeprom().

6.45.2.8 void eeprom_get_settings_structure (struct_setting ∗ *data*)

get the settings from the eeprom

Parameters:

data Where the data should be saved

Definition at line 118 of file eeprom.c.

References eeprom_m24_read_byte(), and struct_eeprom_table::settings.

Referenced by load_settings().

6.45.2.9 unsigned char eeprom_read_startup_byte (void)

Will read the startup byte from the EEPROM, which does indicate if the unit has been started before or not

Returns:

The status of the startup byte

Definition at line 45 of file eeprom.c.

References EEPROM_STARTUP_BYTE_ADDR.

Referenced by main().

6.45.2.10 void eeprom_save_ant_structure (unsigned char *band_index*, struct_antenna ∗ *content*)

Save the antenna structure to the eeprom.

Parameters:

band_index Which band it is

content The data to be saved

Definition at line 230 of file eeprom.c.

References struct_eeprom_table::antenna, and eeprom_m24_write_block().

Referenced by computer_interface_parse_data().

6.45.2.11 void eeprom_save_ant_sub_menu_array_structure (unsigned char *band_index*, unsigned char *ant_index*, struct_sub_menu_array ∗ *data*)

Save the sub menu array data to the EEPROM.

Parameters:

band_index The band we wish to save the settings for

ant_index The antenna index of the data

data The data to save to the EEPROM

Definition at line 269 of file eeprom.c.

References struct_eeprom_table::antenna1_sub_menu, struct_eeprom_table::antenna2_sub_-menu, struct_eeprom_table::antenna3_sub_menu, struct_eeprom_table::antenna4_sub_-menu, and eeprom_m24_write_block().

Referenced by computer_interface_parse_data().

6.45.2.12 void eeprom_save_band_data (unsigned char *band*, struct_band * *data*)

Save the band data to the eeprom.

Parameters:

band Which band we wish to save the data to

data The data to save to the EEPROM

Definition at line 255 of file eeprom.c.

References struct_eeprom_table::band, and eeprom_m24_write_block().

Referenced by computer_interface_parse_data().

6.45.2.13 void eeprom_save_ptt_data (struct_ptt * *data*)

Save the band data to the eeprom.

Parameters:

data The data to save to the EEPROM

Definition at line 261 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::struct_ptt.

Referenced by computer_interface_parse_data().

6.45.2.14 void eeprom_save_radio_settings_structure (struct_radio_settings * *data*)

Save the radio settings to the eeprom.

Parameters:

data The data to save to the EEPROM

Definition at line 248 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::radio_settings.

Referenced by computer_interface_parse_data().

6.45.2.15 void eeprom_save_runtime_settings (struct_runtime_settings * *content*)

Save the runtime_settings structure to the eeprom.

Parameters:

content The data to be saved

Definition at line 177 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::runtime_settings.

Referenced by main(), and main_save_settings().

6.45.2.16 void eeprom_save_rx_ant_structure (struct_rx_antennas ∗ *data*)

Save the rx antenna structure to the eeprom.

Parameters:

data The data to save to the EEPROM

Definition at line 236 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::rx_antennas.

Referenced by computer_interface_parse_data().

6.45.2.17 void eeprom_save_settings_structure (struct_setting ∗ *data*)

Save the device settings to the eeprom.

Parameters:

data The data to save to the EEPROM

Definition at line 242 of file eeprom.c.

References eeprom_m24_write_block(), and struct_eeprom_table::settings.

Referenced by computer_interface_parse_data().

6.45.2.18 void eeprom_write_startup_byte (unsigned char *val*)

This function will write a byte in the EEPROM so we can keep track of if the unit has ever been started.

Parameters:

val What value we wish to write to the EEPROM

Definition at line 51 of file eeprom.c.

References EEPROM_STARTUP_BYTE_ADDR.

Referenced by main().

6.46 front_panel/eeprom_m24.c File Reference

EEPROM hardware functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <string.h>
#include <avr/interrupt.h>
#include "eeprom_m24.h"
#include "../i2c.h"
#include "../i2cconf.h"
```

Functions

- void __inline__ eeprom_tiny_delay (void)
- unsigned char eeprom_m24_write_byte (unsigned int eeprom_address, unsigned char value)

  Write a byte of data to the EEPROM.
- unsigned char eeprom_m24_read_byte (unsigned int eeprom_address)

  Read a byte of data from the EEPROM.
- unsigned char eeprom_m24_write_block (unsigned int start_address, unsigned int length, unsigned char *data)

  Write a block of data to the EEPROM.
- unsigned char eeprom_m24_read_block (unsigned int start_address, unsigned int length, unsigned char *data)

  Read a block of data from the EEPROM - NOT FINISHED!!

6.46.1 Detailed Description

EEPROM hardware functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/eeprom_m24.c"
```

Definition in file eeprom_m24.c.

6.46.2 Function Documentation

6.46.2.1 unsigned char eeprom_m24_read_block (unsigned int *start_address*, unsigned int *length*, unsigned char ∗ *data*)

Read a block of data from the EEPROM - NOT FINISHED!!

Parameters:

start_address The start address of where we wish to read the data from

length The length of the data we wish read

data The memory area we wish to store the data to

Definition at line 215 of file eeprom_m24.c.

References EEPROM_M24_ADDR, i2cGetReceivedByte(), i2cReceiveByte(), i2cSendByte(), i2cSendStart(), i2cSendStop(), and i2cWaitForComplete().

6.46.2.2 unsigned char eeprom_m24_read_byte (unsigned int *eeprom_address*)

Read a byte of data from the EEPROM.

Parameters:

eeprom_address The address where we wish to read the byte from

Returns:

The value at eeprom_address

Definition at line 96 of file eeprom_m24.c.

References EEPROM_M24_ADDR, i2cGetReceivedByte(), i2cReceiveByte(), i2cSendByte(), i2cSendStart(), i2cSendStop(), and i2cWaitForComplete().

Referenced by eeprom_get_ant_sub_menu_array_structure(), eeprom_get_antenna_data(), eeprom_get_band_data(), eeprom_get_ptt_data(), eeprom_get_radio_settings_structure(), eeprom_get_runtime_settings(), eeprom_get_rx_antenna_data(), eeprom_get_settings_-structure(), and eeprom_read_table().

6.46.2.3 unsigned char eeprom_m24_write_block (unsigned int *start_address*, unsigned int *length*, unsigned char ∗ *data*)

Write a block of data to the EEPROM.

Parameters:

start_address The start address of where we wish to store the data

length The length of the data we wish to store

data The content we wish to write to the EEPROM

Definition at line 146 of file eeprom_m24.c.

References EEPROM_M24_ADDR, eeprom_tiny_delay(), i2cSendByte(), i2cSendStart(), i2cSendStop(), and i2cWaitForComplete().

Referenced by eeprom_create_table(), eeprom_save_ant_structure(), eeprom_save_ant_sub_-menu_array_structure(), eeprom_save_band_data(), eeprom_save_ptt_data(), eeprom_-save_radio_settings_structure(), eeprom_save_runtime_settings(), eeprom_save_rx_ant_-structure(), and eeprom_save_settings_structure().

6.46.2.4 unsigned char eeprom_m24_write_byte (unsigned int *eeprom_address*, unsigned char *value*)

Write a byte of data to the EEPROM.

Parameters:

eeprom_address The address where we wish to store the byte

value The value we wish to store at eeprom_address

Definition at line 43 of file eeprom_m24.c.

References EEPROM_M24_ADDR, i2cSendByte(), i2cSendStart(), i2cSendStop(), and i2cWaitForComplete().

6.46.2.5 void __inline__ eeprom_tiny_delay (void)

Just a tiny delay

Definition at line 35 of file eeprom_m24.c.

Referenced by eeprom_m24_write_block().

6.47 front_panel/eeprom_m24.h File Reference

EEPROM hardware functions.

Defines

- #define EEPROM_M24_ADDR 0xA0

  The address of the external EEPROM.

Functions

- unsigned char eeprom_m24_write_byte (unsigned int eeprom_address, unsigned char value)

  Write a byte of data to the EEPROM.

- unsigned char eeprom_m24_read_byte (unsigned int eeprom_address)

  Read a byte of data from the EEPROM.

- unsigned char eeprom_m24_write_block (unsigned int start_address, unsigned int length, unsigned char ∗data)

  Write a block of data to the EEPROM.

- unsigned char eeprom_m24_read_block (unsigned int start_address, unsigned int length, unsigned char ∗data)

  Read a block of data from the EEPROM - NOT FINISHED!!

6.47.1 Detailed Description

EEPROM hardware functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/eeprom_m24.h"
```

Definition in file eeprom_m24.h.

6.47.2 Function Documentation

6.47.2.1 unsigned char eeprom_m24_read_block (unsigned int *start_address*, unsigned int *length*, unsigned char ∗ *data*)

Read a block of data from the EEPROM - NOT FINISHED!!

Parameters:

start_ address The start address of where we wish to read the data from

length The length of the data we wish read

data The memory area we wish to store the data to

Definition at line 215 of file eeprom_m24.c.

References EEPROM_M24_ADDR, i2cGetReceivedByte(), i2cReceiveByte(), i2cSendByte(), i2cSendStart(), i2cSendStop(), and i2cWaitForComplete().

6.47.2.2 unsigned char eeprom_m24_read_byte (unsigned int *eeprom_ address*)

Read a byte of data from the EEPROM.

Parameters:

eeprom_ address The address where we wish to read the byte from

Returns:

The value at eeprom_address

Definition at line 96 of file eeprom_m24.c.

References EEPROM_M24_ADDR, i2cGetReceivedByte(), i2cReceiveByte(), i2cSendByte(), i2cSendStart(), i2cSendStop(), and i2cWaitForComplete().

Referenced by eeprom_get_ant_sub_menu_array_structure(), eeprom_get_antenna_data(), eeprom_get_band_data(), eeprom_get_ptt_data(), eeprom_get_radio_settings_structure(), eeprom_get_runtime_settings(), eeprom_get_rx_antenna_data(), eeprom_get_settings_-structure(), and eeprom_read_table().

6.47.2.3 unsigned char eeprom_m24_write_block (unsigned int *start_ address*, unsigned int *length*, unsigned char * *data*)

Write a block of data to the EEPROM.

Parameters:

start_ address The start address of where we wish to store the data

length The length of the data we wish to store

data The content we wish to write to the EEPROM

Definition at line 146 of file eeprom_m24.c.

References EEPROM_M24_ADDR, eeprom_tiny_delay(), i2cSendByte(), i2cSendStart(), i2cSendStop(), and i2cWaitForComplete().

Referenced by eeprom_create_table(), eeprom_save_ant_structure(), eeprom_save_ant_sub_-menu_array_structure(), eeprom_save_band_data(), eeprom_save_ptt_data(), eeprom_-save_radio_settings_structure(), eeprom_save_runtime_settings(), eeprom_save_rx_ant_-structure(), and eeprom_save_settings_structure().

6.47.2.4 unsigned char eeprom_m24_write_byte (unsigned int *eeprom_address*, unsigned char *value*)

Write a byte of data to the EEPROM.

Parameters:

eeprom_address The address where we wish to store the byte

value The value we wish to store at eeprom_address

Definition at line 43 of file eeprom_m24.c.

References EEPROM_M24_ADDR, i2cSendByte(), i2cSendStart(), i2cSendStop(), and i2cWaitForComplete().

6.48 front_panel/errors.h File Reference

List of error codes.

Defines

- #define NR_OF_ERRORS 5
  Define which tells us how many different error types that currently exist.
- #define ERROR_TYPE_BUS_RESEND 0
  Error that the bus had to resend a message more times than the max limit.
- #define ERROR_TYPE_BUS_SYNC 1
  Error that no sync was recieved within the default time frame.
- #define ERROR_TYPE_BUS_TX_QUEUE_FULL 2
  The TX queue of the bus has gotten full.
- #define ERROR_TYPE_BUS_RX_QUEUE_FULL 3
  The RX queue of the bus has gotten full.
- #define ERROR_TYPE_INT_COMM_RESEND 4
  Internal communication resend fail.

6.48.1 Detailed Description

List of error codes.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/errors.h"
```

Definition in file errors.h.

6.49 front_panel/event_handler.c File Reference

Event handler of various things.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include "event_handler.h"
#include "main.h"
#include "board.h"
#include "display.h"
#include "glcd.h"
#include "led_control.h"
#include "../delay.h"
#include "../i2c.h"
#include "../global.h"
#include "band_ctrl.h"
#include "antenna_ctrl.h"
#include "remote_control.h"
#include "eeprom_m24.h"
#include "rotary_encoder.h"
#include "menu.h"
#include "radio_interface.h"
#include "sequencer.h"
#include "interrupt_handler.h"
#include "../internal_comm.h"
#include "errors.h"
#include "sub_menu.h"
```

Functions

- void event_set_error (unsigned char error_type, unsigned char state)
  Set that an error has occured.
- unsigned char event_get_errors (void)
  Retrieve the state error flags.
- unsigned char event_get_error_state (unsigned char error_type)
  Retrieve the state of a specific error type.

- void event_internal_comm_parse_message (UC_MESSAGE message)

  Function which will parse the internal communication message.

- void __inline__ event_set_rx_antenna (unsigned char ant_index)

  Set an RX antenna. Will set the proper flags and call the antenna_ctrl_change_rx_ant function.

- void event_handler_process_ps2 (unsigned char key_code)

  Process an PS2 event.

- void event_pulse_sensor_up (void)

  The pulse sensor was turned up.

- void event_pulse_sensor_down (void)

  The pulse sensor was turned down.

- void event_update_display (void)

  Function to be called if we wish to update the display.

- void event_poll_buttons (void)

  Function which will poll all buttons and perform the proper action depending on their state.

- void event_poll_ext_device (void)

  Function which will poll the external devices and perform the proper actions depending on their state.

- void event_tx_button1_pressed (void)

  Perform the action of TX antenna button 1 if it was pressed.

- void event_tx_button2_pressed (void)

  Perform the action of TX antenna button 2 if it was pressed.

- void event_tx_button3_pressed (void)

  Perform the action of TX antenna button 3 if it was pressed.

- void event_tx_button4_pressed (void)

  Perform the action of TX antenna button 4 if it was pressed.

- void event_aux2_button_pressed (void)

  Perform the actions that should be done when AUX 2 button is pressed.

- void event_sub_button_pressed (void)

  Perform the actions that should be done when the SUB menu button is pressed.

- void event_rxant_button_pressed (void)

  Perform the action of RX antenna button if it was pressed.

- void event_rotate_button_pressed (void)

  Perform the action of Rotate button if it was pressed.

- void event_bus_parse_message (void)

  Parse a message from the communication bus.

- void event_parse_ext_event (unsigned int ext_event_status)

  Parse an external event and perform the proper action.

Variables

- x unsigned int main_flags

  Different flags, description is found in main.h.

- unsigned int flag_errors = 0

  Contains the errors which are set.

6.49.1 Detailed Description

Event handler of various things.

,

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/event_handler.c"
```

Definition in file event_handler.c.

6.49.2 Function Documentation

6.49.2.1 unsigned char event_get_error_state (unsigned char *error_type*)

Retrieve the state of a specific error type.

Parameters:

error_type Which kind of error we wish to check the state for

Returns:

The current state of this error

Definition at line 82 of file event_handler.c.

References flag_errors.

6.49.2.2 void event_handler_process_ps2 (unsigned char *key_code*)

Process an PS2 event.

Parameters:

key_code The key that was pressed

Definition at line 149 of file event_handler.c.

References event_rxant_button_pressed(), event_set_rx_antenna(), event_tx_button1_-pressed(), event_tx_button2_pressed(), event_tx_button3_pressed(), event_tx_button4_-pressed(), EXT_CTRL_SEL_NONE, EXT_CTRL_SEL_RX_ANT1, EXT_CTRL_SEL_-RX_ANT10, EXT_CTRL_TOGGLE_RX_ANT_MODE, EXT_CTRL_TOGGLE_TX_-ANT1, EXT_CTRL_TOGGLE_TX_ANT2, EXT_CTRL_TOGGLE_TX_ANT3, EXT_-CTRL_TOGGLE_TX_ANT4, ext_key_get_assignment(), KEYPAD_BTN_0, KEYPAD_-BTN_1, KEYPAD_BTN_2, KEYPAD_BTN_3, KEYPAD_BTN_4, KEYPAD_BTN_5, KEYPAD_BTN_6, KEYPAD_BTN_7, KEYPAD_BTN_8, KEYPAD_BTN_9, KEYPAD_-BTN_A, KEYPAD_BTN_B, KEYPAD_BTN_C, KEYPAD_BTN_D, KEYPAD_BTN_E, KEYPAD_BTN_F, and KEYPAD_BTN_G.

Referenced by event_internal_comm_parse_message().

6.49.2.3 void event_internal_comm_parse_message (UC_MESSAGE *message*)

Function which will parse the internal communication message.

Parameters:

message The message that we wish to parse

Definition at line 91 of file event_handler.c.

References band_ctrl_change_band(), UC_MESSAGE::cmd, computer_interface_is_-active(), struct_status::current_display, CURRENT_DISPLAY_SHUTDOWN_VIEW, UC_MESSAGE::data, display_shutdown_view(), event_add_message(), event_handler_-process_ps2(), INT_COMM_GET_BAND_BCD_STATUS, INT_COMM_PC_CTRL, INT_COMM_PS2_KEYPRESSED, INT_COMM_TURN_DEVICE_OFF, main_save_-settings(), radio_get_current_band(), radio_set_current_band(), remote_control_parse_-command(), send_ping(), shutdown_device(), and status.

Referenced by main().

6.49.2.4 void event_parse_ext_event (unsigned int *ext_event_status*)

Parse an external event and perform the proper action.

Parameters:

ext_event_status The status of the external "hardware" event flags

Definition at line 900 of file event_handler.c.

References struct_status::ext_devices_current_state, sequencer_computer_rts_activated(), sequencer_computer_rts_deactivated(), sequencer_footsw_pressed(), sequencer_footsw_-released(), sequencer_get_radio_sense(), sequencer_get_rts_polarity(), sequencer_get_-sense_polarity(), sequencer_radio_sense_activated(), sequencer_radio_sense_deactivated(), status, STATUS_FOOTSWITCH_BIT, STATUS_RADIO_SENSE1_BIT, STATUS_-RADIO_SENSE2_BIT, and STATUS_USB2_RTS_BIT.

Referenced by event_poll_ext_device().

6.49.2.5 void event_set_error (unsigned char *error_type*, unsigned char *state*)

Set that an error has occured.

Parameters:

error_type The type of error that has occured, defines can be found in errors.h

state State of the error

Definition at line 65 of file event_handler.c.

References flag_errors, and main_update_ptt_status().

Referenced by bus_resend_message(), internal_comm_resend(), ISR(), menu_action(), rx_-queue_add(), and tx_queue_add().

6.49.2.6 void __inline__ event_set_rx_antenna (unsigned char *ant_index*)

Set an RX antenna. Will set the proper flags and call the antenna_ctrl_change_rx_ant function.

Parameters:

ant_index The index of the RX antenna we wish to chose

Definition at line 140 of file event_handler.c.

References antenna_ctrl_change_rx_ant(), FLAG_UPDATE_DISPLAY, main_flags, struct_-status::selected_rx_antenna, and status.

Referenced by event_handler_process_ps2().

6.50 front_panel/event_handler.h File Reference

Event handler of various things.

```
#include "../wmv_bus/bus.h"
#include "../wmv_bus/bus_rx_queue.h"
#include "../wmv_bus/bus_tx_queue.h"
#include "../wmv_bus/bus_commands.h"
#include "../internal_comm.h"
#include "../internal_comm_commands.h"
```

Defines

- #define KEYPAD_BTN_1 0x69

  External keyboard keycode for Button 1.
- #define KEYPAD_BTN_2 0x72

  External keyboard keycode for Button 2.
- #define KEYPAD_BTN_3 0x7A

  External keyboard keycode for Button 3.
- #define KEYPAD_BTN_4 0x6B

  External keyboard keycode for Button 4.
- #define KEYPAD_BTN_5 0x73

  External keyboard keycode for Button 5.
- #define KEYPAD_BTN_6 0x74

  External keyboard keycode for Button 6.
- #define KEYPAD_BTN_7 0x6C

  External keyboard keycode for Button 7.
- #define KEYPAD_BTN_8 0x75

  External keyboard keycode for Button 8.
- #define KEYPAD_BTN_9 0x7D

  External keyboard keycode for Button 9.
- #define KEYPAD_BTN_0 0x70

  External keyboard keycode for Button 0.
- #define KEYPAD_BTN_A 0x77

  External keyboard keycode for Button A.
- #define KEYPAD_BTN_B 0x4A

  External keyboard keycode for Button B.

- #define KEYPAD_BTN_C 0x7C

  External keyboard keycode for Button C.

- #define KEYPAD_BTN_D 0x7B

  External keyboard keycode for Button D.

- #define KEYPAD_BTN_E 0x79

  External keyboard keycode for Button E.

- #define KEYPAD_BTN_F 0x5A

  External keyboard keycode for Button F.

- #define KEYPAD_BTN_G 0x71

  External keyboard keycode for Button G.

- #define EXT_CTRL_SEL_NONE 0

  Ext ctrl - No function assigned.

- #define EXT_CTRL_SEL_RX_ANT1 1

  Ext ctrl - Set RX antenna #1.

- #define EXT_CTRL_SEL_RX_ANT2 2

  Ext ctrl - Set RX antenna #2.

- #define EXT_CTRL_SEL_RX_ANT3 3

  Ext ctrl - Set RX antenna #3.

- #define EXT_CTRL_SEL_RX_ANT4 4

  Ext ctrl - Set RX antenna #4.

- #define EXT_CTRL_SEL_RX_ANT5 5

  Ext ctrl - Set RX antenna #5.

- #define EXT_CTRL_SEL_RX_ANT6 6

  Ext ctrl - Set RX antenna #6.

- #define EXT_CTRL_SEL_RX_ANT7 7

  Ext ctrl - Set RX antenna #7.

- #define EXT_CTRL_SEL_RX_ANT8 8

  Ext ctrl - Set RX antenna #8.

- #define EXT_CTRL_SEL_RX_ANT9 9

  Ext ctrl - Set RX antenna #9.

- #define EXT_CTRL_SEL_RX_ANT10 10

  Ext ctrl - Set RX antenna #10.

- #define EXT_CTRL_TOGGLE_TX_ANT1 9

Ext ctrl - Toggle TX antenna combination #1.

- #define EXT_CTRL_TOGGLE_TX_ANT2 10

  Ext ctrl - Toggle TX antenna combination #2.
- #define EXT_CTRL_TOGGLE_TX_ANT3 11

  Ext ctrl - Toggle TX antenna combination #3.
- #define EXT_CTRL_TOGGLE_TX_ANT4 12

  Ext ctrl - Toggle TX antenna combination #4.
- #define EXT_CTRL_TOGGLE_RX_ANT1 13

  Ext ctrl - Toggle RX antenna combination #1.
- #define EXT_CTRL_TOGGLE_RX_ANT2 14

  Ext ctrl - Toggle RX antenna combination #2.
- #define EXT_CTRL_TOGGLE_RX_ANT3 15

  Ext ctrl - Toggle RX antenna combination #3.
- #define EXT_CTRL_TOGGLE_RX_ANT4 16

  Ext ctrl - Toggle RX antenna combination #4.
- #define EXT_CTRL_TOGGLE_RX_ANT_MODE 17

  Ext ctrl - Toggle RX antenna enabled.
- #define EXT_CTRL_TOGGLE_TXRX_MODE 18

  Ext ctrl - Toggle TX/RX mode on/off.
- #define EXT_CTRL_SET_ARRAY_DIR1 19

  Ext ctrl - Select array direction #1.
- #define EXT_CTRL_SET_ARRAY_DIR2 20

  Ext ctrl - Select array direction #2.
- #define EXT_CTRL_SET_ARRAY_DIR3 21

  Ext ctrl - Select array direction #3.
- #define EXT_CTRL_SET_ARRAY_DIR4 22

  Ext ctrl - Select array direction #4.
- #define EXT_CTRL_SET_ARRAY_DIR5 23

  Ext ctrl - Select array direction #5.
- #define EXT_CTRL_SET_ARRAY_DIR6 24

  Ext ctrl - Select array direction #6.
- #define EXT_CTRL_SET_ARRAY_DIR7 25

  Ext ctrl - Select array direction #7.

- #define EXT_CTRL_SET_ARRAY_DIR8 26

  Ext ctrl - Select array direction #8.

- #define EXT_CTRL_SET_STACK_COMB1 27

  Ext ctrl - Select stack combo #1.

- #define EXT_CTRL_SET_STACK_COMB2 28

  Ext ctrl - Select stack combo #2.

- #define EXT_CTRL_SET_STACK_COMB3 29

  Ext ctrl - Select stack combo #3.

- #define EXT_CTRL_SET_STACK_COMB4 30

  Ext ctrl - Select stack combo #4.

- #define EXT_CTRL_SET_STACK_COMB5 31

  Ext ctrl - Select stack combo #5.

- #define EXT_CTRL_SET_STACK_COMB6 32

  Ext ctrl - Select stack combo #6.

- #define EXT_CTRL_AMPLIFIER_TOGGLE_ON_OFF 33

  Ext ctrl - Toggle the amplifier on/off.

- #define EXT_CTRL_AMPLIFIER_TOGGLE_STANDBY 34

  Ext ctrl - Toggle the amplifier standby.

- #define EXT_CTRL_AMPLIFIER_TUNE 35

  Ext ctrl - Tune the amplifier to the correct band.

- #define EXT_CTRL_AMPLIFIER_RESET 36

  Ext ctrl - Reset the amplifier.

Functions

- void event_set_error (unsigned char error_type, unsigned char state)

  Set that an error has occured.

- unsigned char event_get_errors (void)

  Retrieve the state error flags.

- unsigned char event_get_error_state (unsigned char error_type)

  Retrieve the state of a specific error type.

- void event_internal_comm_parse_message (UC_MESSAGE message)

  Function which will parse the internal communication message.

- void event_handler_process_ps2 (unsigned char key_code)

Process an PS2 event.

- void event_pulse_sensor_up (void)
  The pulse sensor was turned up.
- void event_pulse_sensor_down (void)
  The pulse sensor was turned down.
- void event_update_display (void)
  Function to be called if we wish to update the display.
- void event_poll_buttons (void)
  Function which will poll all buttons and perform the proper action depending on their state.
- void event_poll_ext_device (void)
  Function which will poll the external devices and perform the proper actions depending on their state.
- void event_bus_parse_message (void)
  Parse a message from the communication bus.
- void event_parse_ext_event (unsigned int ext_event_status)
  Parse an external event and perform the proper action.
- void event_sub_button_pressed (void)
  Perform the actions that should be done when the SUB menu button is pressed.
- void event_tx_button1_pressed (void)
  Perform the action of TX antenna button 1 if it was pressed.
- void event_tx_button2_pressed (void)
  Perform the action of TX antenna button 2 if it was pressed.
- void event_tx_button3_pressed (void)
  Perform the action of TX antenna button 3 if it was pressed.
- void event_tx_button4_pressed (void)
  Perform the action of TX antenna button 4 if it was pressed.
- void event_rotate_button_pressed (void)
  Perform the action of Rotate button if it was pressed.
- void event_rxant_button_pressed (void)
  Perform the action of RX antenna button if it was pressed.
- void event_aux2_button_pressed (void)
  Perform the actions that should be done when AUX 2 button is pressed.
- void __inline__ event_set_rx_antenna (unsigned char ant_index)
  Set an RX antenna. Will set the proper flags and call the antenna_ctrl_change_rx_ant function.

6.50.1 Detailed Description

Event handler of various things.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/event_handler.h"
```

Definition in file event_handler.h.

6.50.2 Function Documentation

6.50.2.1 unsigned char event_get_error_state (unsigned char *error_type*)

Retrieve the state of a specific error type.

Parameters:

error_type Which kind of error we wish to check the state for

Returns:

The current state of this error

Definition at line 82 of file event_handler.c.

References flag_errors.

6.50.2.2 void event_handler_process_ps2 (unsigned char *key_code*)

Process an PS2 event.

Parameters:

key_code The key that was pressed

Definition at line 149 of file event_handler.c.

References event_rxant_button_pressed(), event_set_rx_antenna(), event_tx_button1_-pressed(), event_tx_button2_pressed(), event_tx_button3_pressed(), event_tx_button4_-pressed(), EXT_CTRL_SEL_NONE, EXT_CTRL_SEL_RX_ANT1, EXT_CTRL_SEL_-RX_ANT10, EXT_CTRL_TOGGLE_RX_ANT_MODE, EXT_CTRL_TOGGLE_TX_-ANT1, EXT_CTRL_TOGGLE_TX_ANT2, EXT_CTRL_TOGGLE_TX_ANT3, EXT_-CTRL_TOGGLE_TX_ANT4, ext_key_get_assignment(), KEYPAD_BTN_0, KEYPAD_-BTN_1, KEYPAD_BTN_2, KEYPAD_BTN_3, KEYPAD_BTN_4, KEYPAD_BTN_5, KEYPAD_BTN_6, KEYPAD_BTN_7, KEYPAD_BTN_8, KEYPAD_BTN_9, KEYPAD_-BTN_A, KEYPAD_BTN_B, KEYPAD_BTN_C, KEYPAD_BTN_D, KEYPAD_BTN_E, KEYPAD_BTN_F, and KEYPAD_BTN_G.

Referenced by event_internal_comm_parse_message().

6.50.2.3 void event_internal_comm_parse_message (UC_MESSAGE *message*)

Function which will parse the internal communication message.

Parameters:

message The message that we wish to parse

Definition at line 91 of file event_handler.c.

References band_ctrl_change_band(), UC_MESSAGE::cmd, computer_interface_is_-active(), struct_status::current_display, CURRENT_DISPLAY_SHUTDOWN_VIEW, UC_MESSAGE::data, display_shutdown_view(), event_add_message(), event_handler_-process_ps2(), INT_COMM_GET_BAND_BCD_STATUS, INT_COMM_PC_CTRL, INT_COMM_PS2_KEYPRESSED, INT_COMM_TURN_DEVICE_OFF, main_save_-settings(), radio_get_current_band(), radio_set_current_band(), remote_control_parse_-command(), send_ping(), shutdown_device(), and status.

Referenced by main().

6.50.2.4 void event_parse_ext_event (unsigned int *ext_event_status*)

Parse an external event and perform the proper action.

Parameters:

ext_event_status The status of the external "hardware" event flags

Definition at line 900 of file event_handler.c.

References struct_status::ext_devices_current_state, sequencer_computer_rts_activated(), sequencer_computer_rts_deactivated(), sequencer_footsw_pressed(), sequencer_footsw_-released(), sequencer_get_radio_sense(), sequencer_get_rts_polarity(), sequencer_get_-sense_polarity(), sequencer_radio_sense_activated(), sequencer_radio_sense_deactivated(), status, STATUS_FOOTSWITCH_BIT, STATUS_RADIO_SENSE1_BIT, STATUS_-RADIO_SENSE2_BIT, and STATUS_USB2_RTS_BIT.

Referenced by event_poll_ext_device().

6.50.2.5 void event_set_error (unsigned char *error_type*, unsigned char *state*)

Set that an error has occured.

Parameters:

error_type The type of error that has occured, defines can be found in errors.h

state State of the error

Definition at line 65 of file event_handler.c.

References flag_errors, and main_update_ptt_status().

Referenced by bus_resend_message(), internal_comm_resend(), ISR(), menu_action(), rx_-queue_add(), and tx_queue_add().

6.50.2.6 void __inline__ event_set_rx_antenna (unsigned char *ant_index*)

Set an RX antenna. Will set the proper flags and call the antenna_ctrl_change_rx_ant function.

Parameters:

ant_index The index of the RX antenna we wish to chose

Definition at line 140 of file event_handler.c.

References antenna_ctrl_change_rx_ant(), FLAG_UPDATE_DISPLAY, main_flags, struct_-status::selected_rx_antenna, and status.

Referenced by event_handler_process_ps2().

6.51 front_panel/glcd.c File Reference

Graphic LCD API functions.

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "glcd.h"
#include "fonts.h"
#include "ks0108.h"
#include "pictures.h"
#include <string.h>
#include <stdio.h>
```

Functions

- void **glcd_update_area** (unsigned char x1, unsigned char x2, unsigned char y1, unsigned char y2)
- void **glcd_update** (unsigned int top, unsigned int bottom)
- void **glcd_glyph** (unsigned char left, unsigned char top, unsigned char width, unsigned char height, const prog_char *glyph, unsigned char store_width)
- void **glcd_set_byte** (unsigned char x, unsigned char y, unsigned char curr_byte)
- void **glcd_text** (unsigned char left, unsigned char top, unsigned char font, char *str, unsigned char length)
- void **glcd_clear_area** (unsigned char x1, unsigned char x2, unsigned char y1, unsigned char y2)
- void glcd_set_dot (unsigned char x, unsigned char y, unsigned char mode)

  set a dot on the display (x is horiz 0:127, y is vert 0:63)

- void **glcd_invert_area** (unsigned char x1, unsigned char x2, unsigned char y1, unsigned char y2)
- void glcd_line (unsigned char x1, unsigned char x2, unsigned char y)

  draw line

- void glcd_rectangle (unsigned char x, unsigned char y, unsigned char a, unsigned char b)

  draw rectangle (coords????)

- void glcd_circle (unsigned char xcenter, unsigned char ycenter, unsigned char radius)

  draw circle of radius at xcenter,ycenter

- void **glcd_invert** ()
- void **glcd_clear** (void)
- void **glcd_print_picture** (void)

Variables

- unsigned char **rxed_data**

6.51.1 Detailed Description

Graphic LCD API functions.

Definition in file glcd.c.

6.52 front_panel/glcd.h File Reference

Graphic LCD API functions.

`#include <avr/io.h>`

`#include "../global.h"`

Defines

- #define GLCD_LEFT 0
- #define **GLCD_TOP** 0
- #define **GLCD_RIGHT** 128
- #define **GLCD_BOTTOM** 64
- #define **GLCD_Y_BYTES** 8
- #define **GLCD_X_BYTES** 128
- #define **GLCD_MAXPAGE** 8
- #define **GLCD_MAXADDRESS** 64
- #define **glcd_update_all**() glcd_update(GLCD_TOP, GLCD_BOTTOM);
- #define **GLCD_MODE_CLEAR** 0
- #define **GLCD_MODE_SET** 1
- #define **GLCD_MODE_XOR** 2
- #define **LINE1** 0
- #define **LINE2** 1
- #define **LINE3** 2
- #define **LINE4** 3
- #define **LINE5** 4
- #define **LINE6** 5
- #define **LINE7** 6
- #define **LINE8** 7
- #define **ON** 1
- #define **OFF** 0

Functions

- void glcd_set_dot (unsigned char x, unsigned char y, unsigned char mode)

  set a dot on the display (x is horiz 0:127, y is vert 0:63)

- void glcd_line (unsigned char x1, unsigned char x2, unsigned char y)

  draw line

- void glcd_rectangle (unsigned char x, unsigned char y, unsigned char a, unsigned char b)

  draw rectangle (coords????)

- void glcd_circle (unsigned char xcenter, unsigned char ycenter, unsigned char radius)

  draw circle of radius at xcenter,ycenter

- void **glcd_print_picture** (void)
- void **glcd_invert_area** (unsigned char x1, unsigned char x2, unsigned char y1, unsigned char y2)

- void **glcd_update** (unsigned int top, unsigned int bottom)
- void **glcd_text** (unsigned char left, unsigned char top, unsigned char font, char *str, unsigned char length)
- void **glcd_invert** (void)
- void **glcd_clear** (void)
- void **glcd_update_area** (unsigned char x1, unsigned char x2, unsigned char y1, unsigned char y2)
- void **glcd_clear_area** (unsigned char x1, unsigned char x2, unsigned char y1, unsigned char y2)
- void **glcd_set_byte** (unsigned char x, unsigned char y, unsigned char curr_byte)

6.52.1 Detailed Description

Graphic LCD API functions.

Definition in file glcd.h.

6.53 front_panel/interrupt_handler.c File Reference

Handles different external interrupts.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "main.h"
#include "board.h"
```

Functions

- int ih_poll_buttons (void)
- unsigned char ih_poll_ext_devices (void)

6.53.1 Detailed Description

Handles different external interrupts.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/interrupt_handler.c"
```

Definition in file interrupt_handler.c.

6.53.2 Function Documentation

6.53.2.1 int ih_poll_buttons (void)

Polls the status of all buttons on the front panel and returns it as an integer. The bit mask pattern is defined in board.h

Returns:

Which buttons that are currently pressed, see mask pattern in board.h

The following is done because of a hardware bug! The pullups on the uC are too small to actually charge the debounce capacitor in time. The way we solve it is that by making the pin to an output we can charge the capacitor and then go over to using the pin as input and reactivate the pullups again.

Definition at line 37 of file interrupt_handler.c.

References BUTTON1_RX_BIT, BUTTON1_TX_BIT, BUTTON2_RX_BIT, BUTTON2_-TX_BIT, BUTTON3_RX_BIT, BUTTON3_TX_BIT, BUTTON4_RX_BIT, BUTTON4_-TX_BIT, BUTTON_AUX1_BIT, BUTTON_AUX2_BIT, BUTTON_MENU_BIT, BUTTON_PULSE_BIT, BUTTON_ROTATE_BIT, BUTTON_RXANT_BIT, BUTTON_-SUBMENU_BIT, BUTTON_TXRX_BIT, FLAG_BUTTON1_RX_BIT, FLAG_-BUTTON1_TX_BIT, FLAG_BUTTON2_RX_BIT, FLAG_BUTTON2_TX_BIT, FLAG_-BUTTON3_RX_BIT, FLAG_BUTTON3_TX_BIT, FLAG_BUTTON4_RX_BIT, FLAG_-BUTTON4_TX_BIT, FLAG_BUTTON_AUX1_BIT, FLAG_BUTTON_AUX2_BIT, FLAG_BUTTON_MENU_BIT, FLAG_BUTTON_PULSE_BIT, FLAG_BUTTON_-ROTATE_BIT, FLAG_BUTTON_RXANT_BIT, FLAG_BUTTON_SUBMENU_BIT, and FLAG_BUTTON_TXRX_BIT.

Referenced by event_poll_buttons().

6.53.2.2 unsigned char ih_poll_ext_devices (void)

Polls the status of all the external inputs. This function does not care if the device is active low or active high. It will just return the current state so the handeling of leveling needs to be done elsewhere. The bit mask pattern is defined in board.h

Returns:

The status of the external devices

Definition at line 108 of file interrupt_handler.c.

References EXT_FOOTSWITCH_BIT, EXT_RADIO_SENSE1_BIT, EXT_RADIO_-SENSE2_BIT, EXT_USB1_DTR_BIT, EXT_USB2_DTR_BIT, EXT_USB2_RTS_BIT, STATUS_FOOTSWITCH_BIT, STATUS_RADIO_SENSE1_BIT, STATUS_RADIO_-SENSE2_BIT, STATUS_USB1_DTR_BIT, STATUS_USB2_DTR_BIT, and STATUS_-USB2_RTS_BIT.

Referenced by event_poll_ext_device(), and main().

6.54 front_panel/interrupt_handler.h File Reference

Handles different external interrupts.

Functions

- int ih_poll_buttons (void)
- unsigned char ih_poll_ext_devices (void)

6.54.1 Detailed Description

Handles different external interrupts.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/interrupt_handler.h"
```

Definition in file interrupt_handler.h.

6.54.2 Function Documentation

6.54.2.1 int ih_poll_buttons (void)

Polls the status of all buttons on the front panel and returns it as an integer. The bit mask pattern is defined in board.h

Returns:

Which buttons that are currently pressed, see mask pattern in board.h

The following is done because of a hardware bug! The pullups on the uC are too small to actually charge the debounce capacitor in time. The way we solve it is that by making the pin to an output we can charge the capacitor and then go over to using the pin as input and reactivate the pullups again.

Definition at line 37 of file interrupt_handler.c.

References BUTTON1_RX_BIT, BUTTON1_TX_BIT, BUTTON2_RX_BIT, BUTTON2_-TX_BIT, BUTTON3_RX_BIT, BUTTON3_TX_BIT, BUTTON4_RX_BIT, BUTTON4_-TX_BIT, BUTTON_AUX1_BIT, BUTTON_AUX2_BIT, BUTTON_MENU_BIT, BUTTON_PULSE_BIT, BUTTON_ROTATE_BIT, BUTTON_RXANT_BIT, BUTTON_-SUBMENU_BIT, BUTTON_TXRX_BIT, FLAG_BUTTON1_RX_BIT, FLAG_-BUTTON1_TX_BIT, FLAG_BUTTON2_RX_BIT, FLAG_BUTTON2_TX_BIT, FLAG_-BUTTON3_RX_BIT, FLAG_BUTTON3_TX_BIT, FLAG_BUTTON4_RX_BIT, FLAG_-BUTTON4_TX_BIT, FLAG_BUTTON_AUX1_BIT, FLAG_BUTTON_AUX2_BIT, FLAG_BUTTON_MENU_BIT, FLAG_BUTTON_PULSE_BIT, FLAG_BUTTON_-ROTATE_BIT, FLAG_BUTTON_RXANT_BIT, FLAG_BUTTON_SUBMENU_BIT, and FLAG_BUTTON_TXRX_BIT.

Referenced by event_poll_buttons().

6.54.2.2 unsigned char ih_poll_ext_devices (void)

Polls the status of all the external inputs. This function does not care if the device is active low or active high. It will just return the current state so the handeling of leveling needs to be done elsewhere. The bit mask pattern is defined in board.h

Returns:

The status of the external devices

Definition at line 108 of file interrupt_handler.c.

References EXT_FOOTSWITCH_BIT, EXT_RADIO_SENSE1_BIT, EXT_RADIO_-SENSE2_BIT, EXT_USB1_DTR_BIT, EXT_USB2_DTR_BIT, EXT_USB2_RTS_BIT, STATUS_FOOTSWITCH_BIT, STATUS_RADIO_SENSE1_BIT, STATUS_RADIO_-SENSE2_BIT, STATUS_USB1_DTR_BIT, STATUS_USB2_DTR_BIT, and STATUS_-USB2_RTS_BIT.

Referenced by event_poll_ext_device(), and main().

6.55 front_panel/ks0108.c File Reference

Graphic LCD driver for HD61202/KS0108 displays.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "../global.h"
#include "ks0108.h"
#include "glcd.h"
```

Functions

- void **glcd_init_hw** (void)
- void **glcd_controller_select** (u08 controller)
- void **glcd_busy_wait** (u08 controller)
- void **glcd_control_write** (u08 controller, u08 data)
- u08 **glcd_control_read** (u08 controller)
- void **glcd_data_write** (u08 data)
- u08 **glcd_data_read** (void)
- void **glcd_reset** (u08 reset_state)
- u08 **glcd_get_x_address** ()
- u08 **glcd_get_y_address** ()
- void **glcd_set_x_address** (u08 xAddr)
- void **glcd_set_y_address** (u08 yAddr)
- void glcd_init ()

  Initialize the display, clear it, and prepare it for access.

- void glcd_home (void)

  Set display memory access point back to upper,left corner.

- void glcd_clear_screen (void)

  Clear the display.

- void glcd_start_line (u08 start)

  Set display memory access point to row [line] and column [col] assuming 5x7 font.

- void glcd_set_address (u08 x, u08 yLine)

  Set display memory access point to [x] horizontal pixel and [y] vertical line.

- void glcd_goto_char (u08 line, u08 col)

  Set display memory access point to row [line] and column [col] assuming 5x7 font.

- void glcd_delay (u16 p)

  Generic delay routine for timed glcd access.

Variables

- GrLcdStateType **GrLcdState**

6.55.1 Detailed Description

Graphic LCD driver for HD61202/KS0108 displays.

Definition in file ks0108.c.

6.56 front_panel/ks0108.h File Reference

Graphic LCD driver for HD61202/KS0108 displays.

`#include "../global.h"`

`#include "ks0108conf.h"`

Classes

- struct **struct_GrLcdCtrlrStateType**
- struct **struct_GrLcdStateType**

Defines

- #define GLCD_ON_CTRL 0x3E
- #define **GLCD_ON_DISPLAY** 0x01
- #define **GLCD_START_LINE** 0xC0
- #define **GLCD_SET_PAGE** 0xB8
- #define **GLCD_SET_Y_ADDR** 0x40
- #define **GLCD_STATUS_BUSY** 0x80
- #define **GLCD_STATUS_ONOFF** 0x20
- #define **GLCD_STATUS_RESET** 0x10
- #define **GLCD_NUM_CONTROLLERS** 2

Typedefs

- typedef struct struct_GrLcdCtrlrStateType **GrLcdCtrlrStateType**
- typedef struct struct_GrLcdStateType **GrLcdStateType**

Functions

- void **glcd_init_hw** (void)
- void **glcd_busy_wait** (u08 controller)
- void **glcd_control_write** (u08 controller, u08 data)
- u08 **glcd_control_read** (u08 controller)
- void **glcd_data_write** (u08 data)
- u08 **glcd_data_read** (void)
- void **glcd_set_x_address** (u08 xAddr)
- void **glcd_set_y_address** (u08 yAddr)
- u08 **glcd_get_x_address** (void)
- u08 **glcd_get_y_address** (void)
- u08 **get_data_port** (void)
- void glcd_init (void)

  Initialize the display, clear it, and prepare it for access.
- void glcd_clear_screen (void)

  Clear the display.
- void glcd_home (void)

Set display memory access point back to upper,left corner.

- void glcd_goto_char (u08 line, u08 col)

  Set display memory access point to row [line] and column [col] assuming 5x7 font.

- void glcd_set_address (u08 x, u08 yLine)

  Set display memory access point to [x] horizontal pixel and [y] vertical line.

- void glcd_start_line (u08 start)

  Set display memory access point to row [line] and column [col] assuming 5x7 font.

- void glcd_delay (u16 p)

  Generic delay routine for timed glcd access.

6.56.1 Detailed Description

Graphic LCD driver for HD61202/KS0108 displays.

Definition in file ks0108.h.

6.57 front_panel/ks0108conf.h File Reference

Graphic LCD driver configuration.

Defines

- #define **GLCD_PORT_INTERFACE**
- #define **GLCD_CTRL_PORT** PORTK
- #define **GLCD_CTRL_DDR** DDRK
- #define **GLCD_CTRL_RS** PK4
- #define **GLCD_CTRL_RW** PK3
- #define **GLCD_CTRL_E** PK5
- #define **GLCD_CTRL_CS0** PK1
- #define **GLCD_CTRL_CS1** PK0
- #define **GLCD_CTRL_CS2** PA1
- #define **GLCD_CTRL_CS3** PA0
- #define **GLCD_CTRL_RESET** PK2
- #define **GLCD_DATA_PORT** PORTF
- #define **GLCD_DATA_DDR** DDRF
- #define **GLCD_DATA_PIN** PINF
- #define **GLCD_XPIXELS** 128
- #define **GLCD_YPIXELS** 64
- #define **GLCD_CONTROLLER_XPIXELS** 64
- #define **GLCD_TEXT_LINES** 8
- #define **GLCD_TEXT_LINE_LENGTH** 22

6.57.1 Detailed Description

Graphic LCD driver configuration.

Definition in file ks0108conf.h.

6.58 front_panel/led_control.c File Reference

Front panel LED control functions.

```
#include <stdio.h>
#include <avr/io.h>
#include "led_control.h"
#include "board.h"
#include "../global.h"
```

Functions

- void led_set_band (unsigned char band)
  Set the band LEDs to the proper band.
- void led_set_band_none (void)
  Turn off all band leds.
- void led_set_ptt (enum enum_led_ptt_state state)
  Set the PTT LED.
- void led_set_error (enum enum_led_state state)
  Set the error LED status.
- void led_set_rotation_active (enum enum_led_state state)
  Set the rotating led to active state, indicates if any antenna on the current band is rotating.
- void led_set_tx_ant (unsigned char index, enum enum_led_state state)
  Set the TX Antenna LED status.
- void led_set_rx_ant (unsigned char index, enum enum_led_state state)
  Set the RX Antenna LED status.
- void led_set_rotate (enum enum_led_state state)
  Set the Rotate LED status.
- void led_set_txrx (enum enum_led_state state)
  Set the TX/RX mode LED status.
- void led_set_rxant (enum enum_led_state state)
  Set the RX antenna LED status.
- void led_set_aux (enum enum_led_state state)
  Set the AUX LED status.
- void led_set_submenu (enum enum_led_state state)
  Set the AUX LED status.

- void led_set_menu (enum enum_led_state state)
  Set the menu LED status.

- void led_set_all (enum enum_led_state state)
  Set all the LEDs.

6.58.1 Detailed Description

Front panel LED control functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/led_control.c"
```

Definition in file led_control.c.

6.58.2 Function Documentation

6.58.2.1 void led_set_all (enum enum_led_state *state*)

Set all the LEDs.

Parameters:

state The state of the LED

Definition at line 225 of file led_control.c.

References led_set_aux(), led_set_band(), led_set_error(), led_set_menu(), led_set_ptt(), led_set_rotate(), led_set_rx_ant(), led_set_rxant(), led_set_tx_ant(), led_set_txrx(), LED_STATE_OFF, LED_STATE_ON, LED_STATE_PTT_ACTIVE, and LED_STATE_-PTT_OK.

Referenced by main().

6.58.2.2 void led_set_aux (enum enum_led_state *state*)

Set the AUX LED status.

Parameters:

state The state of the LED

Definition at line 198 of file led_control.c.

References LED_AUX_BIT, and LED_STATE_ON.

Referenced by led_set_all().

6.58.2.3 void led_set_band (unsigned char *band*)

Set the band LEDs to the proper band.

Parameters:

band The band we wish to turn on the LED for

Definition at line 32 of file led_control.c.

Referenced by band_ctrl_change_band(), led_set_all(), and main().

6.58.2.4 void led_set_error (enum enum_led_state *state*)

Set the error LED status.

Parameters:

state The state of the LED

Definition at line 67 of file led_control.c.

References LED_ERROR_BIT, and LED_STATE_ON.

Referenced by bus_resend_message(), internal_comm_resend(), ISR(), led_set_all(), menu_-action(), rx_queue_add(), shutdown_device(), and tx_queue_add().

6.58.2.5 void led_set_menu (enum enum_led_state *state*)

Set the menu LED status.

Parameters:

state The state of the LED

Definition at line 216 of file led_control.c.

References LED_MENU_BIT, and LED_STATE_ON.

Referenced by event_poll_buttons(), and led_set_all().

6.58.2.6 void led_set_ptt (enum enum_led_ptt_state *state*)

Set the PTT LED.

Parameters:

state The state of the LED

Definition at line 47 of file led_control.c.

References LED_PTT_GREEN_BIT, LED_PTT_RED_BIT, LED_STATE_PTT_ACTIVE, LED_STATE_PTT_INHIBIT, and LED_STATE_PTT_OK.

Referenced by led_set_all(), main(), and main_update_ptt_status().

6.58.2.7 void led_set_rotate (enum enum_led_state *state*)

Set the Rotate LED status.

Parameters:

state The state of the LED

Definition at line 169 of file led_control.c.

References LED_ROTATE_BIT, and LED_STATE_ON.

Referenced by event_poll_buttons(), event_rotate_button_pressed(), event_rxant_button_-pressed(), and led_set_all().

6.58.2.8 void led_set_rotation_active (enum enum_led_state *state*)

Set the rotating led to active state, indicates if any antenna on the current band is rotating.

Parameters:

state The state of the LED

Definition at line 76 of file led_control.c.

References LED_ROTATION_ACTIVE_BIT, and LED_STATE_ON.

Referenced by ISR().

6.58.2.9 void led_set_rx_ant (unsigned char *index*, enum enum_led_state *state*)

Set the RX Antenna LED status.

Parameters:

index Which LED we wish to change the status of
state The state of the LED

Definition at line 128 of file led_control.c.

References LED_RX_BUTTON1_BIT, LED_RX_BUTTON2_BIT, LED_RX_BUTTON3_-BIT, LED_RX_BUTTON4_BIT, and LED_STATE_ON.

Referenced by band_ctrl_change_band(), and led_set_all().

6.58.2.10 void led_set_rxant (enum enum_led_state *state*)

Set the RX antenna LED status.

Parameters:

state The state of the LED

Definition at line 187 of file led_control.c.

References LED_RXANT_BIT, and LED_STATE_ON.

Referenced by band_ctrl_change_band(), event_rxant_button_pressed(), and led_set_all().

6.58.2.11 void led_set_submenu (enum enum_led_state *state*)

Set the AUX LED status.

Parameters:

state The state of the LED

Definition at line 207 of file led_control.c.

References LED_STATE_ON, and LED_SUBMENU_BIT.

Referenced by event_poll_buttons(), and event_sub_button_pressed().

6.58.2.12 void led_set_tx_ant (unsigned char *index*, enum enum_led_state *state*)

Set the TX Antenna LED status.

Parameters:

index Which LED we wish to change the status of

state The state of the LED

Definition at line 86 of file led_control.c.

References LED_STATE_ON, LED_TX_BUTTON1_BIT, LED_TX_BUTTON2_BIT, LED_TX_BUTTON3_BIT, and LED_TX_BUTTON4_BIT.

Referenced by band_ctrl_change_band(), event_tx_button1_pressed(), event_tx_button2_-pressed(), event_tx_button3_pressed(), event_tx_button4_pressed(), ISR(), led_set_all(), and set_tx_ant_leds().

6.58.2.13 void led_set_txrx (enum enum_led_state *state*)

Set the TX/RX mode LED status.

Parameters:

state The state of the LED

Definition at line 178 of file led_control.c.

References LED_STATE_ON, and LED_TXRX_BIT.

Referenced by led_set_all().

6.59 front_panel/led_control.h File Reference

Front panel LED control functions.

Enumerations

- enum enum_led_ptt_state { LED_STATE_PTT_ACTIVE, LED_STATE_PTT_-INHIBIT, LED_STATE_PTT_OK }

  PTT led state.

- enum enum_led_state { LED_STATE_ON, LED_STATE_OFF, **LED_STATE_ON**, **LED_STATE_OFF** }

  Regular LED state.

Functions

- void led_set_band (unsigned char band)

  Set the band LEDs to the proper band.

- void led_set_band_none (void)

  Turn off all band leds.

- void led_set_ptt (enum enum_led_ptt_state state)

  Set the PTT LED.

- void led_set_error (enum enum_led_state state)

  Set the error LED status.

- void led_set_rotation_active (enum enum_led_state state)

  Set the rotating led to active state, indicates if any antenna on the current band is rotating.

- void led_set_tx_ant (unsigned char index, enum enum_led_state state)

  Set the TX Antenna LED status.

- void led_set_rx_ant (unsigned char index, enum enum_led_state state)

  Set the RX Antenna LED status.

- void led_set_rotate (enum enum_led_state state)

  Set the Rotate LED status.

- void led_set_txrx (enum enum_led_state state)

  Set the TX/RX mode LED status.

- void led_set_rxant (enum enum_led_state state)

  Set the RX antenna LED status.

- void led_set_aux (enum enum_led_state state)

  Set the AUX LED status.

- void led_set_menu (enum enum_led_state state)

  Set the menu LED status.

- void led_set_submenu (enum enum_led_state state)

  Set the AUX LED status.

- void led_set_all (enum enum_led_state state)

  Set all the LEDs.

6.59.1 Detailed Description

Front panel LED control functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/led_control.h"
```

Definition in file led_control.h.

6.59.2 Enumeration Type Documentation

6.59.2.1 enum enum_led_ptt_state

PTT led state.

Enumerator:

LED_STATE_PTT_ACTIVE Used to set the LED to PTT active color.

LED_STATE_PTT_INHIBIT Used to set the LED to INHIBIT color.

LED_STATE_PTT_OK Used to set the LED to PTT OK color (which means that it is OK to transmit).

Definition at line 27 of file led_control.h.

6.59.2.2 enum enum_led_state

Regular LED state.

Enumerator:

LED_STATE_ON Used to set the LED as ON.

LED_STATE_OFF Used to set the LED as OFF.

Definition at line 37 of file led_control.h.

6.59.3 Function Documentation

6.59.3.1 void led_set_all (enum enum_led_state *state*)

Set all the LEDs.

Parameters:

state The state of the LED

Definition at line 225 of file led_control.c.

References led_set_aux(), led_set_band(), led_set_error(), led_set_menu(), led_set_ptt(), led_set_rotate(), led_set_rx_ant(), led_set_rxant(), led_set_tx_ant(), led_set_txrx(), LED_STATE_OFF, LED_STATE_ON, LED_STATE_PTT_ACTIVE, and LED_STATE_-PTT_OK.

Referenced by main().

6.59.3.2 void led_set_aux (enum enum_led_state *state*)

Set the AUX LED status.

Parameters:

state The state of the LED

Definition at line 198 of file led_control.c.

References LED_AUX_BIT, and LED_STATE_ON.

Referenced by led_set_all().

6.59.3.3 void led_set_band (unsigned char *band*)

Set the band LEDs to the proper band.

Parameters:

band The band we wish to turn on the LED for

Definition at line 32 of file led_control.c.

Referenced by band_ctrl_change_band(), led_set_all(), and main().

6.59.3.4 void led_set_error (enum enum_led_state *state*)

Set the error LED status.

Parameters:

state The state of the LED

Definition at line 67 of file led_control.c.

References LED_ERROR_BIT, and LED_STATE_ON.

Referenced by bus_resend_message(), internal_comm_resend(), ISR(), led_set_all(), menu_-action(), rx_queue_add(), shutdown_device(), and tx_queue_add().

6.59.3.5 void led_set_menu (enum enum_led_state *state*)

Set the menu LED status.

Parameters:

state The state of the LED

Definition at line 216 of file led_control.c.

References LED_MENU_BIT, and LED_STATE_ON.

Referenced by event_poll_buttons(), and led_set_all().

6.59.3.6 void led_set_ptt (enum enum_led_ptt_state *state*)

Set the PTT LED.

Parameters:

state The state of the LED

Definition at line 47 of file led_control.c.

References LED_PTT_GREEN_BIT, LED_PTT_RED_BIT, LED_STATE_PTT_ACTIVE, LED_STATE_PTT_INHIBIT, and LED_STATE_PTT_OK.

Referenced by led_set_all(), main(), and main_update_ptt_status().

6.59.3.7 void led_set_rotate (enum enum_led_state *state*)

Set the Rotate LED status.

Parameters:

state The state of the LED

Definition at line 169 of file led_control.c.

References LED_ROTATE_BIT, and LED_STATE_ON.

Referenced by event_poll_buttons(), event_rotate_button_pressed(), event_rxant_button_-pressed(), and led_set_all().

6.59.3.8 void led_set_rotation_active (enum enum_led_state *state*)

Set the rotating led to active state, indicates if any antenna on the current band is rotating.

Parameters:

state The state of the LED

Definition at line 76 of file led_control.c.

References LED_ROTATION_ACTIVE_BIT, and LED_STATE_ON.

Referenced by ISR().

6.59.3.9 void led_set_rx_ant (unsigned char *index*, enum enum_led_state *state*)

Set the RX Antenna LED status.

Parameters:

index Which LED we wish to change the status of

state The state of the LED

Definition at line 128 of file led_control.c.

References LED_RX_BUTTON1_BIT, LED_RX_BUTTON2_BIT, LED_RX_BUTTON3_-BIT, LED_RX_BUTTON4_BIT, and LED_STATE_ON.

Referenced by band_ctrl_change_band(), and led_set_all().

6.59.3.10 void led_set_rxant (enum enum_led_state *state*)

Set the RX antenna LED status.

Parameters:

state The state of the LED

Definition at line 187 of file led_control.c.

References LED_RXANT_BIT, and LED_STATE_ON.

Referenced by band_ctrl_change_band(), event_rxant_button_pressed(), and led_set_all().

6.59.3.11 void led_set_submenu (enum enum_led_state *state*)

Set the AUX LED status.

Parameters:

state The state of the LED

Definition at line 207 of file led_control.c.

References LED_STATE_ON, and LED_SUBMENU_BIT.

Referenced by event_poll_buttons(), and event_sub_button_pressed().

6.59.3.12 void led_set_tx_ant (unsigned char *index*, enum enum_led_state *state*)

Set the TX Antenna LED status.

Parameters:

index Which LED we wish to change the status of

state The state of the LED

Definition at line 86 of file led_control.c.

References LED_STATE_ON, LED_TX_BUTTON1_BIT, LED_TX_BUTTON2_BIT, LED_TX_BUTTON3_BIT, and LED_TX_BUTTON4_BIT.

Referenced by band_ctrl_change_band(), event_tx_button1_pressed(), event_tx_button2_-pressed(), event_tx_button3_pressed(), event_tx_button4_pressed(), ISR(), led_set_all(), and set_tx_ant_leds().

6.59.3.13 void led_set_txrx (enum enum_led_state *state*)

Set the TX/RX mode LED status.

Parameters:

state The state of the LED

Definition at line 178 of file led_control.c.

References LED_STATE_ON, and LED_TXRX_BIT.

Referenced by led_set_all().

6.60 front_panel/menu.c File Reference

Menu system.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include "menu.h"
#include "board.h"
#include "glcd.h"
#include "fonts.h"
#include "display.h"
#include "radio_interface.h"
#include "errors.h"
#include "event_handler.h"
#include "led_control.h"
```

Defines

- #define MENU_OPTION_LEFT_POS 13

  Sets the intend from the left.

- #define MENU_OPTIONS 7

  Number of options in the menu system.

Functions

- void menu_show_text (struct_menu_text menu_text)

  Show the text of a menu on the display.

- void menu_init (void)

  Initialize the menu system.

- void menu_reset (void)

  Function will reset to init values, like menu level etc.

- void menu_show (void)

  Shows the menu.

- void menu_action (unsigned char menu_action_type)

Variables

- const struct_menu_option menu_errors [ ] = {{"Bus resend"},{"No bus sync"}, {"Bus TX queue full"}, {"Bus RX queue full"}, {"Int. comm resend"}}

  Menu options - Errors.

- const struct_menu_option **menu_misc** [ ] = {{"Reboot"}}
- unsigned char current_menu_option_selected [MENU_OPTIONS]

  The current selected menu option.

- unsigned char current_menu_level = 0

  Flag to indicate which menu level we are on.

- unsigned char current_menu_pos = 0

  Flag to indicate the current menu position.

- const struct_menu_option menu_option_band_selection_mode [ ] = {{"Manual"},{"Auto"}}

  Menu system option - band selection mode.

- const struct_menu_option menu_option_amp_ptt_output [ ] = {{"ON"},{"OFF"}}

  Menu system option - amp ptt output.

- const struct_menu_option menu_option_radio_ptt_output [ ] = {{"ON"},{"OFF"}}

  Menu system option - radio ptt output.

- const struct_menu_text menu_system_text [ ]

  Menu system.

6.60.1 Detailed Description

Menu system.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/menu.c "
```

Definition in file menu.c.

6.60.2 Function Documentation

6.60.2.1 void menu_action (unsigned char *menu_action_type*)

This function will handle an menu action "event"

Parameters:

menu_action_type Which action did occur?

Definition at line 173 of file menu.c.

References struct_runtime_settings::amplifier_ptt_output, struct_runtime_settings::band_-change_mode, BAND_CHANGE_MODE_AUTO, BAND_CHANGE_MODE_MANUAL, bootloader_start, current_menu_level, current_menu_option_selected, current_menu_pos, display_set_backlight(), event_get_errors(), event_set_error(), KNOB_FUNCTION_-AUTO, struct_runtime_settings::lcd_backlight_value, led_set_error(), LED_STATE_OFF, MENU_BUTTON_PRESSED, MENU_OPTION_TYPE_NORMAL, MENU_OPTION_-TYPE_SCROLL_NUMBERS, MENU_OPTIONS, MENU_POS_AMP_PTT, MENU_-POS_BACKLIGHT_LEVEL, MENU_POS_BAND_MODE, MENU_POS_MISC, MENU_-POS_RADIO_PTT, MENU_POS_SHOW_ERRORS, MENU_SCROLL_DOWN, MENU_-SCROLL_UP, menu_show(), NR_OF_ERRORS, struct_runtime_settings::radio_ptt_output, runtime_settings, and set_knob_function().

Referenced by event_poll_buttons(), event_pulse_sensor_down(), and event_pulse_sensor_-up().

6.60.2.2 void menu_show_text (struct_menu_text *menu_text*)

Show the text of a menu on the display.

Parameters:

menu_text The menu which we wish to show

Definition at line 79 of file menu.c.

References current_menu_level, current_menu_option_selected, current_menu_pos, display_-calculate_width(), event_get_errors(), glcd_line(), struct_menu_text::header, struct_-runtime_settings::lcd_backlight_value, MENU_OPTION_LEFT_POS, MENU_OPTION_-TYPE_NORMAL, MENU_OPTION_TYPE_SCROLL_NUMBERS, MENU_POS_-BACKLIGHT_LEVEL, MENU_POS_SHOW_ACTIVITY, MENU_POS_SHOW_ERRORS, NR_OF_ERRORS, struct_menu_text::option_count, struct_menu_text::option_type, struct_menu_text::options, struct_menu_text::pos, runtime_settings, and struct_menu_-option::text.

Referenced by menu_show().

6.60.3 Variable Documentation

6.60.3.1 const struct_menu_text menu_system_text[ ]

Initial value:

```
 {
{MENU_POS_BAND_MODE, "Band change", (struct_menu_option *)menu_option_band_selection_mode, 2,MENU_OPTION_TYPE_NORMAL},
{MENU_POS_RADIO_PTT, "Radio PTT", (struct_menu_option *)menu_option_radio_ptt_output, 2,MENU_OPTION_TYPE_NORMAL},
{MENU_POS_AMP_PTT, "Amplifier PTT", (struct_menu_option *)menu_option_amp_ptt_output, 2,MENU_OPTION_TYPE_NORMAL},
{MENU_POS_BACKLIGHT_LEVEL, "Backlight", NULL, 0,MENU_OPTION_TYPE_SCROLL_NUMBERS},
{MENU_POS_SHOW_ACTIVITY, "Network activity", NULL, 0,MENU_OPTION_TYPE_NONE},
{MENU_POS_MISC, "Miscellaneous", (struct_menu_option *)menu_misc, 1,MENU_OPTION_TYPE_NORMAL},
{MENU_POS_SHOW_ERRORS, "Errors", (struct_menu_option *)menu_errors, 0,MENU_OPTION_TYPE_NORMAL},
}
```

Menu system.

Definition at line 67 of file menu.c.

6.61 front_panel/menu.h File Reference

Menu system.

`#include <avr/pgmspace.h>`

Classes

- struct struct_menu_option
  Struct of a menu option.
- struct struct_menu_text
  Menu text structs.

Defines

- #define MENU_OPTION_TYPE_NORMAL 0
  Menu type option normal, regular choices.
- #define MENU_OPTION_TYPE_SCROLL_NUMBERS 1
  Menu type scroll numbers, for example increase/decrease a value.
- #define MENU_OPTION_TYPE_NONE 99
  No menu option.
- #define MENU_POS_BAND_MODE 0
  Show band change mode, auto or manual.
- #define MENU_POS_RADIO_PTT 1
  Show the radio output ptt ON/OFF.
- #define MENU_POS_AMP_PTT 2
  Show the amplifier output ptt ON/OFF.
- #define MENU_POS_BACKLIGHT_LEVEL 3
  Change the backlight level of the LCD.
- #define MENU_POS_SHOW_ACTIVITY 4
  Show network actvitity.
- #define MENU_POS_MISC 5
  Show MISC menu.
- #define MENU_POS_SHOW_ERRORS 6
  Show the error menu.
- #define MENU_SCROLL_UP 0
  Menu flag scroll up.

- #define MENU_SCROLL_DOWN 1

  Menu flag scroll down.

- #define MENU_BUTTON_PRESSED 2

  Menu flag button pressed.

Functions

- void menu_show (void)

  Shows the menu.

- void menu_action (unsigned char menu_action_type)
- void menu_init (void)

  Initialize the menu system.

- void menu_reset (void)

  Function will reset to init values, like menu level etc.

6.61.1 Detailed Description

Menu system.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/menu.h "
```

Definition in file menu.h.

6.61.2 Function Documentation

6.61.2.1 void menu_action (unsigned char *menu_action_type*)

This function will handle an menu action "event"

Parameters:

menu_action_type Which action did occur?

Definition at line 173 of file menu.c.

References struct_runtime_settings::amplifier_ptt_output, struct_runtime_settings::band_-change_mode, BAND_CHANGE_MODE_AUTO, BAND_CHANGE_MODE_MANUAL, bootloader_start, current_menu_level, current_menu_option_selected, current_menu_pos, display_set_backlight(), event_get_errors(), event_set_error(), KNOB_FUNCTION_-AUTO, struct_runtime_settings::lcd_backlight_value, led_set_error(), LED_STATE_OFF, MENU_BUTTON_PRESSED, MENU_OPTION_TYPE_NORMAL, MENU_OPTION_-TYPE_SCROLL_NUMBERS, MENU_OPTIONS, MENU_POS_AMP_PTT, MENU_-POS_BACKLIGHT_LEVEL, MENU_POS_BAND_MODE, MENU_POS_MISC, MENU_-POS_RADIO_PTT, MENU_POS_SHOW_ERRORS, MENU_SCROLL_DOWN, MENU_-SCROLL_UP, menu_show(), NR_OF_ERRORS, struct_runtime_settings::radio_ptt_output, runtime_settings, and set_knob_function().

Referenced by event_poll_buttons(), event_pulse_sensor_down(), and event_pulse_sensor_-up().

6.62 front_panel/pictures.h File Reference

Pictures which can be viewed on the display.

6.62.1 Detailed Description

Pictures which can be viewed on the display.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/pictures.h "
```

Definition in file pictures.h.

6.63 front_panel/powermeter.c File Reference

Power meter.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <string.h>
#include "powermeter.h"
#include "radio_interface.h"
#include "display.h"
#include "glcd.h"
#include "main.h"
```

Defines

- #define **POWERMETER_FLAG_ACTIVE** 0

Functions

- void powermeter_init (unsigned char pickup_addr, unsigned int text_update_rate, unsigned int bargraph_update_rate, unsigned int vswr_limit)

  Initialize the power meter.

- void powermeter_set_active (unsigned char state)

  Activate the power meter display /!*

- void powermeter_update_values (unsigned int fwd_pwr, unsigned int ref_pwr, unsigned int vswr)

  Update the values of the power meter.

- void powermeter_process_tasks (void)

  This function should be called as much as possible and it does all the updates, such checking for new data, updating display etc.

- void powermeter_1ms_tick (void)

  This function should be called at 1 ms intervals. It is to keep track of update rates etc for the display.

Variables

- powermeter_struct powermeter_status

  The current status of the power meter.

- unsigned char powermeter_flags

  Various flags used in the powermeter, defines can be found in powermeter.h.

- unsigned int counter_powermeter_update_text = 0

  The counter which keeps track of when we should update the power meter text.

- unsigned int counter_powermeter_update_bargraph = 0

  The counter which keeps track of when we should update the power meter bargraph.

6.63.1 Detailed Description

Power meter.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-05-12

```
#include "front_panel/powermeter.c"
```

Definition in file powermeter.c.

6.63.2 Function Documentation

6.63.2.1 void powermeter_init (unsigned char *pickup_addr*, unsigned int *text_update_rate*, unsigned int *bargraph_update_rate*, unsigned int *vswr_limit*)

Initialize the power meter.

Parameters:

pickup_addr The address of the powermeter unit that sends the information

text_update_rate How often we should refresh the text on the display

bargraph_update_rate How often we should update the bargraph of the display

vswr_limit What is the SWR limit of the device, when this is exceeded we shut down the possibility to PTT

Definition at line 53 of file powermeter.c.

References powermeter_struct::bargraph_update_rate, powermeter_struct::curr_fwd_-pwr_value, powermeter_struct::curr_ref_pwr_value, powermeter_struct::curr_vswr_value, powermeter_struct::pickup_addr, powermeter_struct::text_update_rate, and powermeter_-struct::vswr_limit.

Referenced by main().

6.63.2.2 void powermeter_set_active (unsigned char *state*)

Activate the power meter display /∗!

Parameters:

state If this is set to 1 we will activate the powermeter, if set to 0 we will deactivate it

Definition at line 66 of file powermeter.c.

6.63.2.3 void powermeter_update_values (unsigned int *fwd_pwr*, unsigned int *ref_pwr*, unsigned int *vswr*)

Update the values of the power meter.

Parameters:

fwd_pwr The current forward power in watts

ref_pwr The current reflected power in watts

vswr The current VSWR value, for example 151 means 1.51:1 in VSWR

Definition at line 86 of file powermeter.c.

6.64 front_panel/powermeter.h File Reference

Power meter functions.

Classes

- struct powermeter_struct

  Struct which contains information of the power meter status.

Functions

- void powermeter_update_values (unsigned int fwd_pwr, unsigned int ref_pwr, unsigned int vswr)

  Update the values of the power meter.

- void powermeter_init (unsigned char pickup_addr, unsigned int text_update_rate, unsigned int bargraph_update_rate, unsigned int vswr_limit)

  Initialize the power meter.

- void powermeter_process_tasks (void)

  This function should be called as much as possible and it does all the updates, such checking for new data, updating display etc.

- void powermeter_1ms_tick (void)

  This function should be called at 1 ms intervals. It is to keep track of update rates etc for the display.

6.64.1 Detailed Description

Power meter functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-05-12

```
#include "front_panel/powermeter.h"
```

Definition in file powermeter.h.

6.64.2 Function Documentation

6.64.2.1 void powermeter_init (unsigned char *pickup_addr*, unsigned int *text_update_rate*, unsigned int *bargraph_update_rate*, unsigned int *vswr_limit*)

Initialize the power meter.

Parameters:

pickup_ addr The address of the powermeter unit that sends the information

text_ update_ rate How often we should refresh the text on the display

bargraph_ update_ rate How often we should update the bargraph of the display

vswr_ limit What is the SWR limit of the device, when this is exceeded we shut down the possibility to PTT

Definition at line 53 of file powermeter.c.

References powermeter_struct::bargraph_update_rate, powermeter_struct::curr_fwd_-pwr_value, powermeter_struct::curr_ref_pwr_value, powermeter_struct::curr_vswr_value, powermeter_struct::pickup_addr, powermeter_struct::text_update_rate, and powermeter_-struct::vswr_limit.

Referenced by main().

6.64.2.2 void powermeter_update_values (unsigned int *fwd_pwr*, unsigned int *ref_pwr*, unsigned int *vswr*)

Update the values of the power meter.

Parameters:

fwd_pwr The current forward power in watts

ref_pwr The current reflected power in watts

vswr The current VSWR value, for example 151 means 1.51:1 in VSWR

Definition at line 86 of file powermeter.c.

6.65 front_panel/radio_interface.c File Reference

Radio interface, such as PTT AMP, PTT Radio, CAT etc.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "radio_interface.h"
#include "led_control.h"
#include "band_ctrl.h"
#include "main.h"
#include "usart.h"
#include "board.h"
#include "eeprom.h"
#include "display.h"
#include "../internal_comm.h"
#include "../internal_comm_commands.h"
#include "../global.h"
```

Functions

- void radio_interface_init (void)

  Initialize the radio interface.

- void radio_process_tasks (void)

  This function is called each lap in the main loop and we can use this to process certain tasks.

- unsigned int radio_get_current_freq (void)
- unsigned char radio_get_current_band (void)

  Retrieve the current band from the radio.

- void radio_ptt_active (void)

  Activate the radio PTT.

- void radio_ptt_deactive (void)

  Deactivate the radio PTT.

- void radio_tx_active (void)

  Set the TX ACTIVE output to high.

- void radio_tx_deactive (void)

  Set the TX ACTIVE output to high.

- void radio_inhibit_high (void)

  Set the inhibit signal to high.

- void radio_inhibit_low (void)

  Set the inhibit signal to low.

- unsigned char radio_get_ptt_status (void)

  Retrieve the ptt status, defines can be found in radio_interface.h.

- unsigned char radio_get_band_portion (void)
- void radio_set_current_band (unsigned char band)
- unsigned char radio_poll_ptt (void)

  Polls the status of the PTT input.

- unsigned char radio_poll_status (void)

  Polls the status of the radio and saves it into the radio_status structure.

- unsigned int radio_parse_freq (unsigned char ∗freq_data, unsigned char length, unsigned char radio_model)

  Parse the radios frequency.

- void radio_amp_ptt_active (void)

  Activate PTT amp.

- void radio_amp_ptt_deactive (void)

  Deactivate PTT amp.

- unsigned char radio_freq_to_band (unsigned int freq)

  Convert a radio frequency (integer) to band data.

- void radio_interface_set_model (unsigned char model)

  Set which radio model is used, saves it in the radio_settings struct.

- void radio_interface_set_interface (unsigned char interface_type)

  Set which radio interface is used, saves it in the radio_settings struct.

- void radio_interface_set_baudrate (unsigned char baudrate)

  Set which baudrate setting is used, saves it in the radio_settings struct.

- void radio_interface_set_stopbits (unsigned char stopbits)

  Set which number of stopbits should be used, saves it in the radio_settings struct.

- void radio_interface_set_civ_addr (unsigned char civ)

  Set which CI-V address the radio has got, saves it in the radio_settings struct.

- void radio_interface_set_ptt_input (unsigned char ptt_input)

  Set which PTT input that is used, saves it in the radio_settings struct.

- void radio_interface_set_poll_interval (unsigned char poll_interval)

Set the poll intervall for the radio band decoding, saves it in the radio_settings struct.

- unsigned char radio_interface_get_model (void)

  Get which radio model is used.

- unsigned char radio_interface_get_interface (void)

  Get which radio interface is used.

- unsigned char radio_interface_get_baudrate (void)

  Get which baudrate setting is used.

- unsigned char radio_interface_get_stopbits (void)

  Get which number of stopbits should be used.

- unsigned char radio_interface_get_civ_addr (void)

  Get which CI-V address the radio has got.

- unsigned char radio_interface_get_ptt_input (void)

  Get which PTT input that is used.

- unsigned char radio_interface_get_poll_interval (void)

  Get the poll intervall for the radio band decoding.

- void radio_interface_load_eeprom (void)

  This function will load data from the eeprom to the radio_settings struct.

- void radio_communicaton_timeout (void)
- unsigned char radio_get_cat_status (void)
- **ISR** (SIG_USART3_DATA)
- ISR (SIG_USART3_RECV)

  Interrupt which is called when a byte is received on the UART.

Variables

- unsigned char ∗ radio_serial_rx_buffer

  Serial receive buffer.

- unsigned char ∗ radio_serial_rx_buffer_start

  Start address of the serial receive buffer.

- struct_radio_status radio_status

  Radio status struct.

- struct_radio_settings radio_settings

  Radio settings struct.

- unsigned char radio_flags

  Flags to indicate various things which has happened to the radio.

- unsigned char ptt_status = 0

  Flag which does indicate if the radio is transmitting, amp is active etc.

- unsigned char radio_rx_data_counter

  External variable of the radio rx data counter used for a timeout.

6.65.1 Detailed Description

Radio interface, such as PTT AMP, PTT Radio, CAT etc.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/radio_interface.c "
```

Definition in file radio_interface.c.

6.65.2 Function Documentation

6.65.2.1 void radio_communicaton_timeout (void)

This function should be called if an timeout has occured on the serial communication. This function will then reset the pointers used for the CAT decoding

Definition at line 415 of file radio_interface.c.

References struct_radio_status::box_sent_request, radio_serial_rx_buffer, and radio_serial_-rx_buffer_start.

Referenced by ISR().

6.65.2.2 unsigned char radio_freq_to_band (unsigned int *freq*)

Convert a radio frequency (integer) to band data.

Parameters:

freq The frequency as integer

Returns:

The band of the frequency sent in as parameter. If band not found then it returns BAND_-UNDEFINED

Definition at line 315 of file radio_interface.c.

References band_ctrl_get_high_portion_high(), and band_ctrl_get_low_portion_low().

Referenced by radio_process_tasks().

6.65.2.3 unsigned char radio_get_band_portion (void)

Get the portion of the band the radio is on.

Returns:

Return BAND_HIGH if it's in the higher portion of the band, BAND_LOW if it's the lower portion. If neither then it returns BAND_UNDEFINED

Definition at line 180 of file radio_interface.c.

References band_ctrl_get_high_portion_high(), band_ctrl_get_high_portion_low(), band_ctrl_get_low_portion_high(), band_ctrl_get_low_portion_low(), struct_radio_-status::current_band, and struct_radio_status::current_freq.

Referenced by band_ctrl_get_portion(), display_update_radio_freq(), and main().

6.65.2.4 unsigned char radio_get_cat_status (void)

This function will tell us if the openASC box has sent any request to the radio

Returns:

1 if a request has been sent, 0 otherwise

Definition at line 423 of file radio_interface.c.

References struct_radio_status::box_sent_request.

Referenced by ISR().

6.65.2.5 unsigned char radio_get_current_band (void)

Retrieve the current band from the radio.

Returns:

The radios band

Definition at line 120 of file radio_interface.c.

References struct_radio_status::current_band.

Referenced by event_internal_comm_parse_message(), and main().

6.65.2.6 unsigned int radio_get_current_freq (void)

Retrieve the frequency from the radio. If it's configured for BCD it just retrieves the freq band The frequency is returned as an integer so for example 21350 means 21 MHz and 350 kHz.

Returns:

The frequency as an integer, max freq 65536

Definition at line 113 of file radio_interface.c.

References struct_radio_status::current_freq.

Referenced by display_update_radio_freq().

6.65.2.7 unsigned char radio_interface_get_baudrate (void)

Get which baudrate setting is used.

Returns:

Which baudrate setting is used

Definition at line 380 of file radio_interface.c.

References struct_radio_settings::baudrate.

6.65.2.8 unsigned char radio_interface_get_civ_addr (void)

Get which CI-V address the radio has got.

Returns:

The CI-V address

Definition at line 392 of file radio_interface.c.

References struct_radio_settings::civ_addr.

6.65.2.9 unsigned char radio_interface_get_interface (void)

Get which radio interface is used.

Returns:

The interface type

Definition at line 374 of file radio_interface.c.

References struct_radio_settings::interface_type.

Referenced by band_ctrl_get_portion(), display_update_radio_freq(), event_aux2_button_-pressed(), and main().

6.65.2.10 unsigned char radio_interface_get_model (void)

Get which radio model is used.

Returns:

The radio model

Definition at line 368 of file radio_interface.c.

References struct_radio_settings::radio_model.

6.65.2.11 unsigned char radio_interface_get_poll_interval (void)

Get the poll intervall for the radio band decoding.

Returns:

The poll interval in ms/10

Definition at line 404 of file radio_interface.c.

References struct_radio_settings::poll_interval.

Referenced by ISR().

6.65.2.12 unsigned char radio_interface_get_ptt_input (void)

Get which PTT input that is used.

Returns:

Which PTT input that is used

Definition at line 398 of file radio_interface.c.

References struct_radio_settings::ptt_input.

6.65.2.13 unsigned char radio_interface_get_stopbits (void)

Get which number of stopbits should be used.

Returns:

The number of stopbits that are used to interface the radio

Definition at line 386 of file radio_interface.c.

References struct_radio_settings::stopbits.

6.65.2.14 void radio_interface_set_baudrate (unsigned char *baudrate*)

Set which baudrate setting is used, saves it in the radio_settings struct.

Parameters:

baudrate Which baudrate setting to use

Definition at line 338 of file radio_interface.c.

References struct_radio_settings::baudrate.

6.65.2.15 void radio_interface_set_civ_addr (unsigned char *civ*)

Set which CI-V address the radio has got, saves it in the radio_settings struct.

Parameters:

civ The CI-V address

Definition at line 350 of file radio_interface.c.

References struct_radio_settings::civ_addr.

6.65.2.16 void radio_interface_set_interface (unsigned char *interface_type*)

Set which radio interface is used, saves it in the radio_settings struct.

Parameters:

interface_type The interface type

Definition at line 332 of file radio_interface.c.

References struct_radio_settings::interface_type.

6.65.2.17 void radio_interface_set_model (unsigned char *model*)

Set which radio model is used, saves it in the radio_settings struct.

Parameters:

model The radio model

Definition at line 326 of file radio_interface.c.

References struct_radio_settings::radio_model.

6.65.2.18 void radio_interface_set_poll_interval (unsigned char *poll_interval*)

Set the poll intervall for the radio band decoding, saves it in the radio_settings struct.

Parameters:

poll_interval The poll interval in ms/10

Definition at line 362 of file radio_interface.c.

References struct_radio_settings::poll_interval.

6.65.2.19 void radio_interface_set_ptt_input (unsigned char *ptt_input*)

Set which PTT input that is used, saves it in the radio_settings struct.

Parameters:

ptt_input Which PTT input that is used

Definition at line 356 of file radio_interface.c.

References struct_radio_settings::ptt_input.

6.65.2.20 void radio_interface_set_stopbits (unsigned char *stopbits*)

Set which number of stopbits should be used, saves it in the radio_settings struct.

Parameters:

stopbits The number of stopbits that are used to interface the radio

Definition at line 344 of file radio_interface.c.

References struct_radio_settings::stopbits.

6.65.2.21 unsigned int radio_parse_freq (unsigned char ∗ *freq_data*, unsigned char *length*, unsigned char *radio_model*)

Parse the radios frequency.

Parameters:

freq_data The frequency data sent in as an array of characters

length The length of the frequency data

radio_model The type of radio that the freq should be parsed for

Returns:

The radios frequency in integer format. So for example 21305 is 21 MHz and 305 kHz.

Definition at line 243 of file radio_interface.c.

References RADIO_MODEL_FT1000, RADIO_MODEL_FT1000MKV, RADIO_MODEL_-ICOM, and RADIO_MODEL_KENWOOD.

6.65.2.22 unsigned char radio_poll_ptt (void)

Polls the status of the PTT input.

Returns:

Return RADIO_PTT_ACTIVATE if the radio is PTT and RADIO_PTT_DEACTIVATE if it doesn't

Definition at line 197 of file radio_interface.c.

References RADIO_PTT_DEACTIVE.

6.65.2.23 unsigned char radio_poll_status (void)

Polls the status of the radio and saves it into the radio_status structure.

Returns:

0 if the poll went OK and 1 if it didn't

Definition at line 205 of file radio_interface.c.

References struct_radio_settings::civ_addr, display_update_radio_freq(), INHIBIT_NOT_-OK_TO_SEND_RADIO_TX, INT_COMM_GET_BAND_BCD_STATUS, struct_radio_-settings::interface_type, internal_comm_add_tx_message(), main_get_inhibit_state(), RADIO_INTERFACE_BCD, RADIO_INTERFACE_CAT_MON, RADIO_INTERFACE_-CAT_POLL, struct_radio_settings::radio_model, RADIO_MODEL_ICOM, and usart3_-transmit().

Referenced by main().

6.65.2.24 void radio_set_current_band (unsigned char *band*)

Set the current band

Parameters:

band The band we wish to set

Definition at line 191 of file radio_interface.c.

References struct_radio_status::current_band.

Referenced by event_internal_comm_parse_message().

6.66 front_panel/radio_interface.h File Reference

Radio interface, such as PTT AMP, PTT Radio, CAT etc.

Classes

- struct struct_radio_settings

  Radio settings struct.

- struct struct_radio_status

  The radio status struct.

Defines

- #define RADIO_FLAG_FREQ_CHANGED 0

  Flag to indicate that the frequency has changed.

- #define RADIO_MODEL_KENWOOD 0

  Kenwood radio connected to the box.

- #define RADIO_MODEL_ICOM 1

  ICOM radio connected to the box.

- #define RADIO_MODEL_FT1000 2

  FT1000D radio connected to the box.

- #define RADIO_MODEL_FT1000MP 3

  FT1000MP radio connected to the box.

- #define RADIO_MODEL_FT1000MKV 4

  FT1000MKV radio connected to the box.

- #define RADIO_MODEL_FT2000 5

  FT2000 radio connected to the box.

- #define RADIO_INTERFACE_MANUAL 0

  MANUAL mode which means no way to interface the radio.

- #define RADIO_INTERFACE_CAT_POLL 1

  Serial interface that connects to the radio, POLLING.

- #define RADIO_INTERFACE_CAT_MON 2

  Serial interface that connects to the radio, MONITORING.

- #define RADIO_INTERFACE_BCD 3

  BCD interface that connects the radio.

- #define RADIO_SENSE_UPPER_FLOOR 1

This bit is set if the radio PTT should be sensed from the upper floor.

- #define RADIO_SENSE_LOWER_FLOOR 2

  This bit is set if the radio PTT should be sensed from the lower floor.

- #define RADIO_SENSE_INVERTED 3

  This bit is set if the PTT sense input should be inverted, which means that PTT is active if it's low.

- #define RADIO_PTT_ACTIVE 1

  PTT Activate.

- #define RADIO_PTT_DEACTIVE 2

  PTT Deactivate.

- #define RADIO_SERIAL_RX_BUFFER_LENGTH 50

  Radio serial RX buffer length.

- #define RADIO_FLAG_RADIO_PTT 0

  Flag to indicate the radio PTT is active.

- #define RADIO_FLAG_AMP_PTT 1

  Flag to indicate the amp PTT is active.

- #define RADIO_FLAG_TX_ACTIVE 2

  Flag to indicate the box openASC has enabled a transmission (TX ACTIVE output).

- #define RADIO_SERIAL_BAUDRATE_1200 0

  Serial baudrate 1200 baud.

- #define RADIO_SERIAL_BAUDRATE_2400 1

  Serial baudrate 2400 baud.

- #define RADIO_SERIAL_BAUDRATE_4800 2

  Serial baudrate 4800 baud.

- #define RADIO_SERIAL_BAUDRATE_9600 3

  Serial baudrate 9600 baud.

- #define RADIO_SERIAL_BAUDRATE_14400 4

  Serial baudrate 14400 baud.

- #define RADIO_SERIAL_BAUDRATE_19200 5

  Serial baudrate 19200 baud.

- #define RADIO_SERIAL_BAUDRATE_28800 6

  Serial baudrate 28800 baud.

- #define RADIO_SERIAL_BAUDRATE_38400 7

  Serial baudrate 38400 baud.

- #define RADIO_SERIAL_BAUDRATE_57600 8

  Serial baudrate 57600 baud.

Functions

- void radio_process_tasks (void)

  This function is called each lap in the main loop and we can use this to process certain tasks.

- void radio_interface_init (void)

  Initialize the radio interface.

- unsigned int radio_get_current_freq (void)
- unsigned char radio_get_current_band (void)

  Retrieve the current band from the radio.

- void radio_set_current_band (unsigned char band)
- void **radio_ptt** (unsigned char status)
- unsigned char radio_get_band_portion (void)
- unsigned char radio_poll_status (void)

  Polls the status of the radio and saves it into the radio_status structure.

- void radio_ptt_active (void)

  Activate the radio PTT.

- void radio_ptt_deactive (void)

  Deactivate the radio PTT.

- void radio_amp_ptt_active (void)

  Activate PTT amp.

- void radio_amp_ptt_deactive (void)

  Deactivate PTT amp.

- void radio_inhibit_low (void)

  Set the inhibit signal to low.

- void radio_inhibit_high (void)

  Set the inhibit signal to high.

- unsigned char radio_freq_to_band (unsigned int freq)

  Convert a radio frequency (integer) to band data.

- void radio_interface_set_model (unsigned char model)

  Set which radio model is used, saves it in the radio_settings struct.

- void radio_interface_set_interface (unsigned char interface_type)

  Set which radio interface is used, saves it in the radio_settings struct.

- void radio_interface_set_baudrate (unsigned char baudrate)

    Set which baudrate setting is used, saves it in the radio_settings struct.

- void radio_interface_set_stopbits (unsigned char stopbits)

    Set which number of stopbits should be used, saves it in the radio_settings struct.

- void radio_interface_set_civ_addr (unsigned char civ)

    Set which CI-V address the radio has got, saves it in the radio_settings struct.

- void radio_interface_set_ptt_input (unsigned char ptt_input)

    Set which PTT input that is used, saves it in the radio_settings struct.

- void radio_interface_set_poll_interval (unsigned char poll_interval)

    Set the poll intervall for the radio band decoding, saves it in the radio_settings struct.

- unsigned char radio_interface_get_model (void)

    Get which radio model is used.

- unsigned char radio_interface_get_interface (void)

    Get which radio interface is used.

- unsigned char radio_interface_get_baudrate (void)

    Get which baudrate setting is used.

- unsigned char radio_interface_get_stopbits (void)

    Get which number of stopbits should be used.

- unsigned char radio_interface_get_civ_addr (void)

    Get which CI-V address the radio has got.

- unsigned char radio_interface_get_ptt_input (void)

    Get which PTT input that is used.

- unsigned char radio_interface_get_poll_interval (void)

    Get the poll intervall for the radio band decoding.

- void radio_interface_load_eeprom (void)

    This function will load data from the eeprom to the radio_settings struct.

- void radio_tx_active (void)

    Set the TX ACTIVE output to high.

- void radio_tx_deactive (void)

    Set the TX ACTIVE output to high.

- unsigned char radio_get_ptt_status (void)

    Retrieve the ptt status, defines can be found in radio_interface.h.

- unsigned char radio_get_cat_status (void)
- void radio_communicaton_timeout (void)

6.66.1 Detailed Description

Radio interface, such as PTT AMP, PTT Radio, CAT etc.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/radio_interface.h "
```

Definition in file radio_interface.h.

6.66.2 Function Documentation

6.66.2.1 void radio_communicaton_timeout (void)

This function should be called if an timeout has occured on the serial communication. This function will then reset the pointers used for the CAT decoding

Definition at line 415 of file radio_interface.c.

References struct_radio_status::box_sent_request, radio_serial_rx_buffer, and radio_serial_-rx_buffer_start.

Referenced by ISR().

6.66.2.2 unsigned char radio_freq_to_band (unsigned int *freq*)

Convert a radio frequency (integer) to band data.

Parameters:

freq The frequency as integer

Returns:

The band of the frequency sent in as parameter. If band not found then it returns BAND_-UNDEFINED

Definition at line 315 of file radio_interface.c.

References band_ctrl_get_high_portion_high(), and band_ctrl_get_low_portion_low().

Referenced by radio_process_tasks().

6.66.2.3 unsigned char radio_get_band_portion (void)

Get the portion of the band the radio is on.

Returns:

Return BAND_HIGH if it's in the higher portion of the band, BAND_LOW if it's the lower portion. If neither then it returns BAND_UNDEFINED

Definition at line 180 of file radio_interface.c.

References band_ctrl_get_high_portion_high(), band_ctrl_get_high_portion_low(), band_ctrl_get_low_portion_high(), band_ctrl_get_low_portion_low(), struct_radio_-status::current_band, and struct_radio_status::current_freq.

Referenced by band_ctrl_get_portion(), display_update_radio_freq(), and main().

6.66.2.4 unsigned char radio_get_cat_status (void)

This function will tell us if the openASC box has sent any request to the radio

Returns:

1 if a request has been sent, 0 otherwise

Definition at line 423 of file radio_interface.c.

References struct_radio_status::box_sent_request.

Referenced by ISR().

6.66.2.5 unsigned char radio_get_current_band (void)

Retrieve the current band from the radio.

Returns:

The radios band

Definition at line 120 of file radio_interface.c.

References struct_radio_status::current_band.

Referenced by event_internal_comm_parse_message(), and main().

6.66.2.6 unsigned int radio_get_current_freq (void)

Retrieve the frequency from the radio. If it's configured for BCD it just retrieves the freq band The frequency is returned as an integer so for example 21350 means 21 MHz and 350 kHz.

Returns:

The frequency as an integer, max freq 65536

Definition at line 113 of file radio_interface.c.

References struct_radio_status::current_freq.

Referenced by display_update_radio_freq().

6.66.2.7 unsigned char radio_interface_get_baudrate (void)

Get which baudrate setting is used.

Returns:

Which baudrate setting is used

Definition at line 380 of file radio_interface.c.

References struct_radio_settings::baudrate.

6.66.2.8 unsigned char radio_interface_get_civ_addr (void)

Get which CI-V address the radio has got.

Returns:

The CI-V address

Definition at line 392 of file radio_interface.c.

References struct_radio_settings::civ_addr.

6.66.2.9 unsigned char radio_interface_get_interface (void)

Get which radio interface is used.

Returns:

The interface type

Definition at line 374 of file radio_interface.c.

References struct_radio_settings::interface_type.

Referenced by band_ctrl_get_portion(), display_update_radio_freq(), event_aux2_button_-pressed(), and main().

6.66.2.10 unsigned char radio_interface_get_model (void)

Get which radio model is used.

Returns:

The radio model

Definition at line 368 of file radio_interface.c.

References struct_radio_settings::radio_model.

6.66.2.11 unsigned char radio_interface_get_poll_interval (void)

Get the poll intervall for the radio band decoding.

Returns:

The poll interval in ms/10

Definition at line 404 of file radio_interface.c.

References struct_radio_settings::poll_interval.

Referenced by ISR().

6.66.2.12 unsigned char radio_interface_get_ptt_input (void)

Get which PTT input that is used.

Returns:

Which PTT input that is used

Definition at line 398 of file radio_interface.c.

References struct_radio_settings::ptt_input.

6.66.2.13 unsigned char radio_interface_get_stopbits (void)

Get which number of stopbits should be used.

Returns:

The number of stopbits that are used to interface the radio

Definition at line 386 of file radio_interface.c.

References struct_radio_settings::stopbits.

6.66.2.14 void radio_interface_set_baudrate (unsigned char *baudrate*)

Set which baudrate setting is used, saves it in the radio_settings struct.

Parameters:

baudrate Which baudrate setting to use

Definition at line 338 of file radio_interface.c.

References struct_radio_settings::baudrate.

6.66.2.15 void radio_interface_set_civ_addr (unsigned char *civ*)

Set which CI-V address the radio has got, saves it in the radio_settings struct.

Parameters:

civ The CI-V address

Definition at line 350 of file radio_interface.c.

References struct_radio_settings::civ_addr.

6.66.2.16 void radio_interface_set_interface (unsigned char *interface_type*)

Set which radio interface is used, saves it in the radio_settings struct.

Parameters:

interface_type The interface type

Definition at line 332 of file radio_interface.c.

References struct_radio_settings::interface_type.

6.66.2.17 void radio_interface_set_model (unsigned char *model*)

Set which radio model is used, saves it in the radio_settings struct.

Parameters:

model The radio model

Definition at line 326 of file radio_interface.c.

References struct_radio_settings::radio_model.

6.66.2.18 void radio_interface_set_poll_interval (unsigned char *poll_interval*)

Set the poll intervall for the radio band decoding, saves it in the radio_settings struct.

Parameters:

poll_interval The poll interval in ms/10

Definition at line 362 of file radio_interface.c.

References struct_radio_settings::poll_interval.

6.66.2.19 void radio_interface_set_ptt_input (unsigned char *ptt_input*)

Set which PTT input that is used, saves it in the radio_settings struct.

Parameters:

ptt_input Which PTT input that is used

Definition at line 356 of file radio_interface.c.

References struct_radio_settings::ptt_input.

6.66.2.20 void radio_interface_set_stopbits (unsigned char *stopbits*)

Set which number of stopbits should be used, saves it in the radio_settings struct.

Parameters:

stopbits The number of stopbits that are used to interface the radio

Definition at line 344 of file radio_interface.c.

References struct_radio_settings::stopbits.

6.66.2.21 unsigned char radio_poll_status (void)

Polls the status of the radio and saves it into the radio_status structure.

Returns:

0 if the poll went OK and 1 if it didn't

Definition at line 205 of file radio_interface.c.

References struct_radio_settings::civ_addr, display_update_radio_freq(), INHIBIT_NOT_-OK_TO_SEND_RADIO_TX, INT_COMM_GET_BAND_BCD_STATUS, struct_radio_-settings::interface_type, internal_comm_add_tx_message(), main_get_inhibit_state(), RADIO_INTERFACE_BCD, RADIO_INTERFACE_CAT_MON, RADIO_INTERFACE_-CAT_POLL, struct_radio_settings::radio_model, RADIO_MODEL_ICOM, and usart3_-transmit().

Referenced by main().

6.66.2.22 void radio_set_current_band (unsigned char *band*)

Set the current band

Parameters:

band The band we wish to set

Definition at line 191 of file radio_interface.c.

References struct_radio_status::current_band.

Referenced by event_internal_comm_parse_message().

6.67 front_panel/remote_control.c File Reference

Remote control of the openASC box.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include "event_handler.h"
#include "remote_control.h"
```

Defines

- #define FLAG_REMOTE_CONTROL_MODE_ACTIVE 0

  Flag that the remote control is active.

Functions

- void remote_control_activate_remote_mode (void)

  Activate the remote control mode.

- void remote_control_deactivate_remote_mode (void)

  Deactivate the remote control mode.

- unsigned char remote_control_get_remote_mode (void)

  Get the current remote control mode.

- void remote_control_parse_button (unsigned char button)

  Parse a button press event, will perform an action depending on which button we wish to press.

- void remote_control_parse_command (unsigned char command, unsigned char length, char *data)

  Parse a remote control command and perform the proper action.

Variables

- unsigned char remote_control_flags

  Flags used in the remote control.

6.67.1 Detailed Description

Remote control of the openASC box.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/remote_control.c"
```

Definition in file remote_control.c.

6.67.2 Function Documentation

6.67.2.1 unsigned char remote_control_get_remote_mode (void)

Get the current remote control mode.

Returns:

1 if remote mode is active, 0 if it is not active

Definition at line 50 of file remote_control.c.

References FLAG_REMOTE_CONTROL_MODE_ACTIVE, and remote_control_flags.

6.67.2.2 void remote_control_parse_button (unsigned char *button*)

Parse a button press event, will perform an action depending on which button we wish to press.

Parameters:

button The button we wish to press

Definition at line 56 of file remote_control.c.

Referenced by remote_control_parse_command().

6.67.2.3 void remote_control_parse_command (unsigned char *command*, unsigned char *length*, char ∗ *data*)

Parse a remote control command and perform the proper action.

Parameters:

command The command we wish to parse
length The length of the data
data The data content

Definition at line 67 of file remote_control.c.

References REMOTE_CONTROL_ACTIVATE_MODE, remote_control_activate_remote_mode(), REMOTE_CONTROL_BUTTON_PRESSED, REMOTE_CONTROL_DEACTIVATE_MODE, remote_control_deactivate_remote_mode(), and remote_control_parse_button().

Referenced by event_internal_comm_parse_message().

6.68 front_panel/remote_control.h File Reference

Remote control of the openASC box.

Defines

- #define REMOTE_CONTROL_ACTIVATE_MODE 0x01

  Command to activate the remote control mode.

- #define REMOTE_CONTROL_DEACTIVATE_MODE 0x02

  Command to deactivate the remote control mode.

- #define REMOTE_CONTROL_BUTTON_PRESSED 0x10

  A button should be pressed.

- #define REMOTE_CONTROL_RX_ANT_TEXT 0x11

  Command for sending rx antenna button texts.

Functions

- void remote_control_activate_remote_mode (void)

  Activate the remote control mode.

- void remote_control_deactivate_remote_mode (void)

  Deactivate the remote control mode.

- unsigned char remote_control_get_remote_mode (void)

  Get the current remote control mode.

- void remote_control_parse_command (unsigned char command, unsigned char length, char ∗data)

  Parse a remote control command and perform the proper action.

- void remote_control_parse_button (unsigned char button)

  Parse a button press event, will perform an action depending on which button we wish to press.

6.68.1 Detailed Description

Remote control of the openASC box.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/remote_control.h"
```

Definition in file remote_control.h.

6.68.2 Function Documentation

6.68.2.1 unsigned char remote_control_get_remote_mode (void)

Get the current remote control mode.

Returns:

1 if remote mode is active, 0 if it is not active

Definition at line 50 of file remote_control.c.

References FLAG_REMOTE_CONTROL_MODE_ACTIVE, and remote_control_flags.

6.68.2.2 void remote_control_parse_button (unsigned char *button*)

Parse a button press event, will perform an action depending on which button we wish to press.

Parameters:

button The button we wish to press

Definition at line 56 of file remote_control.c.

Referenced by remote_control_parse_command().

6.68.2.3 void remote_control_parse_command (unsigned char *command*, unsigned char *length*, char * *data*)

Parse a remote control command and perform the proper action.

Parameters:

command The command we wish to parse

length The length of the data

data The data content

Definition at line 67 of file remote_control.c.

References REMOTE_CONTROL_ACTIVATE_MODE, remote_control_activate_-remote_mode(), REMOTE_CONTROL_BUTTON_PRESSED, REMOTE_CONTROL_-DEACTIVATE_MODE, remote_control_deactivate_remote_mode(), and remote_control_-parse_button().

Referenced by event_internal_comm_parse_message().

6.69 front_panel/rotary_encoder.c File Reference

Rotary encoder functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include "board.h"
#include "rotary_encoder.h"
```

Functions

- unsigned char poll_encoder_state (void)

  Poll the rotary encoder pin states.
- int rotary_encoder_poll (void)

  Poll the rotary encoder.

Variables

- unsigned char encoder_last_state = 0

  The last state of the encoder.
- unsigned char encoder_current_state = 0

  The current state of the encoder.

6.69.1 Detailed Description

Rotary encoder functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/rotary_encoder.c"
```

Definition in file rotary_encoder.c.

6.69.2 Function Documentation

6.69.2.1 unsigned char poll_encoder_state (void)

Poll the rotary encoder pin states.

Returns:

The state of the rotary encoder pins

Definition at line 38 of file rotary_encoder.c.

References PULSE_SENSOR_BIT1, and PULSE_SENSOR_BIT2.

Referenced by rotary_encoder_poll().

6.69.2.2 int rotary_encoder_poll (void)

Poll the rotary encoder.

Returns:

Returns 0 if nothing happened, -1 if rotary CCW and 1 if CW

Definition at line 44 of file rotary_encoder.c.

References encoder_current_state, encoder_last_state, and poll_encoder_state().

Referenced by main().

6.70 front_panel/rotary_encoder.h File Reference

Rotary encoder functions.

Functions

- int rotary_encoder_poll (void)

  Poll the rotary encoder.

6.70.1 Detailed Description

Rotary encoder functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/rotary_encoder.h"
```

Definition in file rotary_encoder.h.

6.70.2 Function Documentation

6.70.2.1 int rotary_encoder_poll (void)

Poll the rotary encoder.

Returns:

Returns 0 if nothing happened, -1 if rotary CCW and 1 if CW

Definition at line 44 of file rotary_encoder.c.

References encoder_current_state, encoder_last_state, and poll_encoder_state().

Referenced by main().

6.71 front_panel/sequencer.c File Reference

Sequencer.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "radio_interface.h"
#include "sequencer.h"
#include "main.h"
#include "led_control.h"
#include "usart.h"
#include "../global.h"
#include "../event_queue.h"
#include "antenna_ctrl.h"
#include "eeprom.h"
```

Defines

- #define PTT_ACTIVE_FOOTSWITCH 0

  The footswitch PTT input is active.

- #define PTT_ACTIVE_RADIO_SENSE 1

  The radio sense PTT input is active.

- #define PTT_ACTIVE_COMPUTER_RTS 2

  The computer PTT input is active.

Functions

- unsigned char sequencer_get_ptt_active (void)

  Retrieve which PTT inputs that are currently active, defines above.

- void sequencer_load_eeprom (void)

  This function will load data from the eeprom to the ptt_sequencer struct.

- void sequencer_footsw_pressed (void)

  Function to be called if the footswitch is pressed.

- void sequencer_footsw_released (void)

  Function to be called if the footswitch is released.

- void sequencer_computer_rts_activated (void)

  Function to be called if the computer rts is activated.

- void sequencer_computer_rts_deactivated (void)

  Function to be called if the computer rts is deactivated.

- void sequencer_radio_sense_activated (void)

  Function to be called if the radio sense input is activated.

- void sequencer_radio_sense_deactivated (void)

  Function to be called if the radio sense input is deactivated.

- unsigned char sequencer_get_rts_polarity ()

  Retrieve the polarity of the Computer RTS signal.

- unsigned char sequencer_get_sense_polarity ()

  Retrieve the polarity of the radio sense signal.

- unsigned char sequencer_ptt_active (void)

  Will return if the PTT is active or not.

- unsigned char sequencer_get_radio_sense (void)

  Retrieve if the radio sense should be sensed from upper floor or bottom.

Variables

- unsigned char ptt_active = 0

  The status of the PTT, see defines above.

- struct_ptt ptt_sequencer

  PTT sequencer data.

6.71.1 Detailed Description

Sequencer.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/sequencer.c"
```

Definition in file sequencer.c.

6.71.2 Function Documentation

6.71.2.1 unsigned char sequencer_get_radio_sense (void)

Retrieve if the radio sense should be sensed from upper floor or bottom.

Returns:

0 if lower floor, 1 if upper floor

Definition at line 231 of file sequencer.c.

References struct_ptt::ptt_input, and PTT_INPUT_RADIO_SENSE_UP.

Referenced by event_parse_ext_event().

6.71.2.2 unsigned char sequencer_get_rts_polarity (void)

Retrieve the polarity of the Computer RTS signal.

Returns:

1 if the polarity is active low (inverted), 0 otherwise

Definition at line 210 of file sequencer.c.

References struct_ptt::ptt_input, and PTT_INPUT_INVERTED_COMPUTER_RTS.

Referenced by event_parse_ext_event().

6.71.2.3 unsigned char sequencer_get_sense_polarity (void)

Retrieve the polarity of the radio sense signal.

Returns:

1 if the polarity is active low (inverted), 0 otherwise

Definition at line 219 of file sequencer.c.

References struct_ptt::ptt_input, and PTT_INPUT_INVERTED_RADIO_SENSE.

Referenced by event_parse_ext_event().

6.71.2.4 unsigned char sequencer_ptt_active (void)

Will return if the PTT is active or not.

Returns:

The state of the ptt_active variable, 0 if nothing is PTTing the radio

Definition at line 225 of file sequencer.c.

References ptt_active.

6.72 front_panel/sequencer.h File Reference

Sequencer.

Classes

- struct struct_ptt_sequencer

    All the delays are divided with 10 so 100 is really 1000 ms which makes the maximium delay 2550 ms.

- struct struct_ptt

    PTT Sequencer struct.

Defines

- #define SEQUENCER_EVENT_TYPE_PTT_TX_ACTIVE_ON 1

    Event that the radio should be PTT:ed from footswitch.

- #define SEQUENCER_EVENT_TYPE_PTT_INHIBIT_ON 2

    Event that the inhibit output should be on.

- #define SEQUENCER_EVENT_TYPE_PTT_RADIO_ON 3

    Event that the radio should be PTT:ed from footswitch.

- #define SEQUENCER_EVENT_TYPE_PTT_AMP_ON 4

    Event that the amp should be PTT:ed from footswitch.

- #define SEQUENCER_EVENT_TYPE_PTT_TX_ACTIVE_OFF 5

    Event that the inhibit should be activated from footswitch.

- #define SEQUENCER_EVENT_TYPE_PTT_INHIBIT_OFF 6

    Event that the TX active output should be off.

- #define SEQUENCER_EVENT_TYPE_PTT_RADIO_OFF 7

    Event that the radio should be deactivated from footswitch.

- #define SEQUENCER_EVENT_TYPE_PTT_AMP_OFF 8

    Event that the amp should be deactivated from footswitch.

- #define SEQUENCER_EVENT_TYPE_PTT_INHIBIT_OFF 9

    Event that the TX active output should be off.

- #define PTT_INPUT_FOOTSWITCH 0

    Flag bit offset for the footswitch.

- #define PTT_INPUT_RADIO_SENSE_UP 1

    Flag bit offset for the radio sense on the upper floor.

- #define PTT_INPUT_RADIO_SENSE_LO 2

  Flag bit offset for the radio sense on the lower floor.

- #define PTT_INPUT_COMPUTER_RTS 3

  Flag bit offset for the COMPUTER RTS signal.

- #define PTT_INPUT_INVERTED_RADIO_SENSE 4

  Flag bit offset for an inverted radio sense signal.

- #define PTT_INPUT_INVERTED_COMPUTER_RTS 5

  Flag bit offset for an inverted computer rts signal.

- #define PTT_INPUT_INHIBIT_POLARITY 6

  Flag bit offset for the inhibit polarity.

- #define SEQUENCER_PTT_RADIO_ENABLED 0

  Sequencer enabled for RADIO.

- #define SEQUENCER_PTT_AMP_ENABLED 1

  Sequencer enabled for AMP.

- #define SEQUENCER_PTT_INHIBIT_ENABLED 2

  Sequencer enabled for INHIBIT.

Functions

- unsigned char sequencer_get_ptt_active (void)

  Retrieve which PTT inputs that are currently active, defines above.

- void sequencer_load_eeprom (void)

  This function will load data from the eeprom to the ptt_sequencer struct.

- void sequencer_footsw_pressed (void)

  Function to be called if the footswitch is pressed.

- void sequencer_footsw_released (void)

  Function to be called if the footswitch is released.

- void sequencer_computer_rts_activated (void)

  Function to be called if the computer rts is activated.

- void sequencer_computer_rts_deactivated (void)

  Function to be called if the computer rts is deactivated.

- void sequencer_radio_sense_activated (void)

  Function to be called if the radio sense input is activated.

- void sequencer_radio_sense_deactivated (void)

Function to be called if the radio sense input is deactivated.

- unsigned char sequencer_get_rts_polarity (void)

  Retrieve the polarity of the Computer RTS signal.

- unsigned char sequencer_get_sense_polarity (void)

  Retrieve the polarity of the radio sense signal.

- unsigned char sequencer_ptt_active (void)

  Will return if the PTT is active or not.

- unsigned char sequencer_get_radio_sense (void)

  Retrieve if the radio sense should be sensed from upper floor or bottom.

6.72.1 Detailed Description

Sequencer.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/sequencer.h"
```

Definition in file sequencer.h.

6.72.2 Define Documentation

6.72.2.1 #define SEQUENCER_EVENT_TYPE_PTT_INHIBIT_OFF 9

Event that the TX active output should be off.

Event that the inhibit should be deactiated from footswitch.

Definition at line 47 of file sequencer.h.

6.72.2.2 #define SEQUENCER_EVENT_TYPE_PTT_INHIBIT_OFF 6

Event that the TX active output should be off.

Event that the inhibit should be deactiated from footswitch.

Definition at line 47 of file sequencer.h.

Referenced by sequencer_computer_rts_deactivated(), and sequencer_footsw_released().

6.72.2.3 #define SEQUENCER_EVENT_TYPE_PTT_TX_ACTIVE_ON 1

Event that the radio should be PTT:ed from footswitch.

Sequencer message types, this is used to we can keep track of different messages in the event queue. So if an event is aborted we can easily just remove the upcoming events from the queue, that doesn't need to be executed

Definition at line 31 of file sequencer.h.

Referenced by sequencer_computer_rts_activated(), sequencer_computer_rts_deactivated(), sequencer_footsw_pressed(), and sequencer_footsw_released().

6.72.3 Function Documentation

6.72.3.1 unsigned char sequencer_get_radio_sense (void)

Retrieve if the radio sense should be sensed from upper floor or bottom.

Returns:

0 if lower floor, 1 if upper floor

Definition at line 231 of file sequencer.c.

References struct_ptt::ptt_input, and PTT_INPUT_RADIO_SENSE_UP.

Referenced by event_parse_ext_event().

6.72.3.2 unsigned char sequencer_get_rts_polarity (void)

Retrieve the polarity of the Computer RTS signal.

Returns:

1 if the polarity is active low (inverted), 0 otherwise

Definition at line 210 of file sequencer.c.

References struct_ptt::ptt_input, and PTT_INPUT_INVERTED_COMPUTER_RTS.

Referenced by event_parse_ext_event().

6.72.3.3 unsigned char sequencer_get_sense_polarity (void)

Retrieve the polarity of the radio sense signal.

Returns:

1 if the polarity is active low (inverted), 0 otherwise

Definition at line 219 of file sequencer.c.

References struct_ptt::ptt_input, and PTT_INPUT_INVERTED_RADIO_SENSE.

Referenced by event_parse_ext_event().

6.72.3.4 unsigned char sequencer_ptt_active (void)

Will return if the PTT is active or not.

Returns:

The state of the ptt_active variable, 0 if nothing is PTTing the radio

Definition at line 225 of file sequencer.c.

References ptt_active.

6.73 front_panel/sub_menu.c File Reference

Antenna sub menu functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <string.h>
#include "sub_menu.h"
#include "antenna_ctrl.h"
#include "main.h"
#include "eeprom.h"
#include "band_ctrl.h"
#include "../global.h"
#include "../internal_comm.h"
#include "../wmv_bus/bus.h"
#include "../wmv_bus/bus_rx_queue.h"
#include "../wmv_bus/bus_tx_queue.h"
#include "../wmv_bus/bus_commands.h"
```

Functions

- void sub_menu_load (unsigned char band_index)
  Load a set of sub menu from the EEPROM for a specific band.
- unsigned char ∗ sub_menu_get_text (unsigned char ant_index, unsigned char pos)
  Get the text for the sub menu.
- unsigned char sub_menu_get_current_pos (unsigned char ant_index)
  Get the current position of the sub menu cursor.
- void sub_menu_set_current_pos (unsigned char ant_index, unsigned char new_pos)
  Set the current sub menu option.
- unsigned char sub_menu_get_count (void)
  Get the number of antennas which has got a sub menu configured.
- unsigned char sub_menu_get_type (unsigned char ant_index)
  Get the sub menu type of an antenna.
- void sub_menu_pos_down (unsigned char ant_index)
  This function should be called when we wish to decrease the selected sub menu option.
- void sub_menu_pos_up (unsigned char ant_index)
  This function should be called when we wish to increase the selected sub menu option.

- void sub_menu_send_data_to_bus (unsigned char ant_index, unsigned char pos)

  Send the output string for the sub menu position to the bus.

- void sub_menu_deactivate_all (void)

  Will deactivate all currently selected outputs which has been sent out on the bus.

- void sub_menu_activate_all (void)

  This function will go through the sub menus and if there is one configured it will activae its default option which is index 0.

Variables

- struct_sub_menu_array current_sub_menu_array [4]

  Current sub menu array.

- unsigned char curr_option_selected [4] = {0,0,0,0}

  Which option is currently selected of the sub menu options.

- unsigned char current_activated_sub_outputs [4][SUB_MENU_ARRAY_STR_SIZE]

  Array which we store the current devices which we have activated antenna outputs on.

- unsigned char current_activated_sub_outputs_length [4] = {0,0,0,0}

  How many devices we have activated antenna outputs on.

6.73.1 Detailed Description

Antenna sub menu functions.

Antenna sub menu functions

Author:

Mikael Larsmark, SM2WMV

Date:

2010-04-28

```
#include "front_panel/sub_menu.c"
```

Definition in file sub_menu.c.

6.73.2 Function Documentation

6.73.2.1 unsigned char sub_menu_get_count (void)

Get the number of antennas which has got a sub menu configured.

Returns:

The number of antennas which has got sub menus, (0-4)

Definition at line 101 of file sub_menu.c.

References antenna_ctrl_get_sub_menu_type(), and SUBMENU_NONE.

Referenced by event_sub_button_pressed().

6.73.2.2 unsigned char sub_menu_get_current_pos (unsigned char *ant_index*)

Get the current position of the sub menu cursor.

Parameters:

ant_index The antenna index, (0-3)

Returns:

The cursor position of the sub menu

Definition at line 88 of file sub_menu.c.

References curr_option_selected.

Referenced by display_rotator_directions(), display_show_sub_menu(), main(), sub_menu_-pos_down(), and sub_menu_pos_up().

6.73.2.3 unsigned char∗ sub_menu_get_text (unsigned char *ant_index*, unsigned char *pos*)

Get the text for the sub menu.

Parameters:

ant_index The antenna index we wish to get the antenna text for

pos Which sub menu position to show

Returns:

Returns the text of the sub menu antenna index

Definition at line 77 of file sub_menu.c.

References antenna_ctrl_get_sub_menu_type(), and SUBMENU_VERT_ARRAY.

Referenced by display_rotator_directions(), and display_show_sub_menu().

6.73.2.4 unsigned char sub_menu_get_type (unsigned char *ant_index*)

Get the sub menu type of an antenna.

Parameters:

ant_index The antenna we wish to get the sub menu type of, (0-3)

Returns:

The sub meny type of the antenna

Definition at line 114 of file sub_menu.c.

References antenna_ctrl_get_sub_menu_type().

Referenced by event_pulse_sensor_down(), event_pulse_sensor_up(), sub_menu_pos_down(), and sub_menu_pos_up().

6.73.2.5 void sub_menu_load (unsigned char *band_index*)

Load a set of sub menu from the EEPROM for a specific band.

Parameters:

band_index Which we band

Definition at line 60 of file sub_menu.c.

References antenna_ctrl_get_sub_menu_type(), curr_option_selected, eeprom_get_ant_-sub_menu_array_structure(), and SUBMENU_VERT_ARRAY.

Referenced by band_ctrl_load_band().

6.73.2.6 void sub_menu_pos_down (unsigned char *ant_index*)

This function should be called when we wish to decrease the selected sub menu option.

Parameters:

ant_index Which antenna we wish to decrease the sub menu position of

Definition at line 120 of file sub_menu.c.

References sub_menu_get_current_pos(), sub_menu_get_type(), sub_menu_set_current_-pos(), and SUBMENU_VERT_ARRAY.

Referenced by event_pulse_sensor_down().

6.73.2.7 void sub_menu_pos_up (unsigned char *ant_index*)

This function should be called when we wish to increase the selected sub menu option.

Parameters:

ant_index Which antenna we wish to increase the sub menu position of

Definition at line 131 of file sub_menu.c.

References sub_menu_get_current_pos(), sub_menu_get_type(), sub_menu_set_current_-pos(), and SUBMENU_VERT_ARRAY.

Referenced by event_pulse_sensor_up().

6.73.2.8 void sub_menu_send_data_to_bus (unsigned char *ant_index*, unsigned char *pos*)

Send the output string for the sub menu position to the bus.

Parameters:

ant_index The index of the antenna you wish to send the string of

pos The sub menu position we wish to send the output str of

Definition at line 144 of file sub_menu.c.

References antenna_ctrl_deactivate_outputs(), bus_add_tx_message(), BUS_-CMD_DRIVER_ACTIVATE_SUBMENU_ANT1_OUTPUT, BUS_CMD_DRIVER_-ACTIVATE_SUBMENU_ANT2_OUTPUT, BUS_CMD_DRIVER_ACTIVATE_-SUBMENU_ANT3_OUTPUT, BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT4_-OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT1_OUTPUTS, BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT2_OUTPUTS, BUS_-CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT3_OUTPUTS, BUS_CMD_-DRIVER_DEACTIVATE_ALL_SUBMENU_ANT4_OUTPUTS, BUS_CMD_DRIVER_-DEACTIVATE_SUBMENU_ANT1_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_-SUBMENU_ANT2_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT3_-OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT4_OUTPUT, bus_-get_address(), BUS_MESSAGE_FLAGS_NEED_ACK, current_activated_sub_outputs, current_activated_sub_outputs_length, internal_comm_add_tx_message(), OUTPUT_-ADDR_DELIMITER, struct_sub_menu_array::output_str_dir, and struct_sub_menu_-array::output_str_dir_length.

Referenced by main(), and sub_menu_activate_all().

6.73.2.9 void sub_menu_set_current_pos (unsigned char *ant_index*, unsigned char *new_pos*)

Set the current sub menu option.

Parameters:

ant_index The antenna index (0-3)

new_pos The position we wish to chose

Definition at line 95 of file sub_menu.c.

References curr_option_selected.

Referenced by sub_menu_pos_down(), and sub_menu_pos_up().

6.74 front_panel/sub_menu.h File Reference

Antenna sub menu functions.

`#include "main.h"`

Classes

- struct struct_sub_menu_array

    Struct of a sub menu with the type array.

Functions

- void sub_menu_load (unsigned char band_index)

    Load a set of sub menu from the EEPROM for a specific band.

- unsigned char ∗ sub_menu_get_text (unsigned char ant_index, unsigned char pos)

    Get the text for the sub menu.

- unsigned char sub_menu_get_count (void)

    Get the number of antennas which has got a sub menu configured.

- unsigned char sub_menu_get_current_pos (unsigned char ant_index)

    Get the current position of the sub menu cursor.

- void sub_menu_set_current_pos (unsigned char ant_index, unsigned char new_pos)

    Set the current sub menu option.

- unsigned char sub_menu_get_type (unsigned char ant_index)

    Get the sub menu type of an antenna.

- void sub_menu_pos_down (unsigned char ant_index)

    This function should be called when we wish to decrease the selected sub menu option.

- void sub_menu_pos_up (unsigned char ant_index)

    This function should be called when we wish to increase the selected sub menu option.

- void sub_menu_send_data_to_bus (unsigned char ant_index, unsigned char pos)

    Send the output string for the sub menu position to the bus.

- void sub_menu_deactivate_all (void)

    Will deactivate all currently selected outputs which has been sent out on the bus.

- void sub_menu_activate_all (void)

    This function will go through the sub menus and if there is one configured it will activae its default option which is index 0.

6.74.1 Detailed Description

Antenna sub menu functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-04-28

```
#include "front_panel/sub_menu.h"
```

Definition in file sub_menu.h.

6.74.2 Function Documentation

6.74.2.1 unsigned char sub_menu_get_count (void)

Get the number of antennas which has got a sub menu configured.

Returns:

The number of antennas which has got sub menus, (0-4)

Definition at line 101 of file sub_menu.c.

References antenna_ctrl_get_sub_menu_type(), and SUBMENU_NONE.

Referenced by event_sub_button_pressed().

6.74.2.2 unsigned char sub_menu_get_current_pos (unsigned char *ant_index*)

Get the current position of the sub menu cursor.

Parameters:

ant_index The antenna index, (0-3)

Returns:

The cursor position of the sub menu

Definition at line 88 of file sub_menu.c.

References curr_option_selected.

Referenced by display_rotator_directions(), display_show_sub_menu(), main(), sub_menu_-pos_down(), and sub_menu_pos_up().

6.74.2.3 unsigned char∗ sub_menu_get_text (unsigned char *ant_index*, unsigned char *pos*)

Get the text for the sub menu.

Parameters:

ant_index The antenna index we wish to get the antenna text for

pos Which sub menu position to show

Returns:

Returns the text of the sub menu antenna index

Definition at line 77 of file sub_menu.c.

References antenna_ctrl_get_sub_menu_type(), and SUBMENU_VERT_ARRAY.

Referenced by display_rotator_directions(), and display_show_sub_menu().

6.74.2.4 unsigned char sub_menu_get_type (unsigned char *ant_index*)

Get the sub menu type of an antenna.

Parameters:

ant_index The antenna we wish to get the sub menu type of, (0-3)

Returns:

The sub meny type of the antenna

Definition at line 114 of file sub_menu.c.

References antenna_ctrl_get_sub_menu_type().

Referenced by event_pulse_sensor_down(), event_pulse_sensor_up(), sub_menu_pos_down(), and sub_menu_pos_up().

6.74.2.5 void sub_menu_load (unsigned char *band_index*)

Load a set of sub menu from the EEPROM for a specific band.

Parameters:

band_index Which we band

Definition at line 60 of file sub_menu.c.

References antenna_ctrl_get_sub_menu_type(), curr_option_selected, eeprom_get_ant_-sub_menu_array_structure(), and SUBMENU_VERT_ARRAY.

Referenced by band_ctrl_load_band().

6.74.2.6 void sub_menu_pos_down (unsigned char *ant_index*)

This function should be called when we wish to decrease the selected sub menu option.

Parameters:

ant_index Which antenna we wish to decrease the sub menu position of

Definition at line 120 of file sub_menu.c.

References sub_menu_get_current_pos(), sub_menu_get_type(), sub_menu_set_current_-pos(), and SUBMENU_VERT_ARRAY.

Referenced by event_pulse_sensor_down().

6.74.2.7 void sub_menu_pos_up (unsigned char *ant_index*)

This function should be called when we wish to increase the selected sub menu option.

Parameters:

ant_index Which antenna we wish to increase the sub menu position of

Definition at line 131 of file sub_menu.c.

References sub_menu_get_current_pos(), sub_menu_get_type(), sub_menu_set_current_-pos(), and SUBMENU_VERT_ARRAY.

Referenced by event_pulse_sensor_up().

6.74.2.8 void sub_menu_send_data_to_bus (unsigned char *ant_index*, unsigned char *pos*)

Send the output string for the sub menu position to the bus.

Parameters:

ant_index The index of the antenna you wish to send the string of

pos The sub menu position we wish to send the output str of

Definition at line 144 of file sub_menu.c.

References antenna_ctrl_deactivate_outputs(), bus_add_tx_message(), BUS_-CMD_DRIVER_ACTIVATE_SUBMENU_ANT1_OUTPUT, BUS_CMD_DRIVER_-ACTIVATE_SUBMENU_ANT2_OUTPUT, BUS_CMD_DRIVER_ACTIVATE_-SUBMENU_ANT3_OUTPUT, BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT4_-OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT1_OUTPUTS, BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT2_OUTPUTS, BUS_-CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT3_OUTPUTS, BUS_CMD_-DRIVER_DEACTIVATE_ALL_SUBMENU_ANT4_OUTPUTS, BUS_CMD_DRIVER_-DEACTIVATE_SUBMENU_ANT1_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_-SUBMENU_ANT2_OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT3_-OUTPUT, BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT4_OUTPUT, bus_-get_address(), BUS_MESSAGE_FLAGS_NEED_ACK, current_activated_sub_outputs, current_activated_sub_outputs_length, internal_comm_add_tx_message(), OUTPUT_-ADDR_DELIMITER, struct_sub_menu_array::output_str_dir, and struct_sub_menu_-array::output_str_dir_length.

Referenced by main(), and sub_menu_activate_all().

6.74.2.9 void sub_menu_set_current_pos (unsigned char *ant_index*, unsigned char *new_pos*)

Set the current sub menu option.

Parameters:

ant_index The antenna index (0-3)

new_pos The position we wish to chose

Definition at line 95 of file sub_menu.c.

References curr_option_selected.

Referenced by sub_menu_pos_down(), and sub_menu_pos_up().

6.75 front_panel/usart.c File Reference

USART routines.

`#include <avr/io.h>`

`#include <avr/interrupt.h>`

`#include <stdio.h>`

`#include <string.h>`

Functions

- void usart0_init (unsigned int baudrate)

  Initiliaze the USART0 for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

- void usart0_transmit (char data)

  Send a character to the USART0 Send a single character to the USART used for the communication bus.

- unsigned char usart0_sendstring (char ∗data, unsigned char length)

  Send a string of characters to the USART0 Send a string of characters to the USART used for the communication bus.

- unsigned char usart0_receive (void)

  Retrieve one character from the USART0 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart0_receive_loopback (void)

  The USART0 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char poll_usart0_receive (void)

  Retrieve one character from the USART0 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

- void usart1_init (unsigned int baudrate, unsigned char stopbits)

  Initiliaze the USART1 for the interface towards the computer This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

- unsigned char usart1_transmit (char data)

  Send a character to the USART1 Send a single character to the USART used for the communication bus.

- unsigned char usart1_sendstring (char ∗data, unsigned char length)

  Send a string of characters to the USART1 Send a string of characters to the USART used for the communication bus.

- unsigned char usart1_receive (void)

Retrieve one character from the USART1 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart1_receive_loopback (void)

  The USART1 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char poll_usart1_receive (void)

  Retrieve one character from the USART1 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

- void usart3_init (unsigned int baudrate, unsigned char stopbits)

  Initiliaze the USART3 for the radio interface This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

- unsigned char usart3_transmit (char data)

  Send a character to the USART3 Send a single character to the USART used for the communication bus.

- unsigned char usart3_sendstring (char *data, unsigned char length)

  Send a string of characters to the USART3 Send a string of characters to the USART used for the communication bus.

- unsigned char usart3_receive (void)

  Retrieve one character from the USART3 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart3_receive_loopback (void)

  The USART3 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char poll_usart3_receive (void)

  Retrieve one character from the USART3 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

6.75.1 Detailed Description

USART routines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/usart.c"
```

Definition in file usart.c.

6.75.2 Function Documentation

6.75.2.1 unsigned char poll_usart0_receive (void)

Retrieve one character from the USART0 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 100 of file usart.c.

6.75.2.2 unsigned char poll_usart1_receive (void)

Retrieve one character from the USART1 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 185 of file usart.c.

6.75.2.3 unsigned char poll_usart3_receive (void)

Retrieve one character from the USART3 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 270 of file usart.c.

6.75.2.4 void usart0_init (unsigned int *baudrate*)

Initiliaze the USART0 for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baudrate The baudrate param from the ATMEGA2560 datasheet.

Definition at line 34 of file usart.c.

Referenced by init_usart(), and main().

6.75.2.5 unsigned char usart0_receive (void)

Retrieve one character from the USART0 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 73 of file usart.c.

6.75.2.6 unsigned char usart0_receive_loopback (void)

The USART0 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 85 of file usart.c.

Referenced by init_usart().

6.75.2.7 unsigned char usart0_sendstring (char ∗ *data*, unsigned char *length*)

Send a string of characters to the USART0 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Definition at line 60 of file usart.c.

6.75.2.8 void usart0_transmit (char *data*)

Send a character to the USART0 Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Definition at line 48 of file usart.c.

Referenced by init_usart(), main(), usart0_receive_loopback(), usart0_sendstring(), usart1_-receive_loopback(), and usart3_receive_loopback().

6.75.2.9 void usart1_init (unsigned int *baudrate*, unsigned char *stopbits*)

Initiliaze the USART1 for the interface towards the computer This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baudrate The baudrate param from the ATMEGA2560 datasheet.

stopbits The number of stopbits.

Definition at line 112 of file usart.c.

Referenced by init_usart_computer(), main(), and radio_interface_init().

6.75.2.10 unsigned char usart1_receive (void)

Retrieve one character from the USART1 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 158 of file usart.c.

6.75.2.11 unsigned char usart1_receive_loopback (void)

The USART1 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 170 of file usart.c.

Referenced by init_usart_computer(), and main().

6.75.2.12 unsigned char usart1_sendstring (char ∗ *data*, unsigned char *length*)

Send a string of characters to the USART1 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Definition at line 145 of file usart.c.

6.75.2.13 unsigned char usart1_transmit (char *data*)

Send a character to the USART1 Send a single character to the USART used for the communication bus.

Send a character to the USART Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Definition at line 132 of file usart.c.

Referenced by computer_interface_send_data(), init_usart_computer(), ISR(), main(), usart1_receive_loopback(), and usart1_sendstring().

6.75.2.14 void usart3_init (unsigned int *baudrate*, unsigned char *stopbits*)

Initiliaze the USART3 for the radio interface This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baudrate The baudrate param from the ATMEGA2560 datasheet.
stopbits The number of stopbits.

Definition at line 197 of file usart.c.

Referenced by radio_interface_init().

6.75.2.15 unsigned char usart3_receive (void)

Retrieve one character from the USART3 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 243 of file usart.c.

6.75.2.16 unsigned char usart3_receive_loopback (void)

The USART3 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 255 of file usart.c.

References usart0_transmit().

6.75.2.17 unsigned char usart3_sendstring (char * *data*, unsigned char *length*)

Send a string of characters to the USART3 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send
length The length of the string you wish to send

Definition at line 230 of file usart.c.

References usart3_transmit().

6.75.2.18 unsigned char usart3_transmit (char *data*)

Send a character to the USART3 Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Definition at line 217 of file usart.c.

Referenced by ISR(), radio_poll_status(), and usart3_sendstring().

6.76 motherboard/usart.c File Reference

Motherboard USART routines.

`#include <avr/io.h>`

`#include <avr/interrupt.h>`

`#include <stdio.h>`

`#include <string.h>`

Functions

- unsigned char usart1_transmit (char data)

  Send a character to the USART Send a single character to the USART used for the communication bus.

- void usart0_init (unsigned int baudrate)

  Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA128 baudrate setting.

- unsigned char usart0_transmit (char data)

  Send a character to the USART Send a single character to the USART used for the communication bus.

- unsigned char usart0_sendstring (char ∗data, unsigned char length)

  Send a string of characters to the USART Send a string of characters to the USART used for the communication bus.

- unsigned char usart0_receive (void)

  Retrieve one character from the USART Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart0_receive_loopback (void)

  The USART recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char poll_usart0_receive (void)

  Retrieve one character from the USART Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

- void usart1_init (unsigned int baudrate)

  Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA128 baudrate setting.

- unsigned char usart1_sendstring (char ∗data, unsigned char length)

  Send a string of characters to the USART Send a string of characters to the USART used for the communication bus.

- unsigned char usart1_receive (void)

Retrieve one character from the USART Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart1_receive_loopback (void)

  The USART recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char poll_usart1_receive (void)

  Retrieve one character from the USART Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

6.76.1 Detailed Description

Motherboard USART routines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "motherboard/usart.c"
```

Definition in file usart.c.

6.76.2 Function Documentation

6.76.2.1 unsigned char poll_usart0_receive (void)

Retrieve one character from the USART Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Retrieve one character from the USART0 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 101 of file usart.c.

6.76.2.2 unsigned char poll_usart1_receive (void)

Retrieve one character from the USART Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Retrieve one character from the USART1 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 179 of file usart.c.

6.76.2.3 void usart0_init (unsigned int *baudrate*)

Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA128 baudrate setting.

Initiliaze the USART0 for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baud The baudrate param from the ATMEGA32 datasheet.

Definition at line 34 of file usart.c.

6.76.2.4 unsigned char usart0_receive (void)

Retrieve one character from the USART Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Retrieve one character from the USART0 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 74 of file usart.c.

6.76.2.5 unsigned char usart0_receive_loopback (void)

The USART recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

The USART0 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 86 of file usart.c.

References usart0_transmit().

6.76.2.6 unsigned char usart0_sendstring (char * *data*, unsigned char *length*)

Send a string of characters to the USART Send a string of characters to the USART used for the communication bus.

Send a string of characters to the USART0 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send *length* The length of the string you wish to send

Definition at line 61 of file usart.c.

References usart0_transmit().

6.76.2.7 unsigned char usart0_transmit (char *data*)

Send a character to the USART Send a single character to the USART used for the communication bus.

Send a character to the USART0 Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Definition at line 48 of file usart.c.

6.76.2.8 void usart1_init (unsigned int *baudrate*)

Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA128 baudrate setting.

Parameters:

baudrate The baudrate param from the ATMEGA32 datasheet.

Definition at line 112 of file usart.c.

6.76.2.9 unsigned char usart1_receive (void)

Retrieve one character from the USART Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Retrieve one character from the USART1 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 152 of file usart.c.

6.76.2.10 unsigned char usart1_receive_loopback (void)

The USART recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

The USART1 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 164 of file usart.c.

References usart0_transmit(), and usart1_transmit().

6.76.2.11 unsigned char usart1_sendstring (char * *data*, unsigned char *length*)

Send a string of characters to the USART Send a string of characters to the USART used for the communication bus.

Send a string of characters to the USART1 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Definition at line 139 of file usart.c.

References usart1_transmit().

6.76.2.12 unsigned char usart1_transmit (char *data*)

Send a character to the USART Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Send a character to the USART Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Definition at line 132 of file usart.c.

Referenced by computer_interface_send_data(), init_usart_computer(), ISR(), main(), usart1_receive_loopback(), and usart1_sendstring().

6.77 front_panel/usart.h File Reference

USART routines.

`#include "../global.h"`

Functions

- unsigned char poll_usart0_receive (void)

  Retrieve one character from the USART0 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

- void usart0_init (unsigned int baudrate)

  Initiliaze the USART0 for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

- void usart0_transmit (char data)

  Send a character to the USART0 Send a single character to the USART used for the communication bus.

- unsigned char usart0_receive (void)

  Retrieve one character from the USART0 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart0_receive_loopback (void)

  The USART0 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char usart0_sendstring (char ∗data, unsigned char length)

  Send a string of characters to the USART0 Send a string of characters to the USART used for the communication bus.

- unsigned char poll_usart1_receive (void)

  Retrieve one character from the USART1 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

- void usart1_init (unsigned int baudrate, unsigned char stopbits)

  Initiliaze the USART1 for the interface towards the computer This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

- unsigned char **usart1_transmit** (unsigned char data)
- unsigned char usart1_receive (void)

  Retrieve one character from the USART1 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart1_receive_loopback (void)

  The USART1 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char usart1_sendstring (char ∗data, unsigned char length)

Send a string of characters to the USART1 Send a string of characters to the USART used for the communication bus.

- unsigned char poll_usart3_receive (void)

  Retrieve one character from the USART3 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

- void usart3_init (unsigned int baudrate, unsigned char stopbits)

  Initiliaze the USART3 for the radio interface This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

- unsigned char **usart3_transmit** (unsigned char data)
- unsigned char usart3_receive (void)

  Retrieve one character from the USART3 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart3_receive_loopback (void)

  The USART3 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char usart3_sendstring (char ∗data, unsigned char length)

  Send a string of characters to the USART3 Send a string of characters to the USART used for the communication bus.

6.77.1 Detailed Description

USART routines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "front_panel/usart.h"
```

Definition in file usart.h.

6.77.2 Function Documentation

6.77.2.1 unsigned char poll_usart0_receive (void)

Retrieve one character from the USART0 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Retrieve one character from the USART0 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 100 of file usart.c.

6.77.2.2 unsigned char poll_usart1_receive (void)

Retrieve one character from the USART1 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Retrieve one character from the USART1 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 185 of file usart.c.

6.77.2.3 unsigned char poll_usart3_receive (void)

Retrieve one character from the USART3 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 270 of file usart.c.

6.77.2.4 void usart0_init (unsigned int *baudrate*)

Initiliaze the USART0 for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baudrate The baudrate param from the ATMEGA2560 datasheet.

Initiliaze the USART0 for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baud The baudrate param from the ATMEGA32 datasheet.

Definition at line 34 of file usart.c.

6.77.2.5 unsigned char usart0_receive (void)

Retrieve one character from the USART0 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Retrieve one character from the USART0 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 73 of file usart.c.

6.77.2.6 unsigned char usart0_receive_loopback (void)

The USART0 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

The USART0 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 85 of file usart.c.

6.77.2.7 unsigned char usart0_sendstring (char ∗ *data*, unsigned char *length*)

Send a string of characters to the USART0 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Send a string of characters to the USART0 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Definition at line 60 of file usart.c.

6.77.2.8 void usart0_transmit (char *data*)

Send a character to the USART0 Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Send a character to the USART0 Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Definition at line 48 of file usart.c.

Referenced by init_usart(), main(), usart0_receive_loopback(), usart0_sendstring(), usart1_-receive_loopback(), and usart3_receive_loopback().

6.77.2.9 void usart1_init (unsigned int *baudrate*, unsigned char *stopbits*)

Initiliaze the USART1 for the interface towards the computer This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baudrate The baudrate param from the ATMEGA2560 datasheet.

stopbits The number of stopbits.

Definition at line 112 of file usart.c.

Referenced by init_usart_computer(), main(), and radio_interface_init().

6.77.2.10 unsigned char usart1_receive (void)

Retrieve one character from the USART1 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Retrieve one character from the USART1 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 158 of file usart.c.

6.77.2.11 unsigned char usart1_receive_loopback (void)

The USART1 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

The USART1 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 170 of file usart.c.

6.77.2.12 unsigned char usart1_sendstring (char ∗ *data*, unsigned char *length*)

Send a string of characters to the USART1 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Send a string of characters to the USART1 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Definition at line 145 of file usart.c.

6.77.2.13 void usart3_init (unsigned int *baudrate*, unsigned char *stopbits*)

Initiliaze the USART3 for the radio interface This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baudrate The baudrate param from the ATMEGA2560 datasheet.

stopbits The number of stopbits.

Definition at line 197 of file usart.c.

Referenced by radio_interface_init().

6.77.2.14 unsigned char usart3_receive (void)

Retrieve one character from the USART3 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 243 of file usart.c.

6.77.2.15 unsigned char usart3_receive_loopback (void)

The USART3 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 255 of file usart.c.

References usart0_transmit().

6.77.2.16 unsigned char usart3_sendstring (char * *data*, unsigned char *length*)

Send a string of characters to the USART3 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Definition at line 230 of file usart.c.

References usart3_transmit().

6.78 motherboard/usart.h File Reference

Motherboard USART routines.

`#include "../global.h"`

Functions

- unsigned char poll_usart0_receive (void)

  Retrieve one character from the USART0 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

- void usart0_init (unsigned int baudrate)

  Initiliaze the USART0 for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

- unsigned char **usart0_transmit** (unsigned char data)
- unsigned char usart0_receive (void)

  Retrieve one character from the USART0 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart0_receive_loopback (void)

  The USART0 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char usart0_sendstring (char ∗data, unsigned char length)

  Send a string of characters to the USART0 Send a string of characters to the USART used for the communication bus.

- unsigned char poll_usart1_receive (void)

  Retrieve one character from the USART1 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

- void usart1_init (unsigned int baudrate)

  Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA128 baudrate setting.

- unsigned char **usart1_transmit** (unsigned char data)
- unsigned char usart1_receive (void)

  Retrieve one character from the USART1 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char usart1_receive_loopback (void)

  The USART1 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char usart1_sendstring (char ∗data, unsigned char length)

  Send a string of characters to the USART1 Send a string of characters to the USART used for the communication bus.

6.78.1 Detailed Description

Motherboard USART routines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "motherboard/usart.h"
```

Definition in file usart.h.

6.78.2 Function Documentation

6.78.2.1 unsigned char poll_usart0_receive (void)

Retrieve one character from the USART0 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Retrieve one character from the USART0 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 100 of file usart.c.

6.78.2.2 unsigned char poll_usart1_receive (void)

Retrieve one character from the USART1 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Retrieve one character from the USART1 Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 185 of file usart.c.

6.78.2.3 void usart0_init (unsigned int *baudrate*)

Initiliaze the USART0 for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baudrate The baudrate param from the ATMEGA2560 datasheet.

Initiliaze the USART0 for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA2560 baudrate setting.

Parameters:

baud The baudrate param from the ATMEGA32 datasheet.

Definition at line 34 of file usart.c.

Referenced by init_usart(), and main().

6.78.2.4 unsigned char usart0_receive (void)

Retrieve one character from the USART0 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Retrieve one character from the USART0 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 73 of file usart.c.

6.78.2.5 unsigned char usart0_receive_loopback (void)

The USART0 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

The USART0 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 85 of file usart.c.

References usart0_transmit().

Referenced by init_usart().

6.78.2.6 unsigned char usart0_sendstring (char * *data*, unsigned char *length*)

Send a string of characters to the USART0 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send
length The length of the string you wish to send

Send a string of characters to the USART0 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send
length The length of the string you wish to send

Definition at line 60 of file usart.c.

References usart0_transmit().

6.78.2.7 void usart1_init (unsigned int *baudrate*)

Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the ATMEGA128 baudrate setting.

Parameters:

baudrate The baudrate param from the ATMEGA32 datasheet.

Definition at line 112 of file usart.c.

6.78.2.8 unsigned char usart1_receive (void)

Retrieve one character from the USART1 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Retrieve one character from the USART1 Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 158 of file usart.c.

6.78.2.9 unsigned char usart1_receive_loopback (void)

The USART1 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

The USART1 recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 170 of file usart.c.

References usart0_transmit(), and usart1_transmit().

Referenced by init_usart_computer(), and main().

6.78.2.10 unsigned char usart1_sendstring (char * *data*, unsigned char *length*)

Send a string of characters to the USART1 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Send a string of characters to the USART1 Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Definition at line 145 of file usart.c.

References usart1_transmit().

6.79 i2c.c File Reference

I2C interface using AVR Two-Wire Interface (TWI) hardware.

`#include <avr/io.h>`

`#include <avr/interrupt.h>`

`#include <stdio.h>`

`#include "i2c.h"`

Functions

- void i2c_init (void)

  Initialize I2C (TWI) interface.

- void i2cSetBitrate (unsigned int bitrateKHz)

  Set the I2C transaction bitrate (in KHz).

- void i2cSetLocalDeviceAddr (unsigned char deviceAddr, unsigned char genCallEn)

  Set the local (AVR processor's) I2C device address.

- void i2cSetSlaveReceiveHandler (void(∗i2cSlaveRx_func)(unsigned char receiveDataLength, unsigned char ∗recieveData))

  Set the user function which handles receiving (incoming) data as a slave.

- void i2cSetSlaveTransmitHandler (unsigned char(∗i2cSlaveTx_func)(unsigned char transmitDataLengthMax, unsigned char ∗transmitData))

  Set the user function which handles transmitting (outgoing) data as a slave.

- void i2cSendStart (void)

  Send an I2C start condition in Master mode.

- void i2cSendStop (void)

  Send an I2C stop condition in Master mode.

- void i2cWaitForComplete (void)

  Wait for current I2C operation to complete.

- void i2cSendByte (unsigned char data)

  Send an (address|R/W) combination or a data byte over I2C.

- void i2cReceiveByte (unsigned char ackFlag)

  Receive a data byte over I2C.

- unsigned char i2cGetReceivedByte (void)

  Pick up the data that was received with i2cReceiveByte().

- unsigned char i2cGetStatus (void)

  Get current I2c bus status from TWSR.

- void i2cMasterSend (unsigned char deviceAddr, unsigned char length, unsigned char ∗data)

  send I2C data to a device on the bus

- void i2cMasterReceive (unsigned char deviceAddr, unsigned char length, unsigned char ∗data)

  receive I2C data from a device on the bus

- unsigned char i2cMasterSendNI (unsigned char deviceAddr, unsigned char length, unsigned char ∗data)

  send I2C data to a device on the bus (non-interrupt based)

- unsigned char i2cMasterReceiveNI (unsigned char deviceAddr, unsigned char length, unsigned char ∗data)

  receive I2C data from a device on the bus (non-interrupt based)

- SIGNAL (SIG_2WIRE_SERIAL)

  I2C (TWI) interrupt service routine.

- eI2cStateType i2cGetState (void)

  Get the current high-level state of the I2C interface.

6.79.1 Detailed Description

I2C interface using AVR Two-Wire Interface (TWI) hardware.

Author:

Pascal Stang and Mikael Larsmark, SM2WMV

Date:

2008-04-13

Definition in file i2c.c.

6.80 i2c.h File Reference

I2C interface using AVR Two-Wire Interface (TWI) hardware.

`#include "global.h"`

`#include "i2cconf.h"`

Defines

- #define **TW_START** 0x08
- #define **TW_REP_START** 0x10
- #define **TW_MT_SLA_ACK** 0x18
- #define **TW_MT_SLA_NACK** 0x20
- #define **TW_MT_DATA_ACK** 0x28
- #define **TW_MT_DATA_NACK** 0x30
- #define **TW_MT_ARB_LOST** 0x38
- #define **TW_MR_ARB_LOST** 0x38
- #define **TW_MR_SLA_ACK** 0x40
- #define **TW_MR_SLA_NACK** 0x48
- #define **TW_MR_DATA_ACK** 0x50
- #define **TW_MR_DATA_NACK** 0x58
- #define **TW_ST_SLA_ACK** 0xA8
- #define **TW_ST_ARB_LOST_SLA_ACK** 0xB0
- #define **TW_ST_DATA_ACK** 0xB8
- #define **TW_ST_DATA_NACK** 0xC0
- #define **TW_ST_LAST_DATA** 0xC8
- #define **TW_SR_SLA_ACK** 0x60
- #define **TW_SR_ARB_LOST_SLA_ACK** 0x68
- #define **TW_SR_GCALL_ACK** 0x70
- #define **TW_SR_ARB_LOST_GCALL_ACK** 0x78
- #define **TW_SR_DATA_ACK** 0x80
- #define **TW_SR_DATA_NACK** 0x88
- #define **TW_SR_GCALL_DATA_ACK** 0x90
- #define **TW_SR_GCALL_DATA_NACK** 0x98
- #define **TW_SR_STOP** 0xA0
- #define **TW_NO_INFO** 0xF8
- #define **TW_BUS_ERROR** 0x00
- #define **TWCR_CMD_MASK** 0x0F
- #define **TWSR_STATUS_MASK** 0xF8
- #define **I2C_OK** 0x00
- #define **I2C_ERROR_NODEV** 0x01

Enumerations

- enum **eI2cStateType** {

  I2C_IDLE = 0, **I2C_BUSY** = 1, **I2C_MASTER_TX** = 2, **I2C_MASTER_RX** = 3,

  I2C_SLAVE_TX = 4, **I2C_SLAVE_RX** = 5 }

Functions

- void i2c_init (void)

  Initialize I2C (TWI) interface.

- void i2cSetBitrate (unsigned int bitrateKHz)

  Set the I2C transaction bitrate (in KHz).

- void i2cSetLocalDeviceAddr (unsigned char deviceAddr, unsigned char genCallEn)

  Set the local (AVR processor's) I2C device address.

- void i2cSetSlaveReceiveHandler (void(∗i2cSlaveRx_func)(unsigned char receiveDataLength, unsigned char ∗recieveData))

  Set the user function which handles receiving (incoming) data as a slave.

- void i2cSetSlaveTransmitHandler (unsigned char(∗i2cSlaveTx_func)(unsigned char transmitDataLengthMax, unsigned char ∗transmitData))

  Set the user function which handles transmitting (outgoing) data as a slave.

- void i2cSendStart (void)

  Send an I2C start condition in Master mode.

- void i2cSendStop (void)

  Send an I2C stop condition in Master mode.

- void i2cWaitForComplete (void)

  Wait for current I2C operation to complete.

- void i2cSendByte (unsigned char data)

  Send an (address|R/W) combination or a data byte over I2C.

- void i2cReceiveByte (unsigned char ackFlag)

  Receive a data byte over I2C.

- unsigned char i2cGetReceivedByte (void)

  Pick up the data that was received with i2cReceiveByte().

- unsigned char i2cGetStatus (void)

  Get current I2c bus status from TWSR.

- void i2cMasterSend (unsigned char deviceAddr, unsigned char length, unsigned char ∗data)

  send I2C data to a device on the bus

- void i2cMasterReceive (unsigned char deviceAddr, unsigned char length, unsigned char ∗data)

  receive I2C data from a device on the bus

- unsigned char i2cMasterSendNI (unsigned char deviceAddr, unsigned char length, unsigned char ∗data) *send I2C data to a device on the bus (non-interrupt based)*

- unsigned char i2cMasterReceiveNI (unsigned char deviceAddr, unsigned char length, unsigned char *data)

  receive I2C data from a device on the bus (non-interrupt based)

- eI2cStateType i2cGetState (void)

  Get the current high-level state of the I2C interface.

6.80.1 Detailed Description

I2C interface using AVR Two-Wire Interface (TWI) hardware.

Author:

Pascal Stang and Mikael Larsmark, SM2WMV

Date:

2008-04-13

Definition in file i2c.h.

6.81 i2cconf.h File Reference

I2C (TWI) interface configuration.

Defines

- #define I2C_SEND_DATA_BUFFER_SIZE 0x20

  The size of the transmit buffer.

- #define I2C_RECEIVE_DATA_BUFFER_SIZE 0x20

  The size of the receive buffer.

6.81.1 Detailed Description

I2C (TWI) interface configuration.

Author:

Pascal Stang and Mikael Larsmark, SM2WMV

Date:

2008-04-13

Definition in file i2cconf.h.

6.81.2 Define Documentation

6.81.2.1 #define I2C_SEND_DATA_BUFFER_SIZE 0x20

The size of the transmit buffer.

define I2C data buffer sizes These buffers are used in interrupt-driven Master sending and receiving, and in slave sending and receiving. They must be large enough to store the largest I2C packet you expect to send and receive, respectively.

Definition at line 30 of file i2cconf.h.

Referenced by SIGNAL().

6.82 internal_comm.c File Reference

The internal communication routines.

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include "internal_comm.h"
#include "internal_comm_rx_queue.h"
#include "internal_comm_tx_queue.h"
```

Functions

- void internal_comm_init (void(*func_ptr_rx)(UC_MESSAGE), void(*func_ptr_tx)(char))

  Initialize the internal communication.
- void internal_comm_reset_rx (void)

  Will reset the RX variables.
- unsigned char internal_comm_poll_rx_queue (void)

  Polls the RX queue in the internal communication and calls the function defined in internal_comm_init.
- unsigned char internal_comm_poll_tx_queue (void)

  Polls the TX queue in the internal communication and sends the data if there is a message in the queue.
- void internal_comm_send_ack (void)

  Send an ACK message to the internal communication uart.
- void internal_comm_send_nack (void)

  Send a NACK message to the internal communication uart.
- void internal_comm_send_message (UC_MESSAGE tx_message)

  Send a message to the internal communication uart.
- void internal_comm_add_tx_message (unsigned char command, unsigned char length, char *data)

  Add a message to the transmit queue.
- void internal_comm_resend (void)

  Will trigger a resend of the last message.
- ISR (ISR_INTERNAL_COMM_USART_RECV)

  Interrupt when a byte has been received from the UART.

- ISR (ISR_INTERNAL_COMM_USART_DATA)
  Interrupt when data has been received from the UART.
- void internal_comm_1ms_timer (void)
  Function which should be called each ms.

Variables

- struct_uc_com uc_com
  The uc_com struct.
- UC_MESSAGE uc_new_message
  Where we save any new uc_comm message.
- unsigned char prev_data = 0
  The previous data.
- unsigned char counter_tx_timeout = 0
  Counter which keep track of when we last did a transmission.
- unsigned char counter_rx_timeout = 0
  Counter which keeps track of when we last did receive a character.
- unsigned char resend_count = 0
  The number of times the last message has been resent.
- unsigned char msg_not_acked = 0
  Flag that the message has yet not been acked.
- void(∗ f_ptr_rx)(UC_MESSAGE)
  Function to be called when a message is recieved and should be parsed/executed.
- void(∗ f_ptr_tx)(char)
  Function to be called when we wish to send a message.

6.82.1 Detailed Description

The internal communication routines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "internal_comm.c"
```

Definition in file internal_comm.c.

6.82.2 Function Documentation

6.82.2.1 void internal_comm_add_tx_message (unsigned char *command*, unsigned char *length*, char * *data*)

Add a message to the transmit queue.

Parameters:

command The command we wish to perform

length The length of the data field

data The data we wish to send

Definition at line 166 of file internal_comm.c.

References UC_MESSAGE::checksum, UC_MESSAGE::cmd, UC_MESSAGE::data, int_-comm_tx_queue_add(), and UC_MESSAGE::length.

Referenced by antenna_ctrl_deactivate_outputs(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_send_rx_ant_band_data_to_bus(), antenna_ctrl_send_rx_ant_data_-to_bus(), band_ctrl_send_band_data_to_bus(), computer_interface_parse_data(), main(), parse_internal_comm_message(), ps2_process_key(), radio_poll_status(), shutdown_device(), and sub_menu_send_data_to_bus().

6.82.2.2 void internal_comm_init (void(*)(UC_MESSAGE) *func_ptr_rx*, void(*)(char) *func_ptr_tx*)

Initialize the internal communication.

Parameters:

func_ptr_rx The function you wish to call when a new message has been recieved and should be parsed

func_ptr_tx The function used to send data to the hardware handeling the data transmission

Definition at line 68 of file internal_comm.c.

References struct_uc_com::char_count, struct_uc_com::checksum, f_ptr_rx, f_ptr_tx, struct_uc_com::flags, int_comm_rx_queue_dropall(), and int_comm_tx_queue_dropall().

Referenced by main().

6.82.2.3 unsigned char internal_comm_poll_rx_queue (void)

Polls the RX queue in the internal communication and calls the function defined in internal_-comm_init.

Returns:

1 if a message was in the buffer and got parsed, 0 if not

Definition at line 91 of file internal_comm.c.

References f_ptr_rx, int_comm_rx_queue_drop(), int_comm_rx_queue_get(), and int_-comm_rx_queue_is_empty().

Referenced by main().

6.82.2.4 unsigned char internal_comm_poll_tx_queue (void)

Polls the TX queue in the internal communication and sends the data if there is a message in the queue.

Returns:

1 if a message was in the buffer and got sent, 0 if not

Definition at line 105 of file internal_comm.c.

References int_comm_tx_queue_get(), int_comm_tx_queue_is_empty(), internal_comm_-send_message(), and msg_not_acked.

Referenced by main().

6.82.2.5 void internal_comm_send_message (UC_MESSAGE *tx_message*)

Send a message to the internal communication uart.

Parameters:

tx_message The message we wish to send

Definition at line 147 of file internal_comm.c.

References UC_MESSAGE::checksum, UC_MESSAGE::cmd, counter_tx_timeout, UC_-MESSAGE::data, f_ptr_tx, UC_MESSAGE::length, UC_COMM_MSG_POSTAMBLE, and UC_COMM_MSG_PREAMBLE.

Referenced by internal_comm_poll_tx_queue(), and internal_comm_resend().

6.83 internal_comm.h File Reference

The internal communication routines.

Classes

- struct UC_MESSAGE
- struct struct_uc_com

Defines

- #define UC_PREAMBLE_FOUND 0

  if the device is a motherboard we need to set the proper USARTs used

- #define UC_MESSAGE_IN_BUFFER 1

  Flag that a message is in the buffer.

- #define UC_SIZE_FIXED 5

  Size of UC MESSAGE fixed part.

- #define UC_COMM_MSG_PREAMBLE 0xFE
- #define UC_COMM_MSG_POSTAMBLE 0xFD
- #define UC_COMM_MSG_ACK 0xFB
- #define UC_COMM_MSG_NACK 0xFA
- #define UC_SERIAL_RX_BUFFER_LENGTH 20

  The length of the serial rx buffer used for communication between the uCs.

- #define UC_MESSAGE_DATA_SIZE 15

  The size the data sent between the two devices can be maximum.

- #define UC_COMM_RX_TIMEOUT 3

  After this many ms it will reset the rx flags (in ms).

- #define UC_COMM_TX_TIMEOUT 10

  After this many ms a resend will occur if a message has not been acked (in ms).

- #define UC_COMM_RESEND_COUNT 5

  Number of resends that is allowed.

Functions

- void internal_comm_init (void(*func_ptr_rx)(UC_MESSAGE), void(*func_ptr_-tx)(char))

  Initialize the internal communication.

- unsigned char internal_comm_poll_rx_queue (void)

  Polls the RX queue in the internal communication and calls the function defined in internal_-comm_init.

- unsigned char internal_comm_poll_tx_queue (void)

  Polls the TX queue in the internal communication and sends the data if there is a message in the queue.

- void internal_comm_add_tx_message (unsigned char command, unsigned char length, char ∗data)

  Add a message to the transmit queue.

- void internal_comm_send_ack (void)

  Send an ACK message to the internal communication uart.

- void internal_comm_send_nack (void)

  Send a NACK message to the internal communication uart.

- void internal_comm_send_message (UC_MESSAGE tx_message)

  Send a message to the internal communication uart.

- void internal_comm_reset_rx (void)

  Will reset the RX variables.

- void internal_comm_1ms_timer (void)

  Function which should be called each ms.

- void internal_comm_resend (void)

  Will trigger a resend of the last message.

6.83.1 Detailed Description

The internal communication routines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "internal_comm.h"
```

Definition in file internal_comm.h.

6.83.2 Define Documentation

6.83.2.1 #define UC_COMM_MSG_ACK 0xFB

The acknowledge command of the bus

Definition at line 73 of file internal_comm.h.

Referenced by internal_comm_send_ack(), and ISR().

6.83.2.2 #define UC_COMM_MSG_NACK 0xFA

The NOT acknowledge command of the bus

Definition at line 75 of file internal_comm.h.

Referenced by internal_comm_send_nack(), and ISR().

6.83.2.3 #define UC_COMM_MSG_POSTAMBLE 0xFD

The postamble of the BUS message

Definition at line 71 of file internal_comm.h.

Referenced by internal_comm_send_ack(), internal_comm_send_message(), internal_comm_-send_nack(), and ISR().

6.83.2.4 #define UC_COMM_MSG_PREAMBLE 0xFE

The preamble of the BUS message

Definition at line 69 of file internal_comm.h.

Referenced by internal_comm_send_ack(), internal_comm_send_message(), internal_comm_-send_nack(), and ISR().

6.83.2.5 #define UC_PREAMBLE_FOUND 0

if the device is a motherboard we need to set the proper USARTs used

if the device is a frontpanel we need to set the proper USARTs used Preamble found for the communication between the uCs

Definition at line 61 of file internal_comm.h.

Referenced by ISR().

6.83.3 Function Documentation

6.83.3.1 void internal_comm_add_tx_message (unsigned char *command*, unsigned char *length*, char ∗ *data*)

Add a message to the transmit queue.

Parameters:

command The command we wish to perform

length The length of the data field

data The data we wish to send

Definition at line 166 of file internal_comm.c.

References UC_MESSAGE::checksum, UC_MESSAGE::cmd, UC_MESSAGE::data, int_-comm_tx_queue_add(), and UC_MESSAGE::length.

Referenced by antenna_ctrl_deactivate_outputs(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_send_rx_ant_band_data_to_bus(), antenna_ctrl_send_rx_ant_data_-to_bus(), band_ctrl_send_band_data_to_bus(), computer_interface_parse_data(), main(), parse_internal_comm_message(), ps2_process_key(), radio_poll_status(), shutdown_device(), and sub_menu_send_data_to_bus().

6.83.3.2 void internal_comm_init (void(∗)(UC_MESSAGE) *func_ptr_rx*, void(∗)(char) *func_ptr_tx*)

Initialize the internal communication.

Parameters:

func_ptr_rx The function you wish to call when a new message has been recieved and should be parsed

func_ptr_tx The function used to send data to the hardware handeling the data transmission

Definition at line 68 of file internal_comm.c.

References struct_uc_com::char_count, struct_uc_com::checksum, f_ptr_rx, f_ptr_tx, struct_uc_com::flags, int_comm_rx_queue_dropall(), and int_comm_tx_queue_dropall().

Referenced by main().

6.83.3.3 unsigned char internal_comm_poll_rx_queue (void)

Polls the RX queue in the internal communication and calls the function defined in internal_-comm_init.

Returns:

1 if a message was in the buffer and got parsed, 0 if not

Definition at line 91 of file internal_comm.c.

References f_ptr_rx, int_comm_rx_queue_drop(), int_comm_rx_queue_get(), and int_-comm_rx_queue_is_empty().

Referenced by main().

6.83.3.4 unsigned char internal_comm_poll_tx_queue (void)

Polls the TX queue in the internal communication and sends the data if there is a message in the queue.

Returns:

1 if a message was in the buffer and got sent, 0 if not

Definition at line 105 of file internal_comm.c.

References int_comm_tx_queue_get(), int_comm_tx_queue_is_empty(), internal_comm_-send_message(), and msg_not_acked.

Referenced by main().

6.83.3.5 void internal_comm_send_message (UC_MESSAGE *tx_message*)

Send a message to the internal communication uart.

Parameters:

tx_message The message we wish to send

Definition at line 147 of file internal_comm.c.

References UC_MESSAGE::checksum, UC_MESSAGE::cmd, counter_tx_timeout, UC_-MESSAGE::data, f_ptr_tx, UC_MESSAGE::length, UC_COMM_MSG_POSTAMBLE, and UC_COMM_MSG_PREAMBLE.

Referenced by internal_comm_poll_tx_queue(), and internal_comm_resend().

6.84 internal_comm_commands.h File Reference

The internal communication commands.

Defines

- #define INT_COMM_TURN_DEVICE_OFF 0xC0

  Initialize the shut down sequence.

- #define INT_COMM_PULL_THE_PLUG 0xC1

  Shut down the device.

- #define INT_COMM_AUX_CHANGE_OUTPUT_PIN 0xC2

  Change the state of one of the AUX pins on the X11 connector.

- #define INT_COMM_AUX_READ_INPUT_PIN 0xC3

  Read the input status of an AUX pin on the X11 connector.

- #define INT_COMM_GET_BAND_BCD_STATUS 0xC4

  Read the BCD input on the top floor.

- #define INT_COMM_PS2_KEYPRESSED 0xC5

  A key was pressed on the external PS2 keypad.

- #define INT_COMM_PC_CTRL 0xC6

  Command used to transfer commands from the PC to the front panel and vice versa.

6.84.1 Detailed Description

The internal communication commands.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "internal_comm_commands.c"
```

Definition in file internal_comm_commands.h.

6.85 internal_comm_rx_queue.c File Reference

The internal communication RX QUEUE.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "internal_comm_rx_queue.h"
```

Functions

- void int_comm_rx_queue_init (void)

  Initialize the internal comm rx queue.
- void int_comm_rx_queue_add (UC_MESSAGE message)

  Insert a message into the TX queue (FIFO).
- UC_MESSAGE int_comm_rx_queue_get (void)

  Retrieve the first message from the FIFO TX queue.
- void int_comm_rx_queue_drop (void)
- void int_comm_rx_queue_dropall (void)

  Erase all content in the TX queue.
- unsigned char int_comm_rx_queue_is_empty (void)

  Check if the queue is empty.

Variables

- int_comm_rx_queue_struct int_comm_rx_queue

  The RX queue.

6.85.1 Detailed Description

The internal communication RX QUEUE.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "internal_comm_rx_queue.c"
```

Definition in file internal_comm_rx_queue.c.

6.85.2 Function Documentation

6.85.2.1 void int_comm_rx_queue_add (UC_MESSAGE *message*)

Insert a message into the TX queue (FIFO).

Parameters:

message - The message that should be inserted to the queue

Definition at line 43 of file internal_comm_rx_queue.c.

References rx_linked_list::first, INTERNAL_COMM_RX_QUEUE_SIZE, rx_linked_-list::last, and rx_linked_list::message.

Referenced by ISR().

6.85.2.2 void int_comm_rx_queue_drop (void)

Drops the first message in the queue Frees up the memory space aswell.

Definition at line 66 of file internal_comm_rx_queue.c.

References rx_linked_list::first, and INTERNAL_COMM_RX_QUEUE_SIZE.

Referenced by internal_comm_poll_rx_queue().

6.85.2.3 void int_comm_rx_queue_dropall (void)

Erase all content in the TX queue.

Returns:

The number of items that were cleared

Definition at line 76 of file internal_comm_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::last.

Referenced by internal_comm_init().

6.85.2.4 UC_MESSAGE int_comm_rx_queue_get (void)

Retrieve the first message from the FIFO TX queue.

Returns:

The first message in the queue

Definition at line 59 of file internal_comm_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::message.

Referenced by internal_comm_poll_rx_queue().

6.85.2.5 unsigned char int_comm_rx_queue_is_empty (void)

Check if the queue is empty.

Returns:

1 if the queue is empty and 0 otherwise

Definition at line 84 of file internal_comm_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::last.

Referenced by internal_comm_poll_rx_queue().

6.86 internal_comm_rx_queue.h File Reference

The internal communication RX QUEUE.

`#include "internal_comm.h"`

Classes

- struct rx_linked_list

  The structure of the RX circular buffer.

Defines

- #define INTERNAL_COMM_RX_QUEUE_SIZE 5

  The size of the RX QUEUE.

Typedefs

- typedef struct rx_linked_list int_comm_rx_queue_struct

  The structure of the RX circular buffer.

Functions

- void int_comm_rx_queue_add (UC_MESSAGE message)

  Insert a message into the TX queue (FIFO).

- UC_MESSAGE int_comm_rx_queue_get (void)

  Retrieve the first message from the FIFO TX queue.

- void int_comm_rx_queue_drop (void)
- void int_comm_rx_queue_dropall (void)

  Erase all content in the TX queue.

- void int_comm_rx_queue_init (void)

  Initialize the internal comm rx queue.

- unsigned char int_comm_rx_queue_is_empty (void)

  Check if the queue is empty.

6.86.1 Detailed Description

The internal communication RX QUEUE.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "internal_comm_rx_queue.h"
```

Definition in file internal_comm_rx_queue.h.

6.86.2 Function Documentation

6.86.2.1 void int_comm_rx_queue_add (UC_MESSAGE *message*)

Insert a message into the TX queue (FIFO).

Parameters:

message - The message that should be inserted to the queue

Definition at line 43 of file internal_comm_rx_queue.c.

References rx_linked_list::first, INTERNAL_COMM_RX_QUEUE_SIZE, rx_linked_-list::last, and rx_linked_list::message.

Referenced by ISR().

6.86.2.2 void int_comm_rx_queue_drop (void)

Drops the first message in the queue Frees up the memory space aswell.

Definition at line 66 of file internal_comm_rx_queue.c.

References rx_linked_list::first, and INTERNAL_COMM_RX_QUEUE_SIZE.

Referenced by internal_comm_poll_rx_queue().

6.86.2.3 void int_comm_rx_queue_dropall (void)

Erase all content in the TX queue.

Returns:

The number of items that were cleared

Definition at line 76 of file internal_comm_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::last.

Referenced by internal_comm_init().

6.86.2.4 UC_MESSAGE int_comm_rx_queue_get (void)

Retrieve the first message from the FIFO TX queue.

Returns:

The first message in the queue

Definition at line 59 of file internal_comm_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::message.

Referenced by internal_comm_poll_rx_queue().

6.86.2.5 unsigned char int_comm_rx_queue_is_empty (void)

Check if the queue is empty.

Returns:

1 if the queue is empty and 0 otherwise

Definition at line 84 of file internal_comm_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::last.

Referenced by internal_comm_poll_rx_queue().

6.87 internal_comm_tx_queue.c File Reference

The internal communication TX QUEUE.

`#include <stdio.h>`

`#include <stdlib.h>`

`#include <string.h>`

`#include "internal_comm_tx_queue.h"`

Functions

- void int_comm_int_comm_tx_queue_init (void)
- void int_comm_tx_queue_add (UC_MESSAGE message)

  Insert a message into the TX queue (FIFO).
- UC_MESSAGE int_comm_tx_queue_get (void)

  Retrieve the first message from the FIFO TX queue.
- void int_comm_tx_queue_drop (void)
- void int_comm_tx_queue_dropall (void)

  Erase all content in the TX queue.
- unsigned char int_comm_tx_queue_is_empty (void)

  Check if the queue is empty.

Variables

- int_comm_tx_queue_struct int_comm_tx_queue

  The TX queue.

6.87.1 Detailed Description

The internal communication TX QUEUE.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "internal_comm_tx_queue.c"
```

Definition in file internal_comm_tx_queue.c.

6.87.2 Function Documentation

6.87.2.1 void int_comm_int_comm_tx_queue_init (void)

Initialize the internal communication TX queue

Definition at line 35 of file internal_comm_tx_queue.c.

References tx_linked_list::first, and tx_linked_list::last.

6.87.2.2 void int_comm_tx_queue_add (UC_MESSAGE *message*)

Insert a message into the TX queue (FIFO).

Parameters:

message - The message that should be inserted to the queue

Definition at line 43 of file internal_comm_tx_queue.c.

References tx_linked_list::first, tx_linked_list::last, and tx_linked_list::message.

Referenced by internal_comm_add_tx_message().

6.87.2.3 void int_comm_tx_queue_drop (void)

Drops the first message in the queue Frees up the memory space aswell.

Definition at line 66 of file internal_comm_tx_queue.c.

References tx_linked_list::first.

Referenced by ISR().

6.87.2.4 void int_comm_tx_queue_dropall (void)

Erase all content in the TX queue.

Returns:

The number of items that were cleared

Definition at line 76 of file internal_comm_tx_queue.c.

References tx_linked_list::first, and tx_linked_list::last.

Referenced by internal_comm_init().

6.87.2.5 UC_MESSAGE int_comm_tx_queue_get (void)

Retrieve the first message from the FIFO TX queue.

Returns:

The first message in the queue

Definition at line 59 of file internal_comm_tx_queue.c.

References tx_linked_list::first, and tx_linked_list::message.

Referenced by internal_comm_poll_tx_queue(), and internal_comm_resend().

6.87.2.6 unsigned char int_comm_tx_queue_is_empty (void)

Check if the queue is empty.

Returns:

1 if the queue is empty and 0 otherwise

Definition at line 84 of file internal_comm_tx_queue.c.

References tx_linked_list::first, and tx_linked_list::last.

Referenced by internal_comm_poll_tx_queue().

6.88 powermeter/display_unit/lcd.c File Reference

Character LCD driver for HD44780/SED1278 displays.

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "global.h"
#include "delay.h"
#include "lcd.h"
```

Functions

- void **write_data** (unsigned char data)
- unsigned char **read_data** (void)
- unsigned char **__attribute__** ((progmem))
- void **lcdInitHW** (void)
- void **lcdBusyWait** (void)
- void **lcdControlWrite** (u08 data)
- u08 **lcdControlRead** (void)
- void **lcdDataWrite** (u08 data)
- u08 **lcdDataRead** (void)
- void **lcdInit** ()
- void **lcdHome** (void)
- void **lcdClear** (void)
- void **lcdGotoXY** (u08 x, u08 y)
- void **lcdLoadCustomChar** (u08 *lcdCustomCharArray, u08 romCharNum, u08 lcdCharNum)
- void **lcdPrintData** (char *data, u08 nBytes)
- void **lcdProgressBar** (u16 progress, u16 maxprogress, u08 length)

6.88.1 Detailed Description

Character LCD driver for HD44780/SED1278 displays.

Definition in file lcd.c.

6.89 powermeter/display_unit/lcd.h File Reference

Character LCD driver for HD44780/SED1278 displays.

`#include "global.h"`

`#include "lcdconf.h"`

Defines

- #define **LCD_DELAY** __asm__ __volatile__ ("nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n nop\n");
- #define **LCD_CLR** 0
- #define **LCD_HOME** 1
- #define **LCD_ENTRY_MODE** 2
- #define **LCD_ENTRY_INC** 1
- #define **LCD_ENTRY_SHIFT** 0
- #define **LCD_ON_CTRL** 3
- #define **LCD_ON_DISPLAY** 2
- #define **LCD_ON_CURSOR** 1
- #define **LCD_ON_BLINK** 0
- #define **LCD_MOVE** 4
- #define **LCD_MOVE_DISP** 3
- #define **LCD_MOVE_RIGHT** 2
- #define **LCD_FUNCTION** 5
- #define **LCD_FUNCTION_8BIT** 4
- #define **LCD_FUNCTION_2LINES** 3
- #define **LCD_FUNCTION_10DOTS** 2
- #define **LCD_CGRAM** 6
- #define **LCD_DDRAM** 7
- #define **LCD_BUSY** 7
- #define **LCD_FDEF_1** (1<<LCD_FUNCTION_8BIT)
- #define **LCD_FDEF_2** (1<<LCD_FUNCTION_2LINES)
- #define **LCD_FUNCTION_DEFAULT** ((1<<LCD_FUNCTION) | LCD_FDEF_1 | LCD_FDEF_2)
- #define **LCD_MODE_DEFAULT** ((1<<LCD_ENTRY_MODE) | (1<<LCD_-ENTRY_INC))
- #define **LCDCHAR_PROGRESS05** 0
- #define **LCDCHAR_PROGRESS15** 1
- #define **LCDCHAR_PROGRESS25** 2
- #define **LCDCHAR_PROGRESS35** 3
- #define **LCDCHAR_PROGRESS45** 4
- #define **LCDCHAR_PROGRESS55** 5
- #define **LCDCHAR_REWINDARROW** 6
- #define **LCDCHAR_STOPBLOCK** 7
- #define **LCDCHAR_PAUSEBARS** 8
- #define **LCDCHAR_FORWARDARROW** 9
- #define **LCDCHAR_SCROLLUPARROW** 10
- #define **LCDCHAR_SCROLLDNARROW** 11
- #define **LCDCHAR_BLANK** 12
- #define **LCDCHAR_ANIPLAYICON0** 13

- #define **LCDCHAR_ANIPLAYICON1** 14
- #define **LCDCHAR_ANIPLAYICON2** 15
- #define **LCDCHAR_ANIPLAYICON3** 16
- #define **PROGRESSPIXELS_PER_CHAR** 6

Functions

- unsigned char **__attribute__** ((progmem)) LcdCustomChar[ ]
- void **lcdInitHW** (void)
- void **lcdBusyWait** (void)
- void **lcdControlWrite** (u08 data)
- u08 **lcdControlRead** (void)
- void **lcdDataWrite** (u08 data)
- u08 **lcdDataRead** (void)
- void **lcdInit** (void)
- void **lcdHome** (void)
- void **lcdClear** (void)
- void **lcdGotoXY** (u08 row, u08 col)
- void **lcdLoadCustomChar** (u08 ∗lcdCustomCharArray, u08 romCharNum, u08 lcdCharNum)
- void **lcdPrintData** (char ∗data, u08 nBytes)
- void **lcdProgressBar** (u16 progress, u16 maxprogress, u08 length)

6.89.1 Detailed Description

Character LCD driver for HD44780/SED1278 displays.

Definition in file lcd.h.

6.90 powermeter/display_unit/lcdconf.h File Reference

Character LCD driver configuration.

Defines

- #define **LCD_PORT_INTERFACE**
- #define **LCD_CTRL_PORT** PORTC
- #define **LCD_CTRL_DDR** DDRC
- #define **LCD_CTRL_RS** 5
- #define **LCD_CTRL_RW** 6
- #define **LCD_CTRL_E** 7
- #define **LCD_DATA_POUT** PORTA
- #define **LCD_DATA_PIN** PINA
- #define **LCD_DATA_DDR** DDRA
- #define **LCD_LINES** 2
- #define **LCD_LINE_LENGTH** 20
- #define **LCD_LINE0_DDRAMADDR** 0x00
- #define **LCD_LINE1_DDRAMADDR** 0x40
- #define **LCD_LINE2_DDRAMADDR** 0x14
- #define **LCD_LINE3_DDRAMADDR** 0x54

6.90.1 Detailed Description

Character LCD driver configuration.

Definition in file lcdconf.h.

6.91 powermeter/display_unit/output.h File Reference

Output functions.

Functions

- void **output_show_display** (unsigned char index)
- void **output_update_leds** (void)

6.91.1 Detailed Description

Output functions.

Author:

Mikael Larsmark, SM2WMV

Date:

2009-06-23

Definition in file output.h.

6.92 wmv_bus/bus.c File Reference

The communication bus protocol used in the openASC project.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/wdt.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "bus.h"
#include "bus_tx_queue.h"
#include "bus_rx_queue.h"
#include "bus_commands.h"
#include "bus_ping.h"
#include "global.h"
#include "bus_usart.h"
```

Functions

- void bus_init (void)
  Init the communication bus.
- void bus_set_address (unsigned char addr)
  Set the address of this device on the bus.
- unsigned char bus_allowed_to_send (void)
  Returns if you are allowed to transmit data to the bus or not.
- unsigned char bus_get_address (void)
  Returns the address of this device.
- void bus_send_message (void)
  Sends the first message in the FIFO TX queue to the communication bus.
- void __inline__ bus_reset_tx_status (void)
  Function that resets the bus status variables.
- void __inline__ bus_reset_rx_status (void)
  Function that resets the bus status variables.
- unsigned char bus_is_master (void)
  Returns if the bus is set to be master.
- void bus_set_is_master (unsigned char state, unsigned char count)

Set the status if the device should be master or not.

- void bus_send_nack (unsigned char to_addr, unsigned char error_type)

  Send an NOT acknowledge.

- void bus_send_ack (unsigned char to_addr)

  Send an acknowledge.

- unsigned char bus_get_device_count (void)

  Receive the device count on the bus.

- void bus_set_device_count (unsigned char device_count)

  Set the number of devices that are on the bus.

- void bus_resend_message (void)

  Resend the last message.

- void bus_check_tx_status (void)

  Checks if there is anything that should be sent in the TX queue.

- void bus_add_tx_message (unsigned char from_addr, unsigned char to_addr, unsigned char flags, unsigned char cmd, unsigned char length, unsigned char data[ ])

  Adds a message to the TX queue which will be sent as soon as possible.

- void bus_add_rx_message (unsigned char from_addr, unsigned char to_addr, unsigned char flags, unsigned char cmd, unsigned char length, unsigned char data[ ])

  Adds a message to the RX queue which will be sent as soon as possible.

- void bus_add_new_message (void)

  Adds the message bus_new_message into the RX queue.

- void bus_message_nacked (unsigned char addr, unsigned char error_type)

  The message last sent was NACKED from the receiver.

- void bus_message_acked (unsigned char addr)

  The message last sent was acknowledged from the receiver.

- ISR (ISR_BUS_USART_DATA)
- ISR (ISR_BUS_USART_RECV)
- ISR (ISR_BUS_USART_TRANS)
- ISR (ISR_BUS_TIMER_COMPARE)

Variables

- bus_status_struct bus_status

  The bus status structure.

- unsigned char calc_checksum = 0

  Variable used to calculate the checksum when receiving a message.

- BUS_MESSAGE bus_new_message

  The new message.

- unsigned char timer_bus_timeout = 0

  Counter that keeps track of how long time ago it was when we received a new character and if it's over the limit we erase all the RX buffer.

- unsigned int counter_sync_timeout = 0

  Counter that keeps track of how long time ago it was when we received a new SYNC message on the BUS.

- unsigned int counter_130us = 0

  Counter which keeps track of each time the 130us timer counts up.

6.92.1 Detailed Description

The communication bus protocol used in the openASC project.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "wmv_bus/bus.c"
```

Definition in file bus.c.

6.92.2 Function Documentation

6.92.2.1 void bus_add_rx_message (unsigned char *from_addr*, unsigned char *to_addr*, unsigned char *flags*, unsigned char *cmd*, unsigned char *length*, unsigned char *data*[ ])

Adds a message to the RX queue which will be sent as soon as possible.

Parameters:

from_addr The address of the sender

to_addr The address to the receiever

flags Different flags, see defines

cmd The command performed

length The length of the data received

data The data receieved

Definition at line 324 of file bus.c.

References BUS_MESSAGE::cmd, BUS_MESSAGE::data, BUS_MESSAGE::flags, BUS_-MESSAGE::from_addr, BUS_MESSAGE::length, rx_queue_add(), and BUS_MESSAGE::to_-addr.

6.92.2.2 void bus_add_tx_message (unsigned char *from_addr*, unsigned char *to_addr*, unsigned char *flags*, unsigned char *cmd*, unsigned char *length*, unsigned char *data*[ ])

Adds a message to the TX queue which will be sent as soon as possible.

Parameters:

from_addr The address of the sender

to_addr The address to the receiever

flags Different flags, see defines

cmd The command wanted to be performed

length The length of the data wanting to be sent

data The data wanted to be transmitted to the receiever

Definition at line 291 of file bus.c.

References bus_allowed_to_send(), BUS_CMD_SYNC, BUS_MESSAGE::checksum, BUS_-MESSAGE::cmd, BUS_MESSAGE::data, BUS_MESSAGE::flags, BUS_MESSAGE::from_-addr, BUS_MESSAGE::length, BUS_MESSAGE::to_addr, and tx_queue_add().

Referenced by antenna_ctrl_deactivate_outputs(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_send_rx_ant_band_data_to_bus(), antenna_ctrl_send_rx_ant_data_-to_bus(), band_ctrl_send_band_data_to_bus(), bus_parse_message(), bus_send_ack(), bus_send_nack(), ISR(), main(), send_ping(), and sub_menu_send_data_to_bus().

6.92.2.3 unsigned char bus_allowed_to_send (void)

Returns if you are allowed to transmit data to the bus or not.

Returns:

1 if it's allowed to transmit and 0 if not

Definition at line 114 of file bus.c.

References BUS_STATUS_MASTER_SENT_SYNC_BIT, and bus_status_struct::flags.

Referenced by bus_add_tx_message(), ISR(), and main().

6.92.2.4 unsigned char bus_get_address (void)

Returns the address of this device.

Returns:

The address of this device

Definition at line 123 of file bus.c.

References bus_status_struct::ext_addr.

Referenced by antenna_ctrl_deactivate_outputs(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_send_rx_ant_band_data_to_bus(), antenna_ctrl_send_rx_ant_data_-to_bus(), band_ctrl_send_band_data_to_bus(), bus_parse_message(), ISR(), main(), send_ping(), and sub_menu_send_data_to_bus().

6.92.2.5 unsigned char bus_get_device_count (void)

Receive the device count on the bus.

Returns:

The number of devices on the bus

Definition at line 235 of file bus.c.

References bus_status_struct::device_count.

Referenced by main().

6.92.2.6 unsigned char bus_is_master (void)

Returns if the bus is set to be master.

Returns:

1 if it is configured to be master, 0 otherwise

Definition at line 196 of file bus.c.

References BUS_STATUS_DEVICE_IS_MASTER_BIT, and bus_status_struct::flags.

Referenced by ISR(), and main().

6.92.2.7 void bus_message_nacked (unsigned char *addr*, unsigned char *error_type*)

The message last sent was NACKED from the receiver.

Parameters:

addr The address of the device that sent the NACK
error_type Contains information why the message was NACKED

Definition at line 348 of file bus.c.

References bus_resend_message(), BUS_STATUS_MESSAGE_ACK_TIMEOUT, bus_-status_struct::flags, BUS_MESSAGE::to_addr, and tx_queue_get().

Referenced by bus_parse_message(), and event_bus_parse_message().

6.92.2.8 void bus_send_nack (unsigned char *to_addr*, unsigned char *error_type*)

Send an NOT acknowledge.

Parameters:

to_addr Which address we wish to send the ping to
error_type Why was the message nacked, see bus.h for more information about BUS errors

Definition at line 223 of file bus.c.

References bus_add_tx_message(), BUS_BROADCAST_ADDR, BUS_CMD_NACK, and bus_status_struct::ext_addr.

Referenced by ISR().

6.92.2.9 void bus_set_address (unsigned char *addr*)

Set the address of this device on the bus.

Parameters:

addr The address of this device

Definition at line 108 of file bus.c.

References bus_status_struct::ext_addr.

Referenced by main().

6.92.2.10 void bus_set_device_count (unsigned char *device_count*)

Set the number of devices that are on the bus.

Parameters:

device_count The number of devices on the bus, ie the number of time slots

Definition at line 241 of file bus.c.

References BUS_TIME_MULTIPLIER, bus_status_struct::device_count, and bus_status_-struct::device_count_mult.

6.92.2.11 void bus_set_is_master (unsigned char *state*, unsigned char *count*)

Set the status if the device should be master or not.

Parameters:

state 1 if you wish the device to be master, 0 if you wish that it should be slave
count The nr of devices

Definition at line 206 of file bus.c.

References BUS_STATUS_ALLOWED_TO_SEND_BIT, BUS_STATUS_DEVICE_IS_-MASTER_BIT, BUS_STATUS_FORCE_SYNC, BUS_TIME_MULTIPLIER, bus_status_-struct::device_count, bus_status_struct::device_count_mult, and bus_status_struct::flags.

Referenced by main().

6.92.2.12 ISR (ISR_BUS_TIMER_COMPARE)

Timer interrupt with ~130us intervals

Definition at line 512 of file bus.c.

References BUS_ACK_WRAPAROUND_LIMIT, bus_is_master(), bus_resend_message(), bus_reset_rx_status(), BUS_STATUS_MASTER_SENT_SYNC_BIT, BUS_STATUS_-MESSAGE_ACK_TIMEOUT, BUS_STATUS_TIME_SLOT_ACTIVE, BUS_SYNC_-TIMEOUT_LIMIT, BUS_TIMEOUT_LIMIT, counter_130us, counter_sync_timeout, bus_status_struct::device_count, bus_status_struct::device_count_mult, ERROR_TYPE_-BUS_SYNC, event_set_error(), bus_status_struct::flags, bus_status_struct::frame_counter, led_set_error(), LED_STATE_ON, bus_status_struct::lower_limit, timer_bus_timeout, tx_queue_dropall(), bus_status_struct::upper_limit, and bus_status_struct::wraparounds.

6.92.2.13 ISR (ISR_BUS_USART_TRANS)

USART data transmit interrupt

Definition at line 506 of file bus.c.

References BUS_STATUS_RECEIVE_ON, BUS_STATUS_SEND_ACTIVE, and bus_-status_struct::flags.

6.92.2.14 ISR (ISR_BUS_USART_RECV)

USART data receive interrupt

Definition at line 386 of file bus.c.

References bus_add_new_message(), BUS_BROADCAST_ADDR, BUS_CHECKSUM_-ERROR, BUS_CMD_PING, BUS_CMD_SYNC, BUS_MESSAGE_FLAGS_NEED_ACK, bus_ping_new_stamp(), bus_reset_rx_status(), bus_send_ack(), bus_send_nack(), BUS_-STATUS_ALLOWED_TO_SEND_BIT, BUS_STATUS_MASTER_SENT_SYNC_BIT, BUS_STATUS_PREAMBLE_FOUND_BIT, BUS_STATUS_RECEIVE_ON, BUS_TIME_-MULTIPLIER, calc_checksum, bus_status_struct::char_count, BUS_MESSAGE::checksum, BUS_MESSAGE::cmd, counter_sync_timeout, BUS_MESSAGE::data, bus_status_-struct::device_count, bus_status_struct::device_count_mult, bus_status_struct::ext_addr, BUS_MESSAGE::flags, bus_status_struct::flags, bus_status_struct::frame_counter, BUS_-MESSAGE::from_addr, BUS_MESSAGE::length, bus_status_struct::prev_char, timer_bus_-timeout, and BUS_MESSAGE::to_addr.

6.92.2.15 ISR (ISR_BUS_USART_DATA)

USART data interrupt

Definition at line 381 of file bus.c.

6.93 wmv_bus/bus.h File Reference

The communication bus protocol used in the openASC project.

Classes

- struct BUS_MESSAGE
- struct rx_linked_list

  The structure of the RX circular buffer.
- struct tx_linked_list

  The structure of the TX circular buffer.
- struct bus_status_struct

Defines

- #define DEF_NR_DEVICES 25

  Define the proper interrupt routines depending on hardware.
- #define DEFAULT_STARTUP_DELAY 90

  The startup time for the device. This is so that all units dont send ping at the same time.
- #define BUS_MSG_PREAMBLE 0xFE
- #define BUS_MSG_POSTAMBLE 0xFD
- #define BUS_MSG_ACK 0xFA
- #define BUS_MSG_NACK 0xFB
- #define BUS_BROADCAST_ADDR 0x00
- #define DEVICE_ID_MAINBOX 1

  Device ID for the mainbox.
- #define DEVICE_ID_DRIVER_POS 2

  Device ID for the positive driver module.
- #define DEVICE_ID_DRIVER_NEG 3

  Device ID for the negative driver module.
- #define DEVICE_ID_ROTATOR_UNIT 4

  Device ID for the rotator unit.
- #define DEVICE_ID_COMPUTER 5

  Device ID for a computer.
- #define DEVICE_ID_POWERMETER_PICKUP 6

  Device ID for a power meter pickup.
- #define DEVICE_ID_GENERAL_IO 7

  Device ID for the General I/O card.

- #define BUS_MAX_RESENDS 10
- #define BUS_DEVICE_STATUS_MESSAGE_INTERVAL 1500
- #define BUS_MASTER_SYNC_INTERVAL 1000
- #define BUS_SYNC_TIMEOUT_LIMIT 3200
- #define BUS_ACK_WRAPAROUND_LIMIT 10
- #define BUS_TIMEOUT_LIMIT 5
- #define BUS_STATUS_DEVICE_IS_MASTER_BIT 0
- #define BUS_STATUS_ALLOWED_TO_SEND_BIT 1
- #define BUS_STATUS_PREAMBLE_FOUND_BIT 2
- #define BUS_STATUS_MASTER_SENT_SYNC_BIT 3
- #define BUS_STATUS_TIME_SLOT_ACTIVE 4
- #define BUS_STATUS_SEND_MESSAGE 5
- #define BUS_STATUS_SEND_ACTIVE 6
- #define BUS_STATUS_RECEIVE_ON 7
- #define BUS_STATUS_FORCE_SYNC 8
- #define BUS_STATUS_MESSAGE_ACK_TIMEOUT 9
- #define BUS_TIME_MULTIPLIER 4
- #define BUS_TIME_INTERRUPT_INTERVAL 130
- #define BUS_TIME_FRAME_LIMIT 520
- #define BUS_SLOT_DEAD_TIME 130
- #define SERIAL_RX_BUF_LENGTH 20

  Length of the RX buffer.

- #define BUS_MESSAGE_DATA_SIZE 15

  Define the length of the bus message data field length.

- #define BUS_MESSAGE_FLAGS_NEED_ACK 0
- #define BUS_CHECKSUM_ERROR 0

  BUS ERRORs.

Typedefs

- typedef struct rx_linked_list rx_queue_struct

  The structure of the RX circular buffer.

- typedef struct tx_linked_list tx_queue_struct

  The structure of the TX circular buffer.

Functions

- void bus_add_tx_message (unsigned char from_addr, unsigned char to_addr, unsigned char flags, unsigned char cmd, unsigned char length, unsigned char data[ ])

  Adds a message to the TX queue which will be sent as soon as possible.

- void bus_add_rx_message (unsigned char from_addr, unsigned char to_addr, unsigned char flags, unsigned char cmd, unsigned char length, unsigned char data[ ])

  Adds a message to the RX queue which will be sent as soon as possible.

- void bus_set_address (unsigned char addr)

  Set the address of this device on the bus.

- unsigned char bus_get_address (void)

  Returns the address of this device.

- void bus_init (void)

  Init the communication bus.

- void bus_resend_message (void)

  Resend the last message.

- void bus_send_ack (unsigned char to_addr)

  Send an acknowledge.

- void bus_send_nack (unsigned char to_addr, unsigned char error_type)

  Send an NOT acknowledge.

- void bus_message_acked (unsigned char addr)

  The message last sent was acknowledged from the receiver.

- void bus_message_nacked (unsigned char addr, unsigned char error_type)

  The message last sent was NACKED from the receiver.

- void __inline__ bus_reset_tx_status (void)

  Function that resets the bus status variables.

- void __inline__ bus_reset_rx_status (void)

  Function that resets the bus status variables.

- unsigned char bus_is_master (void)

  Returns if the bus is set to be master.

- void bus_set_is_master (unsigned char state, unsigned char count)

  Set the status if the device should be master or not.

- unsigned char bus_get_device_count (void)

  Receive the device count on the bus.

- void bus_set_device_count (unsigned char device_count)

  Set the number of devices that are on the bus.

- unsigned char bus_allowed_to_send (void)

  Returns if you are allowed to transmit data to the bus or not.

- void bus_check_tx_status (void)

  Checks if there is anything that should be sent in the TX queue.

6.93.1 Detailed Description

The communication bus protocol used in the openASC project.

Definition in file bus.h.

6.93.2 Define Documentation

6.93.2.1 #define BUS_ACK_WRAPAROUND_LIMIT 10

The timeout limit between a message that was sent to when it will be a resend, this is counted as number of wraparounds on the bus, ie 5 would mean 5 wraparounds

Definition at line 171 of file bus.h.

Referenced by ISR().

6.93.2.2 #define BUS_BROADCAST_ADDR 0x00

Bus broadcast address - All broadcast messages should contain an ID explaining which kind of device that is sending the message

Definition at line 137 of file bus.h.

Referenced by bus_send_message(), bus_send_nack(), ISR(), main(), and send_ping().

6.93.2.3 #define BUS_DEVICE_STATUS_MESSAGE_INTERVAL 1500

The interval between each status message (time is in ms)

Definition at line 158 of file bus.h.

Referenced by ISR(), and main().

6.93.2.4 #define BUS_MASTER_SYNC_INTERVAL 1000

The interval which the SYNC command is sent out from the master (time in ms) This value is not allowed to be over 5000 ms since that will make it too big for the timer controlling the SYNC timeout on all the devices.

Definition at line 163 of file bus.h.

Referenced by main().

6.93.2.5 #define BUS_MAX_RESENDS 10

The number of times a message is resent before it's dropped and an error flag is set

Definition at line 155 of file bus.h.

Referenced by bus_resend_message().

6.93.2.6 #define BUS_MESSAGE_FLAGS_NEED_ACK 0

BUS MESSAGE flags The message should be ACKED

Definition at line 218 of file bus.h.

Referenced by antenna_ctrl_deactivate_outputs(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_send_rx_ant_band_data_to_bus(), antenna_ctrl_send_rx_ant_data_to_-bus(), band_ctrl_send_band_data_to_bus(), ISR(), and sub_menu_send_data_to_bus().

6.93.2.7 #define BUS_MSG_ACK 0xFA

The acknowledge command of the bus

Definition at line 131 of file bus.h.

6.93.2.8 #define BUS_MSG_NACK 0xFB

The NOT acknowledge command of the bus

Definition at line 133 of file bus.h.

6.93.2.9 #define BUS_MSG_POSTAMBLE 0xFD

The postamble of the BUS message

Definition at line 129 of file bus.h.

Referenced by bus_send_message().

6.93.2.10 #define BUS_MSG_PREAMBLE 0xFE

The preamble of the BUS message

Definition at line 127 of file bus.h.

Referenced by bus_send_message().

6.93.2.11 #define BUS_SLOT_DEAD_TIME 130

The dead time of the time slot, in us. This should be set pretty high to accept rather high clock drift There is a dead period both before and after our slot, so total dead time is BUS_SLOT_-DEAD_TIME $*$ 2

Definition at line 208 of file bus.h.

Referenced by bus_init().

6.93.2.12 #define BUS_STATUS_ALLOWED_TO_SEND_BIT 1

This bit shows if it's allowed to send messages on the bus, ie first sync has been received

Definition at line 181 of file bus.h.

6.93.2.13 #define BUS_STATUS_DEVICE_IS_MASTER_BIT 0

This bit is to set if the device is MASTER in the bus_status.flags variable

Definition at line 179 of file bus.h.

Referenced by bus_is_master(), and bus_set_is_master().

6.93.2.14 #define BUS_STATUS_FORCE_SYNC 8

This bit is to force a SYNC message to be sent

Definition at line 195 of file bus.h.

Referenced by bus_check_tx_status(), and bus_set_is_master().

6.93.2.15 #define BUS_STATUS_MASTER_SENT_SYNC_BIT 3

This bit is to set if the MASTER has sent a SYNC message so we are allowed to start sending

Definition at line 185 of file bus.h.

Referenced by bus_allowed_to_send(), and ISR().

6.93.2.16 #define BUS_STATUS_MESSAGE_ACK_TIMEOUT 9

This bit is to see if a message should be acked or not, used for the timeout of an acknowledge

Definition at line 197 of file bus.h.

Referenced by bus_message_acked(), bus_message_nacked(), bus_send_message(), and ISR().

6.93.2.17 #define BUS_STATUS_PREAMBLE_FOUND_BIT 2

This bit is set if the preamble is found in the bus_status.flags variable

Definition at line 183 of file bus.h.

6.93.2.18 #define BUS_STATUS_RECEIVE_ON 7

This bit is to indicate that we are CURRENTLY receiving a message

Definition at line 193 of file bus.h.

Referenced by bus_init(), bus_send_message(), and ISR().

6.93.2.19 #define BUS_STATUS_SEND_ACTIVE 6

This bit is to indicate that we are CURRENTLY sending a message

Definition at line 191 of file bus.h.

Referenced by bus_send_message(), and ISR().

6.93.2.20 #define BUS_STATUS_SEND_MESSAGE 5

This bit is to indicate that we should try to send the message currently in the TX queue

Definition at line 189 of file bus.h.

Referenced by bus_check_tx_status(), bus_resend_message(), and bus_reset_tx_status().

6.93.2.21 #define BUS_STATUS_TIME_SLOT_ACTIVE 4

This bit is to set if the device own time slot is currently active, ie it is possibly allowed to send

Definition at line 187 of file bus.h.

Referenced by bus_check_tx_status(), and ISR().

6.93.2.22 #define BUS_SYNC_TIMEOUT_LIMIT 3200

This limit is used to detect if it was too long ago since we receieved a SYNC message from the master. If so it will stop with all outgoing communication.

Definition at line 167 of file bus.h.

Referenced by ISR().

6.93.2.23 #define BUS_TIME_FRAME_LIMIT 520

The time frame size of the bus time slots, in us

Definition at line 205 of file bus.h.

Referenced by bus_init().

6.93.2.24 #define BUS_TIME_INTERRUPT_INTERVAL 130

The interval of the timer interrupts, in us

Definition at line 203 of file bus.h.

Referenced by bus_init().

6.93.2.25 #define BUS_TIME_MULTIPLIER 4

This is the multiplier for the send window BUS_TIME_INTERRUPT_INTERVAL ∗ BUS_-TIME_MULTIPLIER = BUS_TIME_FRAME_LIMIT

Definition at line 201 of file bus.h.

Referenced by bus_set_device_count(), bus_set_is_master(), and ISR().

6.93.2.26 #define BUS_TIMEOUT_LIMIT 5

Timeout limit for how long it can take without receiving a message before the buffer is cleared, this is counted as time, 5 would mean 5 ∗ 130 us

Definition at line 175 of file bus.h.

Referenced by ISR().

6.93.2.27 #define DEF_NR_DEVICES 25

Define the proper interrupt routines depending on hardware.

The default number of devices

Definition at line 121 of file bus.h.

Referenced by bus_ping_get_failed_count(), bus_ping_get_failed_ping(), bus_ping_init(), bus_ping_tick(), and main().

6.93.3 Function Documentation

6.93.3.1 void bus_add_rx_message (unsigned char *from_addr*, unsigned char *to_addr*, unsigned char *flags*, unsigned char *cmd*, unsigned char *length*, unsigned char *data*[ ])

Adds a message to the RX queue which will be sent as soon as possible.

Parameters:

from_addr The address of the sender

to_addr The address to the receiever

flags Different flags, see defines

cmd The command performed

length The length of the data received

data The data receieved

Definition at line 324 of file bus.c.

References BUS_MESSAGE::cmd, BUS_MESSAGE::data, BUS_MESSAGE::flags, BUS_-MESSAGE::from_addr, BUS_MESSAGE::length, rx_queue_add(), and BUS_MESSAGE::to_-addr.

6.93.3.2 void bus_add_tx_message (unsigned char *from_addr*, unsigned char *to_addr*, unsigned char *flags*, unsigned char *cmd*, unsigned char *length*, unsigned char *data*[ ])

Adds a message to the TX queue which will be sent as soon as possible.

Parameters:

from_addr The address of the sender

to_addr The address to the receiever

flags Different flags, see defines

cmd The command wanted to be performed

length The length of the data wanting to be sent

data The data wanted to be transmitted to the receiever

Definition at line 291 of file bus.c.

References bus_allowed_to_send(), BUS_CMD_SYNC, BUS_MESSAGE::checksum, BUS_-MESSAGE::cmd, BUS_MESSAGE::data, BUS_MESSAGE::flags, BUS_MESSAGE::from_-addr, BUS_MESSAGE::length, BUS_MESSAGE::to_addr, and tx_queue_add().

Referenced by antenna_ctrl_deactivate_outputs(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_send_rx_ant_band_data_to_bus(), antenna_ctrl_send_rx_ant_data_-to_bus(), band_ctrl_send_band_data_to_bus(), bus_parse_message(), bus_send_ack(), bus_send_nack(), ISR(), main(), send_ping(), and sub_menu_send_data_to_bus().

6.93.3.3 unsigned char bus_allowed_to_send (void)

Returns if you are allowed to transmit data to the bus or not.

Returns:

1 if it's allowed to transmit and 0 if not

Definition at line 114 of file bus.c.

References BUS_STATUS_MASTER_SENT_SYNC_BIT, and bus_status_struct::flags.

Referenced by bus_add_tx_message(), ISR(), and main().

6.93.3.4 unsigned char bus_get_address (void)

Returns the address of this device.

Returns:

The address of this device

Definition at line 123 of file bus.c.

References bus_status_struct::ext_addr.

Referenced by antenna_ctrl_deactivate_outputs(), antenna_ctrl_send_ant_data_to_bus(), antenna_ctrl_send_rx_ant_band_data_to_bus(), antenna_ctrl_send_rx_ant_data_-to_bus(), band_ctrl_send_band_data_to_bus(), bus_parse_message(), ISR(), main(), send_ping(), and sub_menu_send_data_to_bus().

6.93.3.5 unsigned char bus_get_device_count (void)

Receive the device count on the bus.

Returns:

The number of devices on the bus

Definition at line 235 of file bus.c.

References bus_status_struct::device_count.

Referenced by main().

6.93.3.6 unsigned char bus_is_master (void)

Returns if the bus is set to be master.

Returns:

1 if it is configured to be master, 0 otherwise

Definition at line 196 of file bus.c.

References BUS_STATUS_DEVICE_IS_MASTER_BIT, and bus_status_struct::flags.

Referenced by ISR(), and main().

6.93.3.7 void bus_message_nacked (unsigned char *addr*, unsigned char *error_type*)

The message last sent was NACKED from the receiver.

Parameters:

addr The address of the device that sent the NACK

error_type Contains information why the message was NACKED

Definition at line 348 of file bus.c.

References bus_resend_message(), BUS_STATUS_MESSAGE_ACK_TIMEOUT, bus_-status_struct::flags, BUS_MESSAGE::to_addr, and tx_queue_get().

Referenced by bus_parse_message(), and event_bus_parse_message().

6.93.3.8 void bus_send_nack (unsigned char *to_addr*, unsigned char *error_type*)

Send an NOT acknowledge.

Parameters:

to_addr Which address we wish to send the ping to

error_type Why was the message nacked, see bus.h for more information about BUS errors

Definition at line 223 of file bus.c.

References bus_add_tx_message(), BUS_BROADCAST_ADDR, BUS_CMD_NACK, and bus_status_struct::ext_addr.

Referenced by ISR().

6.93.3.9 void bus_set_address (unsigned char *addr*)

Set the address of this device on the bus.

Parameters:

addr The address of this device

Definition at line 108 of file bus.c.

References bus_status_struct::ext_addr.

Referenced by main().

6.93.3.10 void bus_set_device_count (unsigned char *device_count*)

Set the number of devices that are on the bus.

Parameters:

device_count The number of devices on the bus, ie the number of time slots

Definition at line 241 of file bus.c.

References BUS_TIME_MULTIPLIER, bus_status_struct::device_count, and bus_status_-struct::device_count_mult.

6.93.3.11 void bus_set_is_master (unsigned char *state*, unsigned char *count*)

Set the status if the device should be master or not.

Parameters:

state 1 if you wish the device to be master, 0 if you wish that it should be slave

count The nr of devices

Definition at line 206 of file bus.c.

References BUS_STATUS_ALLOWED_TO_SEND_BIT, BUS_STATUS_DEVICE_IS_-MASTER_BIT, BUS_STATUS_FORCE_SYNC, BUS_TIME_MULTIPLIER, bus_status_-struct::device_count, bus_status_struct::device_count_mult, and bus_status_struct::flags.

Referenced by main().

6.94 wmv_bus/bus_commands.h File Reference

Global commands for the WMV communication bus.

Defines

- #define BUS_CMD_ACK 0xFA
- #define BUS_CMD_NACK 0xFB
- #define BUS_CMD_SYNC 0x01
- #define BUS_CMD_PING 0x02
- #define BUS_CMD_DRIVER_ACTIVATE_TXRX_MODE 0x10
- #define BUS_CMD_DRIVER_DEACTIVATE_TXRX_MODE 0x11
- #define BUS_CMD_DRIVER_ACTIVATE_TX_ANT_COMBO 0x12
- #define BUS_CMD_DRIVER_DEACTIVATE_TX_ANT_COMBO 0x13
- #define BUS_CMD_DRIVER_ACTIVATE_RX_ANT_COMBO 0x14
- #define BUS_CMD_DRIVER_DEACTIVATE_RX_ANT_COMBO 0x15
- #define BUS_CMD_DRIVER_ACTIVATE_ANT_OUTPUT 0x16
- #define BUS_CMD_DRIVER_DEACTIVATE_ANT_OUTPUT 0x17
- #define BUS_CMD_DRIVER_ACTIVATE_BAND_OUTPUT 0x18
- #define BUS_CMD_DRIVER_DEACTIVATE_BAND_OUTPUT 0x19
- #define BUS_CMD_DRIVER_ACTIVATE_RX_ANT_OUTPUT 0x1A
- #define BUS_CMD_DRIVER_DEACTIVATE_RX_ANT_OUTPUT 0x1B
- #define BUS_CMD_DRIVER_DEACTIVATE_RX_BAND_OUTPUT 0x1C
- #define BUS_CMD_DRIVER_DEACTIVATE_ALL_RX_BAND_OUTPUTS 0x1D
- #define BUS_CMD_DRIVER_ACTIVATE_RX_BAND_OUTPUT 0x1E
- #define BUS_CMD_DRIVER_GET_STATUS 0x1F
- #define BUS_CMD_GET_TEMPERATURE 0x20
- #define BUS_CMD_DRIVER_DEACTIVATE_ALL_OUTPUTS 0x21
- #define BUS_CMD_DRIVER_DEACTIVATE_ALL_ANT_OUTPUTS 0x22
- #define BUS_CMD_DRIVER_DEACTIVATE_ALL_BAND_OUTPUTS 0x23
- #define BUS_CMD_DRIVER_DEACTIVATE_ALL_RX_ANTENNA_-OUTPUTS 0x24
- #define BUS_CMD_SET_PTT_SETTINGS 0x25
- #define BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT1_OUTPUT 0x26
- #define BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT1_OUTPUT 0x27
- #define BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT1_-OUTPUTS 0x28
- #define BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT2_OUTPUT 0x29
- #define BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT2_OUTPUT 0x3A
- #define BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT2_-OUTPUTS 0x3B
- #define BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT3_OUTPUT 0x3C
- #define BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT3_OUTPUT 0x3D
- #define BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT3_-OUTPUTS 0x3E
- #define BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT4_OUTPUT 0x3F
- #define BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT4_OUTPUT 0x40
- #define BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_ANT4_-OUTPUTS 0x41
- #define BUS_CMD_ROTATOR_SET_ANGLE 0x60

- #define BUS_CMD_ROTATOR_GET_ANGLE 0x61
- #define BUS_CMD_ROTATOR_GET_STATUS 0x62
- #define BUS_CMD_ROTATOR_ROTATE_CW 0x63
- #define BUS_CMD_ROTATOR_ROTATE_CCW 0x64
- #define BUS_CMD_ROTATOR_STOP 0x65
- #define BUS_CMD_TRANSPARENT 0x66
- #define BUS_CMD_POWERMETER_STATUS 0x70
- #define BUS_CMD_POWERMETER_CALIBRATE 0x71

6.94.1 Detailed Description

Global commands for the WMV communication bus.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "wmv_bus/bus_commands.h"
```

Definition in file bus_commands.h.

6.94.2 Define Documentation

6.94.2.1 #define BUS_CMD_ACK 0xFA

Send an acknowledge

Definition at line 29 of file bus_commands.h.

Referenced by bus_parse_message(), bus_send_ack(), bus_send_message(), and event_bus_-parse_message().

6.94.2.2 #define BUS_CMD_DRIVER_ACTIVATE_ANT_OUTPUT 0x16

Activate a driver output, type = ANT

Definition at line 52 of file bus_commands.h.

Referenced by antenna_ctrl_send_ant_data_to_bus(), bus_parse_message(), and parse_-internal_comm_message().

6.94.2.3 #define BUS_CMD_DRIVER_ACTIVATE_BAND_OUTPUT 0x18

Activate a driver output, type = BAND

Definition at line 56 of file bus_commands.h.

Referenced by band_ctrl_send_band_data_to_bus(), bus_parse_message(), and parse_-internal_comm_message().

6.94.2.4 #define BUS_CMD_DRIVER_ACTIVATE_RX_ANT_COMBO 0x14

Activate a driver combo, type = RX ANT combo

Definition at line 48 of file bus_commands.h.

6.94.2.5 #define BUS_CMD_DRIVER_ACTIVATE_RX_ANT_-OUTPUT 0x1A

Activate a driver output, type = RX Antenna

Definition at line 60 of file bus_commands.h.

Referenced by antenna_ctrl_send_rx_ant_data_to_bus(), bus_parse_message(), and parse_-internal_comm_message().

6.94.2.6 #define BUS_CMD_DRIVER_ACTIVATE_RX_BAND_-OUTPUT 0x1E

Activate RX BAND output

Definition at line 68 of file bus_commands.h.

Referenced by antenna_ctrl_send_rx_ant_band_data_to_bus(), and bus_parse_message().

6.94.2.7 #define BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT1_-OUTPUT 0x26

Activate sub menu output

Definition at line 85 of file bus_commands.h.

Referenced by bus_parse_message(), and sub_menu_send_data_to_bus().

6.94.2.8 #define BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT2_-OUTPUT 0x29

Activate sub menu output

Definition at line 91 of file bus_commands.h.

Referenced by bus_parse_message(), and sub_menu_send_data_to_bus().

6.94.2.9 #define BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT3_-OUTPUT 0x3C

Activate sub menu output

Definition at line 97 of file bus_commands.h.

Referenced by bus_parse_message(), and sub_menu_send_data_to_bus().

6.94.2.10 #define BUS_CMD_DRIVER_ACTIVATE_SUBMENU_ANT4_-OUTPUT 0x3F

Activate sub menu output

Definition at line 103 of file bus_commands.h.

Referenced by bus_parse_message(), and sub_menu_send_data_to_bus().

6.94.2.11 #define BUS_CMD_DRIVER_ACTIVATE_TX_ANT_COMBO 0x12

Activate a driver combo, type = TX ANT

Definition at line 44 of file bus_commands.h.

6.94.2.12 #define BUS_CMD_DRIVER_ACTIVATE_TXRX_MODE 0x10

Activate TX/RX mode

Definition at line 40 of file bus_commands.h.

6.94.2.13 #define BUS_CMD_DRIVER_DEACTIVATE_ALL_ANT_-OUTPUTS 0x22

Deactivate all the ant outputs enabled by this device

Definition at line 77 of file bus_commands.h.

Referenced by antenna_ctrl_deactivate_all(), antenna_ctrl_send_ant_data_to_bus(), bus_-parse_message(), and parse_internal_comm_message().

6.94.2.14 #define BUS_CMD_DRIVER_DEACTIVATE_ALL_BAND_-OUTPUTS 0x23

Deactivate all the band outputs enabled by this device

Definition at line 79 of file bus_commands.h.

Referenced by band_ctrl_deactivate_all(), bus_parse_message(), and parse_internal_comm_-message().

6.94.2.15 #define BUS_CMD_DRIVER_DEACTIVATE_ALL_OUTPUTS 0x21

Deactivate all the outputs enabled by this device

Definition at line 75 of file bus_commands.h.

Referenced by bus_parse_message(), and parse_internal_comm_message().

6.94.2.16 #define BUS_CMD_DRIVER_DEACTIVATE_ALL_RX_-ANTENNA_OUTPUTS 0x24

Deactivate all the RX ANTENNA outputs

Definition at line 81 of file bus_commands.h.

Referenced by antenna_ctrl_change_rx_ant(), antenna_ctrl_send_rx_ant_data_to_bus(), and bus_parse_message().

6.94.2.17 #define BUS_CMD_DRIVER_DEACTIVATE_ALL_RX_BAND_-OUTPUTS 0x1D

Deactivate ALL RX BAND outputs

Definition at line 66 of file bus_commands.h.

Referenced by antenna_ctrl_deactivate_all_rx_band(), antenna_ctrl_send_rx_ant_band_-data_to_bus(), and bus_parse_message().

6.94.2.18 #define BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_-ANT1_OUTPUTS 0x28

Deactivate all sub menu outputs

Definition at line 89 of file bus_commands.h.

Referenced by bus_parse_message(), sub_menu_deactivate_all(), and sub_menu_send_data_-to_bus().

6.94.2.19 #define BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_-ANT2_OUTPUTS 0x3B

Deactivate all sub menu outputs

Definition at line 95 of file bus_commands.h.

Referenced by bus_parse_message(), sub_menu_deactivate_all(), and sub_menu_send_data_-to_bus().

6.94.2.20 #define BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_-ANT3_OUTPUTS 0x3E

Deactivate all sub menu outputs

Definition at line 101 of file bus_commands.h.

Referenced by bus_parse_message(), sub_menu_deactivate_all(), and sub_menu_send_data_-to_bus().

6.94.2.21 #define BUS_CMD_DRIVER_DEACTIVATE_ALL_SUBMENU_-ANT4_OUTPUTS 0x41

Deactivate all sub menu outputs

Definition at line 107 of file bus_commands.h.

Referenced by bus_parse_message(), sub_menu_deactivate_all(), and sub_menu_send_data_-to_bus().

6.94.2.22 #define BUS_CMD_DRIVER_DEACTIVATE_ANT_OUTPUT 0x17

Deactivate a driver output, type = ANT

Definition at line 54 of file bus_commands.h.

Referenced by bus_parse_message(), and parse_internal_comm_message().

6.94.2.23 #define BUS_CMD_DRIVER_DEACTIVATE_BAND_-OUTPUT 0x19

Deactivate a driver output, type = BAND

Definition at line 58 of file bus_commands.h.

6.94.2.24 #define BUS_CMD_DRIVER_DEACTIVATE_RX_ANT_-COMBO 0x15

Deactivate a driver combo, type = RX ANT combo

Definition at line 50 of file bus_commands.h.

6.94.2.25 #define BUS_CMD_DRIVER_DEACTIVATE_RX_ANT_-OUTPUT 0x1B

Deactivate a driver output, type = RX Antenna

Definition at line 62 of file bus_commands.h.

Referenced by bus_parse_message(), and parse_internal_comm_message().

6.94.2.26 #define BUS_CMD_DRIVER_DEACTIVATE_RX_BAND_-OUTPUT 0x1C

Deactivate RX BAND output

Definition at line 64 of file bus_commands.h.

Referenced by bus_parse_message().

6.94.2.27 #define BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT1_-OUTPUT 0x27

Deactivate sub menu output

Definition at line 87 of file bus_commands.h.

Referenced by bus_parse_message(), and sub_menu_send_data_to_bus().

6.94.2.28 #define BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT2_-OUTPUT 0x3A

Deactivate sub menu output

Definition at line 93 of file bus_commands.h.

Referenced by bus_parse_message(), and sub_menu_send_data_to_bus().

6.94.2.29 #define BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT3_-OUTPUT 0x3D

Deactivate sub menu output

Definition at line 99 of file bus_commands.h.

Referenced by bus_parse_message(), and sub_menu_send_data_to_bus().

6.94.2.30 #define BUS_CMD_DRIVER_DEACTIVATE_SUBMENU_ANT4_-OUTPUT 0x40

Deactivate sub menu output

Definition at line 105 of file bus_commands.h.

Referenced by bus_parse_message(), and sub_menu_send_data_to_bus().

6.94.2.31 #define BUS_CMD_DRIVER_DEACTIVATE_TX_ANT_-COMBO 0x13

Deactivate a driver combo, type = TX ANT

Definition at line 46 of file bus_commands.h.

6.94.2.32 #define BUS_CMD_DRIVER_DEACTIVATE_TXRX_MODE 0x11

Deactivate TX/RX mode

Definition at line 42 of file bus_commands.h.

6.94.2.33 #define BUS_CMD_DRIVER_GET_STATUS 0x1F

Get the driver status

Definition at line 71 of file bus_commands.h.

Referenced by bus_parse_message().

6.94.2.34 #define BUS_CMD_GET_TEMPERATURE 0x20

Retrieve the temperature

Definition at line 73 of file bus_commands.h.

Referenced by bus_parse_message().

6.94.2.35 #define BUS_CMD_NACK 0xFB

Send an NOT acknowledge

Definition at line 31 of file bus_commands.h.

Referenced by bus_parse_message(), bus_send_message(), bus_send_nack(), and event_bus_-parse_message().

6.94.2.36 #define BUS_CMD_PING 0x02

Sends a ping which all devices can use to see what's connected to the bus

Definition at line 37 of file bus_commands.h.

Referenced by bus_parse_message(), ISR(), main(), and send_ping().

6.94.2.37 #define BUS_CMD_POWERMETER_CALIBRATE 0x71

PowerMeter calibration command

Definition at line 127 of file bus_commands.h.

6.94.2.38 #define BUS_CMD_POWERMETER_STATUS 0x70

PowerMeter information

Definition at line 125 of file bus_commands.h.

6.94.2.39 #define BUS_CMD_ROTATOR_GET_ANGLE 0x61

Get the current direction

Definition at line 112 of file bus_commands.h.

6.94.2.40 #define BUS_CMD_ROTATOR_GET_STATUS 0x62

Get the current direction

Definition at line 114 of file bus_commands.h.

6.94.2.41 #define BUS_CMD_ROTATOR_ROTATE_CCW 0x64

Rotate CounterClockWise

Definition at line 118 of file bus_commands.h.

6.94.2.42 #define BUS_CMD_ROTATOR_ROTATE_CW 0x63

Rotate ClockWise

Definition at line 116 of file bus_commands.h.

6.94.2.43 #define BUS_CMD_ROTATOR_SET_ANGLE 0x60

Set the target rotation direction and start rotation

Definition at line 110 of file bus_commands.h.

6.94.2.44 #define BUS_CMD_ROTATOR_STOP 0x65

Stop the rotation of the rotator

Definition at line 120 of file bus_commands.h.

6.94.2.45 #define BUS_CMD_SET_PTT_SETTINGS 0x25

Set the PTT settings, which PTT input that corresponds to which device

Definition at line 83 of file bus_commands.h.

Referenced by bus_parse_message().

6.94.2.46 #define BUS_CMD_SYNC 0x01

Transmit the SYNC signal. The SYNC signal contains one variable which describes the number of devices connected to the bus.

Definition at line 35 of file bus_commands.h.

Referenced by bus_add_tx_message(), bus_parse_message(), ISR(), and main().

6.94.2.47 #define BUS_CMD_TRANSPARENT 0x66

Transparent command which just redirects the data to the serial port

Definition at line 122 of file bus_commands.h.

6.95 wmv_bus/bus_ping.c File Reference

The communication bus ping control.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "bus.h"
#include "bus_ping.h"
```

Functions

- void bus_ping_init (void)

  Initialize the ping functions of the bus communication interface.

- void bus_ping_new_stamp (unsigned char from_addr, unsigned char device_type, unsigned char data_len, unsigned char ∗data)

  This function will update the ping list with the sent in arguments and reset the counter to 0.

- void bus_ping_tick (void)

  This function will update the time counter which keeps track of the time stamps of the ping message. Should be called every ms.

- bus_struct_ping_status ∗ bus_ping_get_failed_ping (void)

  This function will return a ping which has failed and will mark it that it has been reported.

- unsigned char bus_ping_get_failed_count (void)

  Goes through the ping list and checks how many has timed out.

- bus_struct_ping_status ∗ bus_ping_get_ping_data (unsigned char index)

  Returns a ping data structure.

Variables

- bus_struct_ping_status ping_list [DEF_NR_DEVICES]

  The ping list.

6.95.1 Detailed Description

The communication bus ping control.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-04-22

```
#include "wmv_bus/bus_ping.c"
```

Definition in file bus_ping.c.

6.95.2 Function Documentation

6.95.2.1 unsigned char bus_ping_get_failed_count (void)

Goes through the ping list and checks how many has timed out.

Returns:

The number of failed pings

Definition at line 84 of file bus_ping.c.

References BUS_PING_TIMEOUT_LIMIT, and DEF_NR_DEVICES.

6.95.2.2 bus_struct_ping_status∗ bus_ping_get_failed_ping (void)

This function will return a ping which has failed and will mark it that it has been reported.

Returns:

A pointer to a structure of type bus_struct_ping_status which contains information of the failed ping

Definition at line 66 of file bus_ping.c.

References BUS_PING_TIMEOUT_LIMIT, DEF_NR_DEVICES, bus_struct_ping_-status::flags, and PING_FLAG_PROCESSED.

6.95.2.3 bus_struct_ping_status∗ bus_ping_get_ping_data (unsigned char *index*)

Returns a ping data structure.

Parameters:

index The index of the ping structure we wish to retrieve from the list

Returns:

The ping data structure

Definition at line 99 of file bus_ping.c.

6.95.2.4 void bus_ping_new_stamp (unsigned char *from_addr*, unsigned char *device_type*, unsigned char *data_len*, unsigned char * *data*)

This function will update the ping list with the sent in arguments and reset the counter to 0.

Parameters:

from_addr The address which the PING message was sent from

device_type Which type of device this is

data_len The number of bytes the data is

data Additional data which might be used for status, for example current band information

Definition at line 41 of file bus_ping.c.

References bus_struct_ping_status::addr, bus_struct_ping_status::device_type, bus_struct_-ping_status::flags, PING_FLAG_PROCESSED, and bus_struct_ping_status::time_last_ping.

Referenced by ISR().

6.96 wmv_bus/bus_ping.h File Reference

The communication bus ping control.

Classes

- struct bus_struct_ping_status

  Struct which contains information of the bus ping information.

Defines

- #define BUS_PING_TIMEOUT_LIMIT 6000

  The timeout for the bus ping. After this time has passed a device is considered "dead".

- #define PING_FLAG_PROCESSED 0

  Bit is set if the ping timeout has been processed.

Functions

- void bus_ping_init (void)

  Initialize the ping functions of the bus communication interface.

- void bus_ping_tick (void)

  This function will update the time counter which keeps track of the time stamps of the ping message. Should be called every ms.

- void bus_ping_new_stamp (unsigned char from_addr, unsigned char device_type, unsigned char data_len, unsigned char ∗data)

  This function will update the ping list with the sent in arguments and reset the counter to 0.

- bus_struct_ping_status ∗ bus_ping_get_failed_ping (void)

  This function will return a ping which has failed and will mark it that it has been reported.

- unsigned char bus_ping_get_failed_count (void)

  Goes through the ping list and checks how many has timed out.

- bus_struct_ping_status ∗ bus_ping_get_ping_data (unsigned char index)

  Returns a ping data structure.

6.96.1 Detailed Description

The communication bus ping control.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-04-22

```
#include "wmv_bus/bus_ping.h"
```

Definition in file bus_ping.h.

6.96.2 Function Documentation

6.96.2.1 unsigned char bus_ping_get_failed_count (void)

Goes through the ping list and checks how many has timed out.

Returns:

The number of failed pings

Definition at line 84 of file bus_ping.c.

References BUS_PING_TIMEOUT_LIMIT, and DEF_NR_DEVICES.

6.96.2.2 bus_struct_ping_status∗ bus_ping_get_failed_ping (void)

This function will return a ping which has failed and will mark it that it has been reported.

Returns:

A pointer to a structure of type bus_struct_ping_status which contains information of the failed ping

Definition at line 66 of file bus_ping.c.

References BUS_PING_TIMEOUT_LIMIT, DEF_NR_DEVICES, bus_struct_ping_status::flags, and PING_FLAG_PROCESSED.

6.96.2.3 bus_struct_ping_status∗ bus_ping_get_ping_data (unsigned char *index*)

Returns a ping data structure.

Parameters:

index The index of the ping structure we wish to retrieve from the list

Returns:

The ping data structure

Definition at line 99 of file bus_ping.c.

6.96.2.4 void bus_ping_new_stamp (unsigned char *from_ addr*, unsigned char *device_ type*, unsigned char *data_ len*, unsigned char ∗ *data*)

This function will update the ping list with the sent in arguments and reset the counter to 0.

Parameters:

from_ addr The address which the PING message was sent from

device_ type Which type of device this is

data_ len The number of bytes the data is

data Additional data which might be used for status, for example current band information

Definition at line 41 of file bus_ping.c.

References bus_struct_ping_status::addr, bus_struct_ping_status::device_type, bus_struct_-ping_status::flags, PING_FLAG_PROCESSED, and bus_struct_ping_status::time_last_ping.

Referenced by ISR().

6.97 wmv_bus/bus_rx_queue.c File Reference

FIFO queue for the RXed messages.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bus_rx_queue.h"
#include "bus.h"
```

Functions

- void rx_queue_init (void)

  Initialize the RX queue.

- void rx_queue_add (BUS_MESSAGE message)

  Insert a message into the RX queue (FIFO).

- BUS_MESSAGE rx_queue_get ()

  Retrieve the first message from the FIFO RX queue.

- void rx_queue_drop (void)
- void rx_queue_dropall (void)

  Erase all content in the RX queue.

- unsigned char rx_queue_is_empty (void)

  Check if the queue is empty.

- unsigned char rx_queue_size (void)

  Get how much size of the RX queue is used at the moment.

Variables

- unsigned char bus_rx_queue_size

  Variable keeps track of how much of the queue that is currently used.

- rx_queue_struct rx_queue

  The rx queue.

6.97.1 Detailed Description

FIFO queue for the RXed messages.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "bus_rx_queue.c"
```

Definition in file bus_rx_queue.c.

6.97.2 Function Documentation

6.97.2.1 void rx_queue_add (BUS_MESSAGE *message*)

Insert a message into the RX queue (FIFO).

Parameters:

message - The message that should be inserted to the queue

Definition at line 55 of file bus_rx_queue.c.

References BUS_RX_QUEUE_SIZE, bus_rx_queue_size, ERROR_TYPE_BUS_RX_-QUEUE_FULL, event_set_error(), rx_linked_list::first, rx_linked_list::last, led_set_error(), LED_STATE_ON, and rx_linked_list::message.

Referenced by bus_add_new_message(), and bus_add_rx_message().

6.97.2.2 void rx_queue_drop (void)

Drops the first message in the queue Frees up the memory space aswell.

Definition at line 91 of file bus_rx_queue.c.

References BUS_RX_QUEUE_SIZE, bus_rx_queue_size, and rx_linked_list::first.

Referenced by bus_parse_message(), and event_bus_parse_message().

6.97.2.3 void rx_queue_dropall (void)

Erase all content in the RX queue.

Returns:

The number of items that were cleared

Definition at line 103 of file bus_rx_queue.c.

References bus_rx_queue_size, rx_linked_list::first, and rx_linked_list::last.

6.97.2.4 BUS_MESSAGE rx_queue_get (void)

Retrieve the first message from the FIFO RX queue.

Returns:

The first message in the queue

Definition at line 84 of file bus_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::message.

Referenced by bus_parse_message(), and event_bus_parse_message().

6.97.2.5 unsigned char rx_queue_is_empty (void)

Check if the queue is empty.

Returns:

1 if the queue is empty and 0 otherwise

Definition at line 112 of file bus_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::last.

Referenced by main().

6.97.2.6 unsigned char rx_queue_size (void)

Get how much size of the RX queue is used at the moment.

Returns:

The size of the queue that is used

Definition at line 121 of file bus_rx_queue.c.

References bus_rx_queue_size.

6.98 wmv_bus/bus_rx_queue.h File Reference

FIFO queue for the RXed messages.

```
#include "bus.h"
```

Functions

- void rx_queue_add (BUS_MESSAGE message)
  Insert a message into the RX queue (FIFO).
- BUS_MESSAGE rx_queue_get (void)
  Retrieve the first message from the FIFO RX queue.
- void rx_queue_drop (void)
- void rx_queue_dropall (void)
  Erase all content in the RX queue.
- void rx_queue_init (void)
  Initialize the RX queue.
- unsigned char rx_queue_is_empty (void)
  Check if the queue is empty.
- unsigned char rx_queue_size (void)
  Get how much size of the RX queue is used at the moment.

6.98.1 Detailed Description

FIFO queue for the RXed messages.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "wmv_bus/bus_rx_queue.h"
```

Definition in file bus_rx_queue.h.

6.98.2 Function Documentation

6.98.2.1 void rx_queue_add (BUS_MESSAGE *message*)

Insert a message into the RX queue (FIFO).

Parameters:

message - The message that should be inserted to the queue

Definition at line 55 of file bus_rx_queue.c.

References BUS_RX_QUEUE_SIZE, bus_rx_queue_size, ERROR_TYPE_BUS_RX_-QUEUE_FULL, event_set_error(), rx_linked_list::first, rx_linked_list::last, led_set_error(), LED_STATE_ON, and rx_linked_list::message.

Referenced by bus_add_new_message(), and bus_add_rx_message().

6.98.2.2 void rx_queue_drop (void)

Drops the first message in the queue Frees up the memory space aswell.

Definition at line 91 of file bus_rx_queue.c.

References BUS_RX_QUEUE_SIZE, bus_rx_queue_size, and rx_linked_list::first.

Referenced by bus_parse_message(), and event_bus_parse_message().

6.98.2.3 void rx_queue_dropall (void)

Erase all content in the RX queue.

Returns:

The number of items that were cleared

Definition at line 103 of file bus_rx_queue.c.

References bus_rx_queue_size, rx_linked_list::first, and rx_linked_list::last.

6.98.2.4 BUS_MESSAGE rx_queue_get (void)

Retrieve the first message from the FIFO RX queue.

Returns:

The first message in the queue

Definition at line 84 of file bus_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::message.

Referenced by bus_parse_message(), and event_bus_parse_message().

6.98.2.5 unsigned char rx_queue_is_empty (void)

Check if the queue is empty.

Returns:

1 if the queue is empty and 0 otherwise

Definition at line 112 of file bus_rx_queue.c.

References rx_linked_list::first, and rx_linked_list::last.

Referenced by main().

6.98.2.6 unsigned char rx_queue_size (void)

Get how much size of the RX queue is used at the moment.

Returns:

The size of the queue that is used

Definition at line 121 of file bus_rx_queue.c.

References bus_rx_queue_size.

6.99 wmv_bus/bus_tx_queue.c File Reference

FIFO queue for the TXed messages.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bus_tx_queue.h"
#include "bus.h"
```

Functions

- void tx_queue_init (void)

  Initialize the TX queue.

- void tx_queue_add (BUS_MESSAGE message)

  Insert a message into the TX queue (FIFO).

- BUS_MESSAGE tx_queue_get (void)

  Retrieve the first message from the FIFO TX queue.

- void tx_queue_drop (void)
- void tx_queue_dropall (void)

  Erase all content in the TX queue.

- unsigned char tx_queue_is_empty (void)

  Check if the queue is empty.

- unsigned char tx_queue_size (void)

  Get how much of the TX queue that is currently being used.

Variables

- unsigned char bus_tx_queue_size

  Variable that keeps track of how much of the TX queue that is being used.

- tx_queue_struct tx_queue

  The tx queue.

6.99.1 Detailed Description

FIFO queue for the TXed messages.

FIFO queue for the TXed messages.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "wmv_bus/bus_tx_queue.c"
```

Definition in file bus_tx_queue.c.

6.99.2 Function Documentation

6.99.2.1 void tx_queue_add (BUS_MESSAGE *message*)

Insert a message into the TX queue (FIFO).

Parameters:

message - The message that should be inserted to the queue

Definition at line 56 of file bus_tx_queue.c.

References BUS_TX_QUEUE_SIZE, bus_tx_queue_size, ERROR_TYPE_BUS_TX_-QUEUE_FULL, event_set_error(), tx_linked_list::first, tx_linked_list::last, led_set_error(), LED_STATE_ON, and tx_linked_list::message.

Referenced by bus_add_tx_message().

6.99.2.2 void tx_queue_drop (void)

Drops the first message in the queue Frees up the memory space aswell.

Definition at line 93 of file bus_tx_queue.c.

References BUS_TX_QUEUE_SIZE, bus_tx_queue_size, and tx_linked_list::first.

Referenced by bus_message_acked(), bus_resend_message(), and bus_send_message().

6.99.2.3 void tx_queue_dropall (void)

Erase all content in the TX queue.

Returns:

The number of items that were cleared

Definition at line 104 of file bus_tx_queue.c.

References bus_tx_queue_size, tx_linked_list::first, and tx_linked_list::last.

Referenced by ISR(), and main().

6.99.2.4 BUS_MESSAGE tx_queue_get (void)

Retrieve the first message from the FIFO TX queue.

Returns:

The first message in the queue

Definition at line 86 of file bus_tx_queue.c.

References tx_linked_list::first, and tx_linked_list::message.

Referenced by bus_message_acked(), bus_message_nacked(), and bus_send_message().

6.99.2.5 unsigned char tx_queue_is_empty (void)

Check if the queue is empty.

Returns:

1 if the queue is empty and 0 otherwise

Definition at line 112 of file bus_tx_queue.c.

References tx_linked_list::first, and tx_linked_list::last.

Referenced by main().

6.99.2.6 unsigned char tx_queue_size (void)

Get how much of the TX queue that is currently being used.

Returns:

How much of the queue is being used

Definition at line 121 of file bus_tx_queue.c.

References bus_tx_queue_size.

6.100 wmv_bus/bus_tx_queue.h File Reference

FIFO queue for the TXed messages.

`#include "bus.h"`

Functions

- void tx_queue_add (BUS_MESSAGE message)
  Insert a message into the TX queue (FIFO).
- BUS_MESSAGE tx_queue_get (void)
  Retrieve the first message from the FIFO TX queue.
- void tx_queue_drop (void)
- void tx_queue_dropall (void)
  Erase all content in the TX queue.
- void tx_queue_init (void)
  Initialize the TX queue.
- unsigned char tx_queue_is_empty (void)
  Check if the queue is empty.
- unsigned char tx_queue_size (void)
  Get how much of the TX queue that is currently being used.

6.100.1 Detailed Description

FIFO queue for the TXed messages.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "wmv_bus/bus_tx_queue.h"
```

Definition in file bus_tx_queue.h.

6.100.2 Function Documentation

6.100.2.1 void tx_queue_add (BUS_MESSAGE *message*)

Insert a message into the TX queue (FIFO).

Parameters:

message - The message that should be inserted to the queue

Definition at line 56 of file bus_tx_queue.c.

References BUS_TX_QUEUE_SIZE, bus_tx_queue_size, ERROR_TYPE_BUS_TX_-QUEUE_FULL, event_set_error(), tx_linked_list::first, tx_linked_list::last, led_set_error(), LED_STATE_ON, and tx_linked_list::message.

Referenced by bus_add_tx_message().

6.100.2.2 void tx_queue_drop (void)

Drops the first message in the queue Frees up the memory space aswell.

Definition at line 93 of file bus_tx_queue.c.

References BUS_TX_QUEUE_SIZE, bus_tx_queue_size, and tx_linked_list::first.

Referenced by bus_message_acked(), bus_resend_message(), and bus_send_message().

6.100.2.3 void tx_queue_dropall (void)

Erase all content in the TX queue.

Returns:

The number of items that were cleared

Definition at line 104 of file bus_tx_queue.c.

References bus_tx_queue_size, tx_linked_list::first, and tx_linked_list::last.

Referenced by ISR(), and main().

6.100.2.4 BUS_MESSAGE tx_queue_get (void)

Retrieve the first message from the FIFO TX queue.

Returns:

The first message in the queue

Definition at line 86 of file bus_tx_queue.c.

References tx_linked_list::first, and tx_linked_list::message.

Referenced by bus_message_acked(), bus_message_nacked(), and bus_send_message().

6.100.2.5 unsigned char tx_queue_is_empty (void)

Check if the queue is empty.

Returns:

1 if the queue is empty and 0 otherwise

Definition at line 112 of file bus_tx_queue.c.

References tx_linked_list::first, and tx_linked_list::last.

Referenced by main().

6.100.2.6 unsigned char tx_queue_size (void)

Get how much of the TX queue that is currently being used.

Returns:

How much of the queue is being used

Definition at line 121 of file bus_tx_queue.c.

References bus_tx_queue_size.

6.101 wmv_bus/bus_usart.c File Reference

Driver unit USART routines.

`#include <avr/io.h>`

`#include <avr/interrupt.h>`

`#include <stdio.h>`

`#include <string.h>`

Functions

- void bus_usart_init (unsigned int baudrate)

  Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the uCs datasheet.

- unsigned char bus_usart_transmit (unsigned char data)

  Send a character to the USART Send a single character to the USART used for the communication bus.

- unsigned char bus_usart_sendstring (char *data, unsigned char length)

  Send a string of characters to the USART Send a string of characters to the USART used for the communication bus.

- unsigned char bus_usart_receive (void)

  Retrieve one character from the USART Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char bus_usart_receive_loopback (void)

  The USART recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char bus_poll_usart_receive (void)

  Retrieve one character from the USART Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

6.101.1 Detailed Description

Driver unit USART routines.

These routines are used to communicate over the WMV bus.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "wmv_bus/bus_usart.c"
```

Definition in file bus_usart.c.

6.101.2 Function Documentation

6.101.2.1 unsigned char bus_poll_usart_receive (void)

Retrieve one character from the USART Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 301 of file bus_usart.c.

6.101.2.2 void bus_usart_init (unsigned int *baudrate*)

Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the uCs datasheet.

Parameters:

baudrate The baudrate param from the ATMEGA32 datasheet.

Definition at line 34 of file bus_usart.c.

Referenced by bus_init().

6.101.2.3 unsigned char bus_usart_receive (void)

Retrieve one character from the USART Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 181 of file bus_usart.c.

6.101.2.4 unsigned char bus_usart_receive_loopback (void)

The USART recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 232 of file bus_usart.c.

References bus_usart_transmit().

6.101.2.5 unsigned char bus_usart_sendstring (char * *data*, unsigned char *length*)

Send a string of characters to the USART Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Definition at line 168 of file bus_usart.c.

References bus_usart_transmit().

6.101.2.6 unsigned char bus_usart_transmit (unsigned char *data*)

Send a character to the USART Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Definition at line 116 of file bus_usart.c.

Referenced by bus_send_message(), bus_usart_receive_loopback(), and bus_usart_-sendstring().

6.102 wmv_bus/bus_usart.h File Reference

BUS usart routines.

Functions

- unsigned char bus_poll_usart_receive (void)

  Retrieve one character from the USART Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

- void bus_usart_init (unsigned int baudrate)

  Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the uCs datasheet.

- unsigned char bus_usart_transmit (unsigned char data)

  Send a character to the USART Send a single character to the USART used for the communication bus.

- unsigned char bus_usart_receive (void)

  Retrieve one character from the USART Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

- unsigned char bus_usart_receive_loopback (void)

  The USART recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

- unsigned char bus_usart_sendstring (char ∗data, unsigned char length)

  Send a string of characters to the USART Send a string of characters to the USART used for the communication bus.

6.102.1 Detailed Description

BUS usart routines.

Author:

Mikael Larsmark, SM2WMV

Date:

2010-01-25

```
#include "wmv_bus/bus_usart.h"
```

Definition in file bus_usart.h.

6.102.2 Function Documentation

6.102.2.1 unsigned char bus_poll_usart_receive (void)

Retrieve one character from the USART Retrieve one character from the USART. With this function you will need to poll the USART, it does NOT wait until a character is in the buffer.

Returns:

The character from the RX USART buffer

Definition at line 301 of file bus_usart.c.

6.102.2.2 void bus_usart_init (unsigned int *baudrate*)

Initiliaze the USART for the communication bus This function is used to initialize the USART which a baudrate that needs to be sent in as a parameter Use the baudrate settings specified in the uCs datasheet.

Parameters:

baudrate The baudrate param from the ATMEGA32 datasheet.

Definition at line 34 of file bus_usart.c.

Referenced by bus_init().

6.102.2.3 unsigned char bus_usart_receive (void)

Retrieve one character from the USART Retrieve one character from the USART. This function will wait until there is a character in the USART RX buffer.

Returns:

The character from the RX USART buffer

Definition at line 181 of file bus_usart.c.

6.102.2.4 unsigned char bus_usart_receive_loopback (void)

The USART recieve loopback This function does wait for a character in the RX buffer and returns it to the transmit buffer.

Returns:

The character from the RX USART buffer

Definition at line 232 of file bus_usart.c.

References bus_usart_transmit().

6.102.2.5 unsigned char bus_usart_sendstring (char ∗ *data*, unsigned char *length*)

Send a string of characters to the USART Send a string of characters to the USART used for the communication bus.

Parameters:

data The string of characters you wish to send

length The length of the string you wish to send

Definition at line 168 of file bus_usart.c.

References bus_usart_transmit().

6.102.2.6 unsigned char bus_usart_transmit (unsigned char *data*)

Send a character to the USART Send a single character to the USART used for the communication bus.

Parameters:

data The character you want to send

Definition at line 116 of file bus_usart.c.

Referenced by bus_send_message(), bus_usart_receive_loopback(), and bus_usart_-sendstring().