My Project

Generated by Doxygen 1.8.15

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

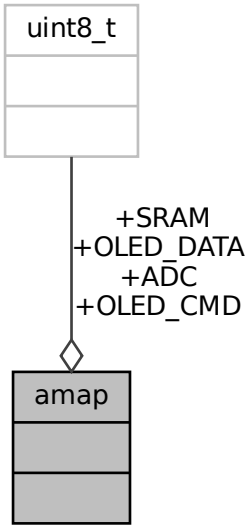
Chapter 3

Class Documentation

3.1 amap Struct Reference

`#include <DEFINITIONS.h>`

Collaboration diagram for amap:



Public Attributes

- uint8_t OLED_CMD [512]
- uint8_t OLED_DATA [512]
- uint8_t ADC [1024]
- uint8_t SRAM [2048]

3.1.1 Member Data Documentation

3.1.1.1 ADC

```
uint8_t amap::ADC[1024]
```

3.1.1.2 OLED_CMD

```
uint8_t amap::OLED_CMD[512]
```

3.1.1.3 OLED_DATA

```
uint8_t amap::OLED_DATA[512]
```

3.1.1.4 SRAM

```
uint8_t amap::SRAM[2048]
```

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

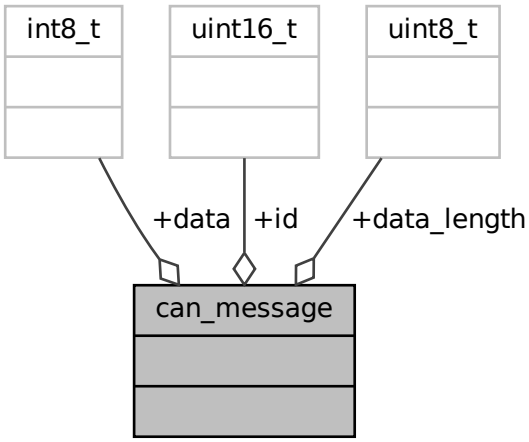
- DEFINITIONS.h

3.2 can_message Struct Reference

Struct for the contents of a CAN message.

`#include <mcp2515_driver.h>`

Collaboration diagram for can_message:



Public Attributes

- uint16_t id
- uint8_t data_length
- int8_t data [8]

3.2.1 Detailed Description

Struct for the contents of a CAN message.

3.2.2 Member Data Documentation

3.2.2.1 data

```
int8_t can_message::data[8]
```

3.2.2.2 data_length

```
uint8_t can_message::data_length
```

3.2.2.3 id

```
uint16_t can_message::id
```

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

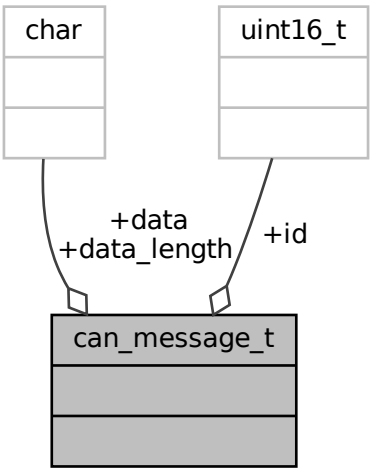
- mcp2515_driver.h

3.3 can_message_t Struct Reference

Contains the data sent with CAN messages.

`#include <can_controller.h>`

Collaboration diagram for can_message_t:



Public Attributes

- uint16_t id
- char data_length
- char data [8]

3.3.1 Detailed Description

Contains the data sent with CAN messages.

3.3.2 Member Data Documentation

3.3.2.1 data

`char can_message_t::data[8]`

3.3.2.2 data_length

```
char can_message_t::data_length
```

3.3.2.3 id

```
uint16_t can_message_t::id
```

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

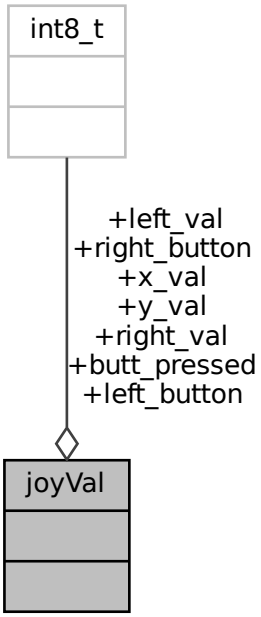
- GccApplication1/can_controller.h

3.4 joyVal Struct Reference

Send amount of time node 2 has been active.

```
#include <motor_controller.h>
```

Collaboration diagram for joyVal:



Public Attributes

- int8_t x_val
- int8_t y_val
- int8_t butt_pressed
- int8_t left_val
- int8_t right_val
- int8_t left_button
- int8_t right_button

3.4.1 Detailed Description

Send amount of time node 2 has been active.

Struct to store joystick values in both directions.

Parameters

msgToSend	Pointer to a CAN_MESSAGE allocated in memory

3.4.2 Member Data Documentation

3.4.2.1 butt_pressed

```
int8_t joyVal::butt_pressed
```

3.4.2.2 left_button

```
int8_t joyVal::left_button
```

3.4.2.3 left_val

```
int8_t joyVal::left_val
```

3.4.2.4 right_button

```
int8_t joyVal::right_button
```

3.4.2.5 right_val

```
int8_t joyVal::right_val
```

3.4.2.6 x_val

```
int8_t joyVal::x_val
```

3.4.2.7 y_val

```
int8_t joyVal::y_val
```

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

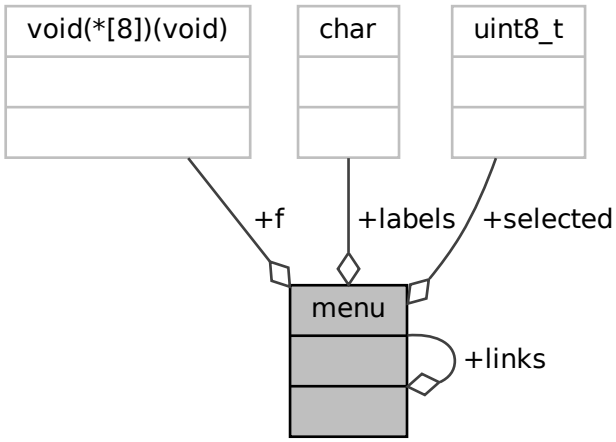
- GccApplication1/motor_controller.h
- joystick_driver.h

3.5 menu Struct Reference

Struct for content in a menu page labels are names of options, one for each line links are references to submenues (linked list) f are function pointers as an option to submenues selected keeps track of the selected option for each menu.

```
#include <menu.h>
```

Collaboration diagram for menu:



Public Attributes

- char * labels [8]
- struct menu * links [8]
- void(* f [8])(void)
- uint8_t selected

3.5.1 Detailed Description

Struct for content in a menu page labels are names of options, one for each line links are references to submenues (linked list) f are function pointers as an option to submenues selected keeps track of the selected option for each menu.

3.5.2 Member Data Documentation

3.5.2.1 f

```
void(* menu::f[8])(void)
```

3.5.2.2 labels

```
char* menu::labels[8]
```

3.5.2.3 links

```
struct menu* menu::links[8]
```

3.5.2.4 selected

```
uint8_t menu::selected
```

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

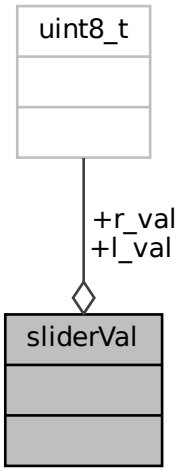
- menu.h

3.6 sliderVal Struct Reference

Struct to store left and right slider values.

`#include <joystick_driver.h>`

Collaboration diagram for sliderVal:



Public Attributes

- uint8_t l_val
- uint8_t r_val

3.6.1 Detailed Description

Struct to store left and right slider values.

3.6.2 Member Data Documentation

3.6.2.1 l_val

`uint8_t sliderVal::l_val`

3.6.2.2 r_val

```
uint8_t sliderVal::r_val
```

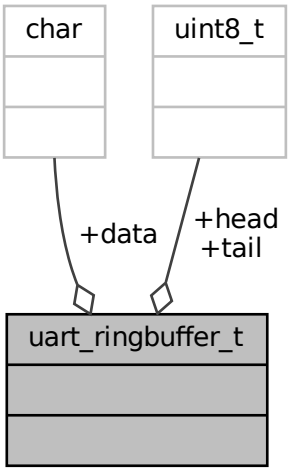
The documentation for this struct was generated from the following file:

- joystick_driver.h

3.7 uart_ringbuffer_t Struct Reference

```
#include <uart.h>
```

Collaboration diagram for uart_ringbuffer_t:



Public Attributes

- uint8_t head
- uint8_t tail
- char data [UART_RINGBUFFER_SIZE]

3.7.1 Member Data Documentation

3.7.1.1 data

```
char uart_ringbuffer_t::data[UART_RINGBUFFER_SIZE]
```

3.7.1.2 head

```
uint8_t uart_ringbuffer_t::head
```

3.7.1.3 tail

```
uint8_t uart_ringbuffer_t::tail
```

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

- GccApplication1/uart.h

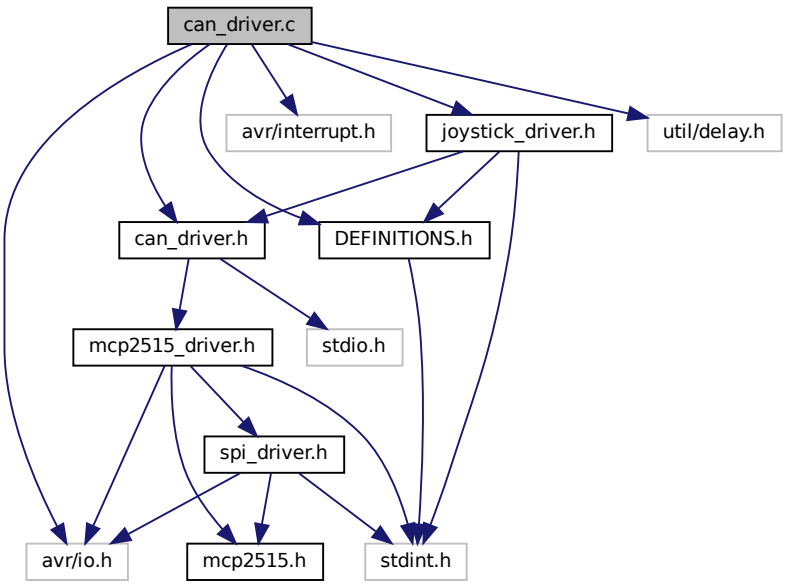
Chapter 4

File Documentation

4.1 can_driver.c File Reference

```
#include "can_driver.h"
#include "DEFINITIONS.h"
#include <avr/interrupt.h>
#include <avr/io.h>
#include "joystick_driver.h"
#include <util/delay.h>
```

Include dependency graph for can_driver.c:



Macros

- #define test_bit(reg, bit) (reg & (1 << bit))
- #define idBufferHighAddress 0x31
- #define idBufferLowAddress 0x32
- #define dataLengthBufferAddress 0x35
- #define dataBufferAddress 0x36

Functions

- ISR (INT0_vect)
- uint8_t can_interrupted ()

  Simple function to check if the CAN-ISR has set the flag high.
- void can_interrupt_enable ()

  Function to enable interrupt via CAN for the AtMega. Sets up the interrupt INT0 on port PD2/INT0.
- void can_init ()

  Initializes the CAN bus on the node by initializing the mcp2515 microcontroller and writing to its CAN control-registers.
- void send_can_msg (can_message *msg)

  Function to send a predetermined message to a reciever over the CAN bus.
- can_message * receive_can_msg (uint8_t buffer_number)

  Function to poll CAN bus for a message to be recieved.
- uint8_t can_check_complete (uint8_t buffer_number)

  Reads the "has been transmitted" - flag from the mcp2515 on the selected buffer.
- void send_stick_can ()

  Updates joystick and slider data and sends it over CAN.
- void send_difficulty_can (uint8_t diff)

  Sends a requested PID difficulty setting over CAN.
- void send_game_start_can ()

  Sends message to node 2 that game has started over CAN.
- void send_reaction_start_can ()

  Sends message to node 2 that reaction test has started over CAN.
- void send_reaction_stop_can ()

  Sends message to node 2 that reaction test has ended over CAN.

Variables

- uint8_t buffer_number = 0
- volatile uint8_t can_flag = 0

4.1.1 Macro Definition Documentation

4.1.1.1 dataBufferAddress

```
#define dataBufferAddress 0x36
```

4.1.1.2 dataLengthBufferAddress

```
#define dataLengthBufferAddress 0x35
```

4.1.1.3 idBufferHighAddress

```
#define idBufferHighAddress 0x31
```

4.1.1.4 idBufferLowAddress

```
#define idBufferLowAddress 0x32
```

4.1.1.5 test_bit

```
#define test_bit(
              reg,
              bit ) (reg & (1 << bit))
```

4.1.2 Function Documentation

4.1.2.1 can_check_complete()

```
uint8_t can_check_complete (
             uint8_t buffer_number )
```

Reads the "has been transmitted" - flag from the mcp2515 on the selected buffer.

Parameters

buffer_number	which buffer to check transmitted-flag on.

Returns

return 1 and resets interuptflag if transmitted, 0 if not

4.1.2.2 can_init()

```
void can_init ( )
```

Initializes the CAN bus on the node by initializing the mcp2515 microcontroller and writing to its CAN control-registers.

4.1.2.3 can_interrupt_enable()

```
void can_interrupt_enable ( )
```

Function to enable interrupt via CAN for the AtMega. Sets up the interrupt INT0 on port PD2/INT0.

4.1.2.4 can_interrupted()

```
uint8_t can_interrupted ( )
```

Simple function to check if the CAN-ISR has set the flag high.

Returns

1 if flag has been set. O if not

4.1.2.5 ISR()

```
ISR (
             INT0_vect  )
```

4.1.2.6 receive_can_msg()

```
can_message* receive_can_msg (
             uint8_t buffer_number )
```

Function to poll CAN bus for a message to be recieved.

Parameters

buffer_number	buffer to check for recieved message

Returns

returns a pointer to a recieved message now stored in memory

4.1.2.7 send_can_msg()

```
void send_can_msg (
             can_message * msg )
```

Function to send a predetermined message to a reciever over the CAN bus.

Parameters

msg	pointer to a memory location where a message is stored. Implemented as a pointer to make sure the variable will not exit scope.

4.1.2.8 send_difficulty_can()

```
void send_difficulty_can (
            uint8_t diff )
```

Sends a requested PID difficulty setting over CAN.

Parameters

diff	1 for easy, 2 for medium, 3 for hard. Defaults to medium

4.1.2.9 send_game_start_can()

```
void send_game_start_can ( )
```

Sends message to node 2 that game has started over CAN.

4.1.2.10 send_reaction_start_can()

```
void send_reaction_start_can ( )
```

Sends message to node 2 that reaction test has started over CAN.

4.1.2.11 send_reaction_stop_can()

```
void send_reaction_stop_can ( )
```

Sends message to node 2 that reaction test has ended over CAN.

4.1.2.12 send_stick_can()

```
void send_stick_can ( )
```

Updates joystick and slider data and sends it over CAN.

4.1.3 Variable Documentation

4.1.3.1 buffer_number

```
uint8_t buffer_number = 0
```

4.1.3.2 can_flag

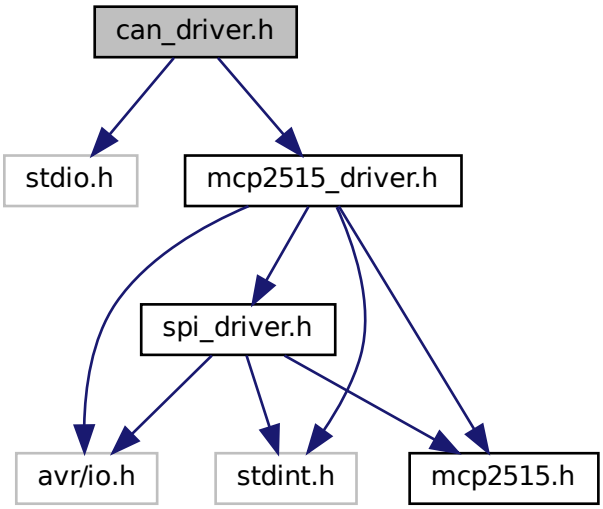
```
volatile uint8_t can_flag = 0
```

4.2 can_driver.h File Reference

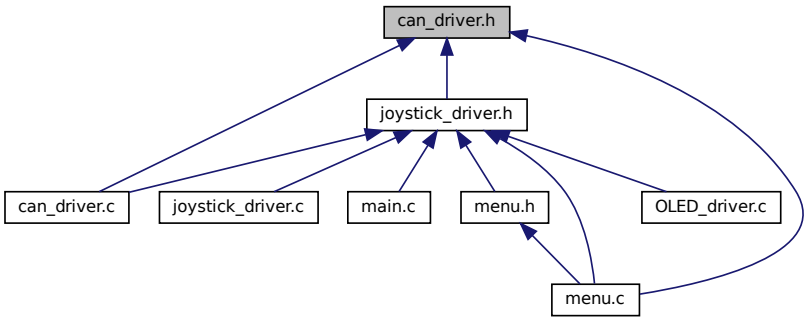
Driver module for handling data transmission over the CAN-bus.

```
#include <stdio.h>
#include "mcp2515_driver.h"
```

Include dependency graph for can_driver.h:



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



Functions

- void can_init ()

  Initializes the CAN bus on the node by initializing the mcp2515 microcontroller and writing to its CAN control-registers.
- void send_can_msg (can_message *msg)

  Function to send a predetermined message to a reciever over the CAN bus.
- can_message * receive_can_msg (uint8_t buffer_number)

  Function to poll CAN bus for a message to be recieved.
- uint8_t can_check_complete (uint8_t buffer_number)

  Reads the "has been transmitted" - flag from the mcp2515 on the selected buffer.
- uint8_t can_interrupted ()

  Simple function to check if the CAN-ISR has set the flag high.
- void can_interrupt_enable ()

  Function to enable interrupt via CAN for the AtMega. Sets up the interrupt INT0 on port PD2/INT0.
- void send_stick_can ()

  Updates joystick and slider data and sends it over CAN.
- void send_difficulty_can (uint8_t diff)

  Sends a requested PID difficulty setting over CAN.
- void send_game_start_can ()

  Sends message to node 2 that game has started over CAN.
- void send_reaction_start_can ()

  Sends message to node 2 that reaction test has started over CAN.
- void send_reaction_stop_can ()

  Sends message to node 2 that reaction test has ended over CAN.
- void send_pong_started ()
- void send_pong_ended ()

4.2.1 Detailed Description

Driver module for handling data transmission over the CAN-bus.

4.2.2 Function Documentation

4.2.2.1 can_check_complete()

```
uint8_t can_check_complete (
            uint8_t buffer_number )
```

Reads the "has been transmitted" - flag from the mcp2515 on the selected buffer.

Parameters

buffer_number	which buffer to check transmitted-flag on.

Returns

return 1 and resets interuptflag if transmitted, 0 if not

4.2.2.2 can_init()

```
void can_init ( )
```

Initializes the CAN bus on the node by initializing the mcp2515 microcontroller and writing to its CAN control-registers.

4.2.2.3 can_interrupt_enable()

```
void can_interrupt_enable ( )
```

Function to enable interrupt via CAN for the AtMega. Sets up the interrupt INT0 on port PD2/INT0.

4.2.2.4 can_interrupted()

```
uint8_t can_interrupted ( )
```

Simple function to check if the CAN-ISR has set the flag high.

Returns

1 if flag has been set. O if not

4.2.2.5 receive_can_msg()

```
can_message* receive_can_msg (
            uint8_t buffer_number )
```

Function to poll CAN bus for a message to be recieved.

Parameters

buffer_number	buffer to check for recieved message

Returns

returns a pointer to a recieved message now stored in memory

4.2.2.6 send_can_msg()

```
void send_can_msg (
            can_message * msg )
```

Function to send a predetermined message to a reciever over the CAN bus.

Parameters

msg	pointer to a memory location where a message is stored. Implemented as a pointer to make sure the variable will not exit scope.

4.2.2.7 send_difficulty_can()

```
void send_difficulty_can (
            uint8_t diff )
```

Sends a requested PID difficulty setting over CAN.

Parameters

diff	1 for easy, 2 for medium, 3 for hard. Defaults to medium

4.2.2.8 send_game_start_can()

```
void send_game_start_can ( )
```

Sends message to node 2 that game has started over CAN.

4.2.2.9 send_pong_ended()

```
void send_pong_ended ( )
```

4.2.2.10 send_pong_started()

```
void send_pong_started ( )
```

4.2.2.11 send_reaction_start_can()

```
void send_reaction_start_can ( )
```

Sends message to node 2 that reaction test has started over CAN.

4.2.2.12 send_reaction_stop_can()

```
void send_reaction_stop_can ( )
```

Sends message to node 2 that reaction test has ended over CAN.

4.2.2.13 send_stick_can()

```
void send_stick_can ( )
```

Updates joystick and slider data and sends it over CAN.

4.3 Debug/can_driver.d File Reference

4.4 Debug/joystick_driver.d File Reference

4.5 Debug/main.d File Reference

4.6 GccApplication1/Debug/main.d File Reference

4.7 Debug/mcp2515_driver.d File Reference

4.8 Debug/menu.d File Reference

4.9 Debug/OLED_driver.d File Reference

4.10 Debug/spi_driver.d File Reference

4.11 Debug/sram_test.d File Reference

4.12 Debug/system_states.d File Reference

4.13 Debug/timer.d File Reference

4.14 GccApplication1/Debug/timer.d File Reference

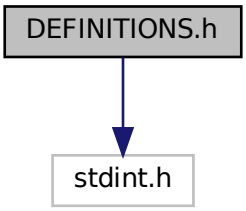
4.15 Debug/USART.d File Reference

4.16 DEFINITIONS.h File Reference

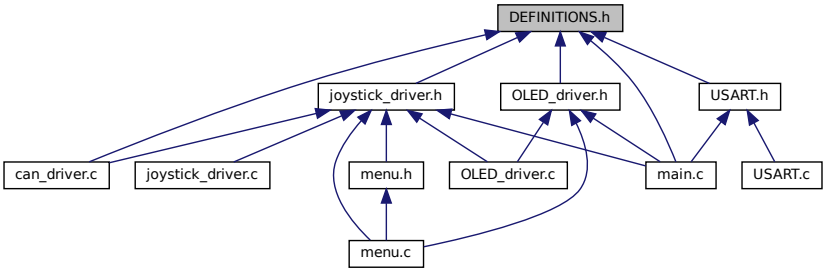
Definitions and constants to be used throughout the system such as clock speed, baud rate, memory mapping and basic bit manipulation functions.

```
#include <stdint.h>
```

Include dependency graph for DEFINITIONS.h:



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



Classes

- struct amap

Macros

- #define F_CPU 4915200
- #define BAUD 9600
- #define MYUBRR F_CPU/16/BAUD-1
- #define set_bit(reg, bit) (reg |= (1 << bit))
- #define clear_bit(reg, bit) (reg &= ~(1 << bit))

4.16.1 Detailed Description

Definitions and constants to be used throughout the system such as clock speed, baud rate, memory mapping and basic bit manipulation functions.

4.16.2 Macro Definition Documentation

4.16.2.1 BAUD

```
#define BAUD 9600
```

4.16.2.2 clear_bit

```
#define clear_bit(
              reg,
              bit ) (reg &= ~(1 << bit))
```

4.16.2.3 F_CPU

```
#define F_CPU 4915200
```

4.16.2.4 MYUBRR

```
#define MYUBRR F_CPU/16/BAUD-1
```

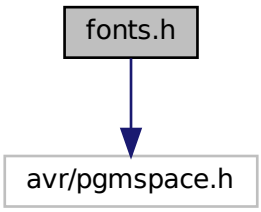
4.16.2.5 set_bit

```
#define set_bit(
              reg,
              bit ) (reg |= (1 << bit))
```

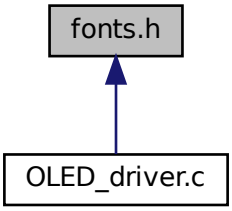
4.17 fonts.h File Reference

```
#include <avr/pgmspace.h>
```

Include dependency graph for fonts.h:



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



Variables

- const unsigned char PROGMEM font8 [95][8]
- const unsigned char PROGMEM font5 [95][5]
- const unsigned char PROGMEM font4 [95][4]
- PGM_P const font_array [ ] PROGMEM = {font8, font5, font4}

4.17.1 Variable Documentation

4.17.1.1 font4

```
const unsigned char PROGMEM font4[95][4]
```

4.17.1.2 font5

```
const unsigned char PROGMEM font5[95][5]
```

4.17.1.3 font8

```
const unsigned char PROGMEM font8[95][8]
```

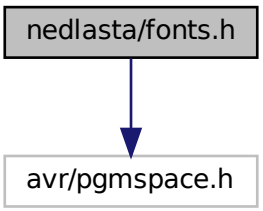
4.17.1.4 PROGMEM

```
PGM_P const font_array [] PROGMEM = {font8, font5, font4}
```

4.18 nedlasta/fonts.h File Reference

```
#include <avr/pgmspace.h>
```

Include dependency graph for fonts.h:



Variables

- const unsigned char PROGMEM font8 [95][8]
- const unsigned char PROGMEM font5 [95][5]
- const unsigned char PROGMEM font4 [95][4]

4.18.1 Variable Documentation

4.18.1.1 font4

```
const unsigned char PROGMEM font4[95][4]
```

4.18.1.2 font5

```
const unsigned char PROGMEM font5[95][5]
```

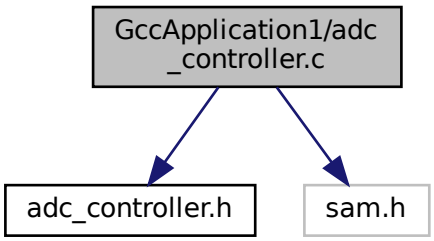
4.18.1.3 font8

```
const unsigned char PROGMEM font8[95][8]
```

4.19 GccApplication1/adc_controller.c File Reference

```
#include "adc_controller.h"
#include "sam.h"
```

Include dependency graph for adc_controller.c:



Functions

- void adc_init ()

  Initializes the adc by setting the right registers and also enables interrupt from the adc.

4.19.1 Function Documentation

4.19.1.1 adc_init()

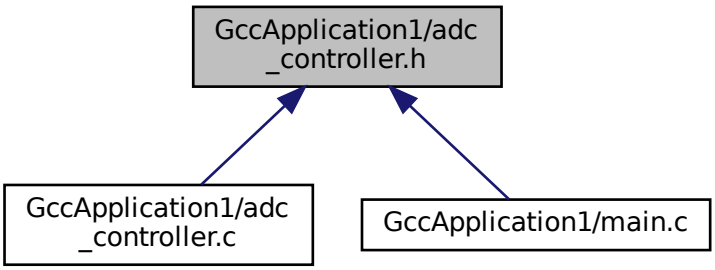
```
void adc_init ( )
```

Initializes the adc by setting the right registers and also enables interrupt from the adc.

4.20 GccApplication1/adc_controller.h File Reference

Module for handling adc initialization.

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



Functions

- void adc_init ()

  Initializes the adc by setting the right registers and also enables interrupt from the adc.

4.20.1 Detailed Description

Module for handling adc initialization.

4.20.2 Function Documentation

4.20.2.1 adc_init()

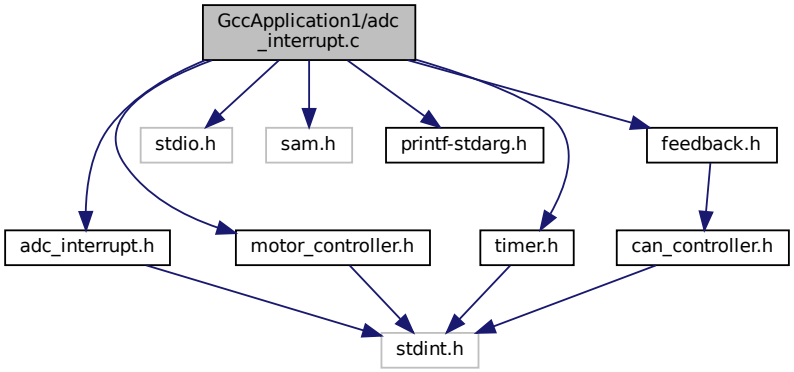
```
void adc_init ( )
```

Initializes the adc by setting the right registers and also enables interrupt from the adc.

4.21 GccApplication1/adc_interrupt.c File Reference

```
#include "adc_interrupt.h"
#include "motor_controller.h"
#include <stdio.h>
#include "sam.h"
#include "printf-stdarg.h"
#include "timer.h"
#include "feedback.h"
```

Include dependency graph for adc_interrupt.c:



Macros

- #define DEBUG_INTERRUPT 1

Functions

- uint8_t get_total_goals ()

  The function returns the number of goals you have.
- uint8_t get_goal_flag ()

  The function returns the goal flag.
- void reset_goal_flag ()

  The function resets the goal flag.
- void ADC_Handler (void)

  The function is called when there has been a goal, it stops the motor and sets the goal flag.

Variables

- uint8_t goal_flag = 0
- uint8_t TOTAL_GOALS = 0

4.21.1 Macro Definition Documentation

4.21.1.1 DEBUG_INTERRUPT

```
#define DEBUG_INTERRUPT 1
```

4.21.2 Function Documentation

4.21.2.1 ADC_Handler()

```
void ADC_Handler (
            void  )
```

The function is called when there has been a goal, it stops the motor and sets the goal flag.

4.21.2.2 get_goal_flag()

```
uint8_t get_goal_flag ( )
```

The function returns the goal flag.

Returns

Returns goal flag which is high if goal interrupt has been activated.

4.21.2.3 get_total_goals()

```
uint8_t get_total_goals ( )
```

The function returns the number of goals you have.

Returns

Returns total goals.

4.21.2.4 reset_goal_flag()

```
void reset_goal_flag ( )
```

The function resets the goal flag.

4.21.3 Variable Documentation

4.21.3.1 goal_flag

```
uint8_t goal_flag = 0
```

4.21.3.2 TOTAL_GOALS

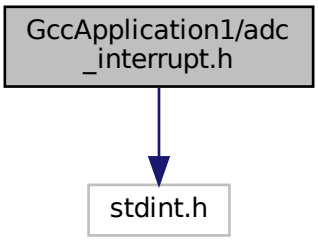
```
uint8_t TOTAL_GOALS = 0
```

4.22 GccApplication1/adc_interrupt.h File Reference

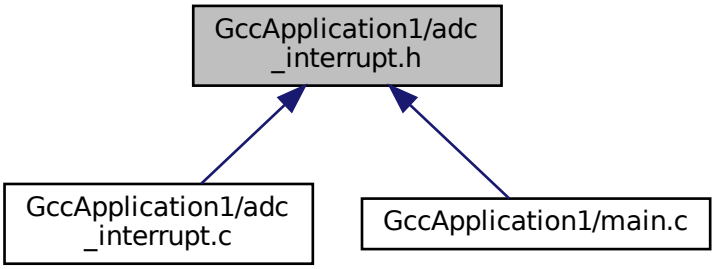
Module for handling goals while playing.

```
#include <stdint.h>
```

Include dependency graph for adc_interrupt.h:



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



Functions

- uint8_t get_total_goals ()

  The function returns the number of goals you have.
- uint8_t get_goal_flag ()

  The function returns the goal flag.
- void reset_goal_flag ()

  The function resets the goal flag.
- void ADC_Handler (void)

  The function is called when there has been a goal, it stops the motor and sets the goal flag.

4.22.1 Detailed Description

Module for handling goals while playing.

4.22.2 Function Documentation

4.22.2.1 ADC_Handler()

```
void ADC_Handler (
            void  )
```

The function is called when there has been a goal, it stops the motor and sets the goal flag.

4.22.2.2 get_goal_flag()

```
uint8_t get_goal_flag ( )
```

The function returns the goal flag.

Returns

Returns goal flag which is high if goal interrupt has been activated.

4.22.2.3 get_total_goals()

```
uint8_t get_total_goals ( )
```

The function returns the number of goals you have.

Returns

Returns total goals.

4.22.2.4 reset_goal_flag()

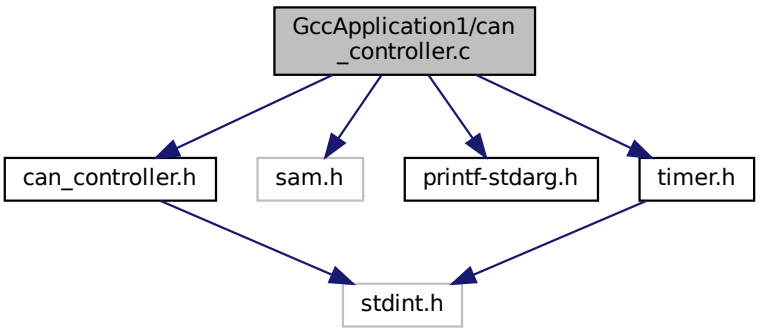
```
void reset_goal_flag ( )
```

The function resets the goal flag.

4.23 GccApplication1/can_controller.c File Reference

```
#include "can_controller.h"
#include "sam.h"
#include "printf-stdarg.h"
#include "timer.h"
```

Include dependency graph for can_controller.c:



Functions

- uint8_t can_init_def_tx_rx_mb (uint32_t can_br)

  Initialize can bus with predefined number of rx and tx mailboxes, CAN0->CAN_MB[0] is used for transmitting CAN0->CAN_MB[1,2] is used for receiving.
- uint8_t can_init (uint32_t can_br, uint8_t num_tx_mb, uint8_t num_rx_mb)

  Initialize can bus.
- uint8_t can_send (CAN_MESSAGE *can_msg, uint8_t tx_mb_id)

  Send can message from mailbox.
- uint8_t can_receive (CAN_MESSAGE *can_msg, uint8_t rx_mb_id)

  Read can message from mailbox.

4.23.1 Function Documentation

4.23.1.1 can_init()

```
uint8_t can_init (
            uint32_t can_br,
            uint8_t num_tx_mb,
            uint8_t num_rx_mb )
```

Initialize can bus.

Parameters

can_br	Value to be set in CAN0->CAN_BR register to match can bus bit timing
num_tx_mb	Number of transmit mailboxes, tx mb indexes: [0 , num_tx_mb-1]
num_rx_mb	Number of receive mailboxes, rx mb indexes: [num_tx_mb, num_rx_mb-1]

Return values

Success(0)	or failure(1)

4.23.1.2 can_init_def_tx_rx_mb()

```
uint8_t can_init_def_tx_rx_mb (
            uint32_t can_br )
```

Initialize can bus with predefined number of rx and tx mailboxes, CAN0->CAN_MB[0] is used for transmitting CA↩ N0->CAN_MB[1,2] is used for receiving.

Parameters

can↩ _br	Value to be set in CAN0->CAN_BR register to match can bus bit timing

Return values

Success(0)	or failure(1)

4.23.1.3 can_receive()

```
uint8_t can_receive (
            CAN_MESSAGE * can_msg,
            uint8_t rx_mb_id )
```

Read can message from mailbox.

Parameters

can_msg	struct instance to save received data
rx_mb↩ _id	ID of receive mailbox to be used

Return values

Success(0)	or failure(1)

4.23.1.4 can_send()

```
uint8_t can_send (
            CAN_MESSAGE * can_msg,
            uint8_t tx_mb_id )
```

Send can message from mailbox.

Parameters

can_msg	message to be sent
tx_mb↩ _id	ID of transmit mailbox to be used

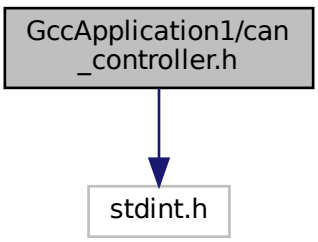
Return values

Success(0)	or failure(1)

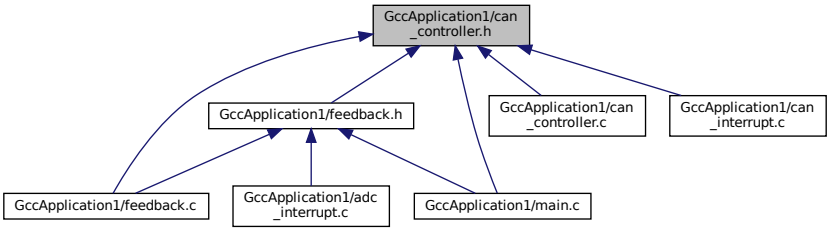
4.24 GccApplication1/can_controller.h File Reference

Module for initializing CAN and controls how CAN messages are sent and received.

```
#include <stdint.h>
```
Include dependency graph for can_controller.h:



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



Classes

- struct can_message_t

  Contains the data sent with CAN messages.

Typedefs

- typedef struct can_message_t CAN_MESSAGE

  Contains the data sent with CAN messages.

Functions

- uint8_t can_init_def_tx_rx_mb (uint32_t can_br)

  Initialize can bus with predefined number of rx and tx mailboxes, CAN0->CAN_MB[0] is used for transmitting CAN0->CAN_MB[1,2] is used for receiving.
- uint8_t can_init (uint32_t can_br, uint8_t num_tx_mb, uint8_t num_rx_mb)

  Initialize can bus.
- uint8_t can_send (CAN_MESSAGE *can_msg, uint8_t mailbox_id)

  Send can message from mailbox.
- uint8_t can_receive (CAN_MESSAGE *can_msg, uint8_t mailbox_id)

  Read can message from mailbox.

4.24.1 Detailed Description

Module for initializing CAN and controls how CAN messages are sent and received.

4.24.2 Typedef Documentation

4.24.2.1 CAN_MESSAGE

```
typedef struct can_message_t CAN_MESSAGE
```

Contains the data sent with CAN messages.

4.24.3 Function Documentation

4.24.3.1 can_init()

```
uint8_t can_init (
            uint32_t can_br,
            uint8_t num_tx_mb,
            uint8_t num_rx_mb )
```

Initialize can bus.

Parameters

can_br	Value to be set in CAN0->CAN_BR register to match can bus bit timing
num_tx_mb	Number of transmit mailboxes, tx mb indexes: [0 , num_tx_mb-1]
num_rx_mb	Number of receive mailboxes, rx mb indexes: [num_tx_mb, num_rx_mb-1]

Returns

Success(0) or failure(1)

Parameters

can_br	Value to be set in CAN0->CAN_BR register to match can bus bit timing
num_tx_mb	Number of transmit mailboxes, tx mb indexes: [0 , num_tx_mb-1]
num_rx_mb	Number of receive mailboxes, rx mb indexes: [num_tx_mb, num_rx_mb-1]

Return values

Success(0)	or failure(1)

4.24.3.2 can_init_def_tx_rx_mb()

```
uint8_t can_init_def_tx_rx_mb (
            uint32_t can_br )
```

Initialize can bus with predefined number of rx and tx mailboxes, CAN0->CAN_MB[0] is used for transmitting CA↩ N0->CAN_MB[1,2] is used for receiving.

Parameters

can↩ _br	Value to be set in CAN0->CAN_BR register to match can bus bit timing

Returns

Success(0) or failure(1)

Parameters

can_br	Value to be set in CAN0->CAN_BR register to match can bus bit timing

Return values

Success(0)	or failure(1)

4.24.3.3 can_receive()

```
uint8_t can_receive (
            CAN_MESSAGE * can_msg,
            uint8_t rx_mb_id )
```

Read can message from mailbox.

Parameters

can_msg	struct instance to save received data
rx_mb_id	ID of receive mailbox to be used

Returns

Success(0) or failure(1)

Parameters

can_msg	struct instance to save received data
rx_mb_id	ID of receive mailbox to be used

Return values

Success(0)	or failure(1)

4.24.3.4 can_send()

```
uint8_t can_send (
```

```
CAN_MESSAGE * can_msg,
uint8_t tx_mb_id )
```

Send can message from mailbox.

Parameters

can_msg	message to be sent
tx_mb↩ _id	ID of transmit mailbox to be used

Returns

Success(0) or failure(1)

Parameters

can_msg	message to be sent
tx_mb↩ _id	ID of transmit mailbox to be used

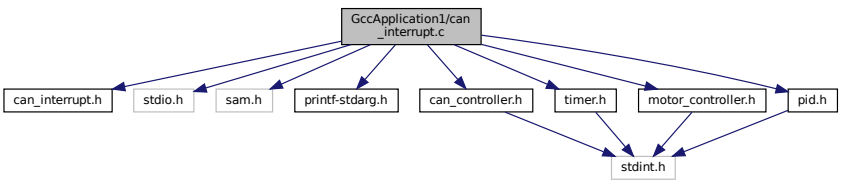
Return values

Success(0)	or failure(1)

4.25 GccApplication1/can_interrupt.c File Reference

```
#include "can_interrupt.h"
#include <stdio.h>
#include "sam.h"
#include "printf-stdarg.h"
#include "can_controller.h"
#include "timer.h"
#include "motor_controller.h"
#include "pid.h"
```

Include dependency graph for can_interrupt.c:



Macros

- #define DEBUG_INTERRUPT 1

Functions

- void CAN0_Handler (void)

  CAN0 Interrupt handler for RX, TX and bus error interrupts.

Variables

- uint16_t starttime = 0

4.25.1 Macro Definition Documentation

4.25.1.1 DEBUG_INTERRUPT

```
#define DEBUG_INTERRUPT 1
```

4.25.2 Function Documentation

4.25.2.1 CAN0_Handler()

```
void CAN0_Handler (
            void  )
```

CAN0 Interrupt handler for RX, TX and bus error interrupts.

Decide what happens to messages of different IDs.

Parameters

void	

Return values

4.25.3 Variable Documentation

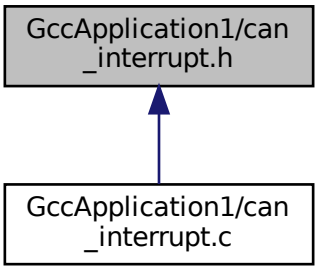
4.25.3.1 starttime

```
uint16_t starttime = 0
```

4.26 GccApplication1/can_interrupt.h File Reference

Module for handling what happens to all CAN messages from node 1.

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



Functions

- void CAN0_Handler (void)

  Decide what happens to messages of different IDs.

4.26.1 Detailed Description

Module for handling what happens to all CAN messages from node 1.

4.26.2 Function Documentation

4.26.2.1 CAN0_Handler()

```
void CAN0_Handler (
             void  )
```

Decide what happens to messages of different IDs.

Decide what happens to messages of different IDs.

Parameters

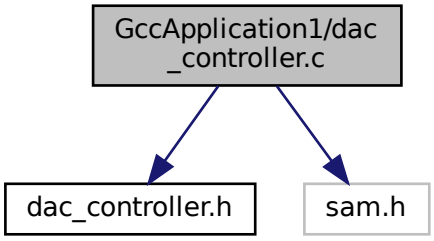
void	

Return values

4.27 GccApplication1/dac_controller.c File Reference

```
#include "dac_controller.h"
#include "sam.h"
```

Include dependency graph for dac_controller.c:



Functions

- void dac_init ()

  Initialize the dac.

4.27.1 Function Documentation

4.27.1.1 dac_init()

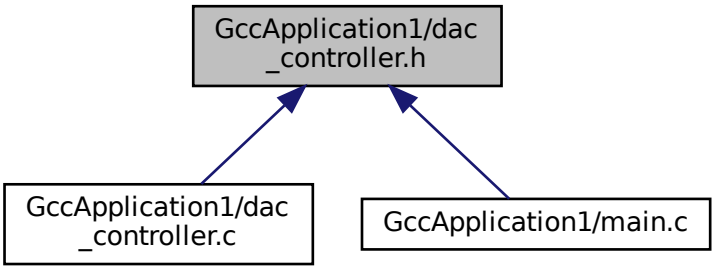
```
void dac_init ( )
```

Initialize the dac.

4.28 GccApplication1/dac_controller.h File Reference

Module for handling dac initialization.

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



Functions

- void dac_init ()

  Initialize the dac.

4.28.1 Detailed Description

Module for handling dac initialization.

4.28.2 Function Documentation

4.28.2.1 dac_init()

```
void dac_init ( )
```

Initialize the dac.

4.29 GccApplication1/Debug/adc_controller.d File Reference

4.30 GccApplication1/Debug/adc_interrupt.d File Reference

4.31 GccApplication1/Debug/can_controller.d File Reference

4.32 GccApplication1/Debug/can_interrupt.d File Reference

4.33 GccApplication1/Debug/dac_controller.d File Reference

4.34 GccApplication1/Debug/Device_Startup/startup_sam3xa.d File Reference

4.35 GccApplication1/Debug/Device_Startup/system_sam3xa.d File Reference

4.36 GccApplication1/Debug/feedback.d File Reference

4.37 GccApplication1/Debug/joystick.d File Reference

4.38 GccApplication1/Debug/motor_controller.d File Reference

4.39 GccApplication1/Debug/pid.d File Reference

4.40 GccApplication1/Debug/printf-stdarg.d File Reference

4.41 GccApplication1/Debug/timer_driver.d File Reference

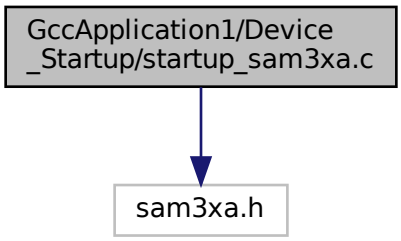
4.42 GccApplication1/Debug/uart.d File Reference

4.43 GccApplication1/Debug/usart.d File Reference

4.44 GccApplication1/Device_Startup/startup_sam3xa.c File Reference

```
#include "sam3xa.h"
```

Include dependency graph for startup_sam3xa.c:



Functions

- void __libc_init_array (void)
- void Dummy_Handler (void)

  Default interrupt handler for unused IRQs.
- void NMI_Handler (void HardFault_Handler void)
- void Reset_Handler (void)

  This is the code that gets called on processor reset. To initialize the device, and call the main() routine.

Variables

- uint32_t _sfixed
- uint32_t _efixed
- uint32_t _etext
- uint32_t _srelocate
- uint32_t _erelocate
- uint32_t _szero
- uint32_t _ezero
- uint32_t _sstack
- uint32_t _estack

4.44.1 Function Documentation

4.44.1.1 __libc_init_array()

```
void __libc_init_array (
            void  )
```

4.44.1.2 Dummy_Handler()

```
void Dummy_Handler (
             void  )
```

Default interrupt handler for unused IRQs.

4.44.1.3 NMI_Handler()

```
void NMI_Handler (
             void HardFault_Handler void )
```

4.44.1.4 Reset_Handler()

```
void Reset_Handler (
             void  )
```

This is the code that gets called on processor reset. To initialize the device, and call the main() routine.

4.44.2 Variable Documentation

4.44.2.1 _efixed

```
uint32_t _efixed
```

4.44.2.2 _erelocate

```
uint32_t _erelocate
```

4.44.2.3 _estack

```
uint32_t _estack
```

4.44.2.4 _etext

```
uint32_t _etext
```

4.44.2.5 _ezero

```
uint32_t _ezero
```

4.44.2.6 _sfixed

```
uint32_t _sfixed
```

4.44.2.7 _srelocate

```
uint32_t _srelocate
```

4.44.2.8 _sstack

```
uint32_t _sstack
```

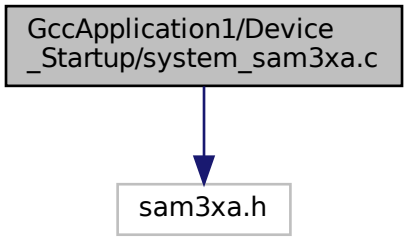
4.44.2.9 _szero

```
uint32_t _szero
```

4.45 GccApplication1/Device_Startup/system_sam3xa.c File Reference

```
#include "sam3xa.h"
```

Include dependency graph for system_sam3xa.c:



Macros

- #define SYS_BOARD_OSCOUNT (CKGR_MOR_MOSCXTST(0x8))
- #define SYS_BOARD_PLLAR (CKGR_PLLAR_ONE | CKGR_PLLAR_MULA(0xdUL) | CKGR_PLLAR_PL↩ LACOUNT(0x3fUL) | CKGR_PLLAR_DIVA(0x1UL))
- #define SYS_BOARD_MCKR (PMC_MCKR_PRES_CLK_2 | PMC_MCKR_CSS_PLLA_CLK)

Functions

- void SystemInit (void)

  Setup the microcontroller system. Initialize the System and update the SystemFrequency variable.
- void SystemCoreClockUpdate (void)
- void system_init_flash (uint32_t dw_clk)

Variables

- uint32_t SystemCoreClock = CHIP_FREQ_MAINCK_RC_4MHZ

4.45.1 Macro Definition Documentation

4.45.1.1 SYS_BOARD_MCKR

```
#define SYS_BOARD_MCKR (PMC_MCKR_PRES_CLK_2 | PMC_MCKR_CSS_PLLA_CLK)
```

4.45.1.2 SYS_BOARD_OSCOUNT

```
#define SYS_BOARD_OSCOUNT (CKGR_MOR_MOSCXTST(0x8))
```

4.45.1.3 SYS_BOARD_PLLAR

```
#define SYS_BOARD_PLLAR (CKGR_PLLAR_ONE | CKGR_PLLAR_MULA(0xdUL) | CKGR_PLLAR_PLLACOUNT(0x3fUL)
| CKGR_PLLAR_DIVA(0x1UL))
```

4.45.2 Function Documentation

4.45.2.1 system_init_flash()

```
void system_init_flash (
            uint32_t dw_clk )
```

Initialize flash.

4.45.2.2 SystemCoreClockUpdate()

```
void SystemCoreClockUpdate (
            void  )
```

4.45.2.3 SystemInit()

```
void SystemInit (
            void  )
```

Setup the microcontroller system. Initialize the System and update the SystemFrequency variable.

4.45.3 Variable Documentation

4.45.3.1 SystemCoreClock

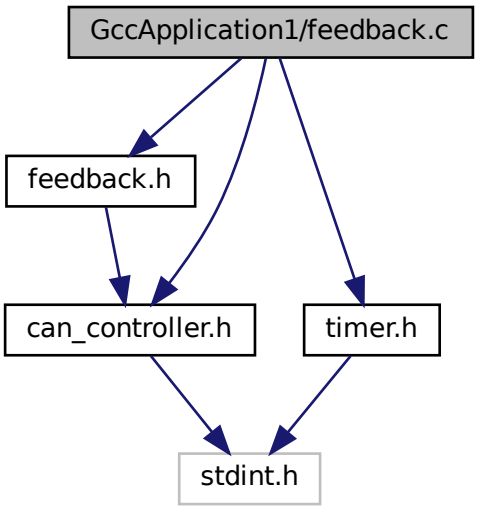
```
uint32_t SystemCoreClock = CHIP_FREQ_MAINCK_RC_4MHZ
```

4.46 GccApplication1/feedback.c File Reference

```
#include "feedback.h"
#include "can_controller.h"
```

```
#include "timer.h"
```

Include dependency graph for feedback.c:



Functions

- void send_time_to_node_1 (CAN_MESSAGE *msgToSend)

  Send amount of time node 2 has been active.
- void send_goal_to_node_1 (CAN_MESSAGE *msgToSend)

  Notifies node 1 that a goal has been scored and at which time.
- void send_motor_info_to_node_1 (CAN_MESSAGE *msgToSend, uint8_t y_pos, uint8_t solenoide)

  Send motor position to node 1.
- void send_reaction_time_to_node_1 (CAN_MESSAGE *msgToSend, uint16_t ms)

  Send reaction time to node 1.

4.46.1 Function Documentation

4.46.1.1 send_goal_to_node_1()

```
void send_goal_to_node_1 (
            CAN_MESSAGE * msgToSend )
```

Notifies node 1 that a goal has been scored and at which time.

Parameters

msgToSend	Pointer to a CAN_MESSAGE allocated in memory

4.46.1.2 send_motor_info_to_node_1()

```
void send_motor_info_to_node_1 (
            CAN_MESSAGE * msgToSend,
            uint8_t y_pos,
            uint8_t solenoide )
```

Send motor position to node 1.

Parameters

msgToSend	Pointer to a CAN_MESSAGE allocated in memory
y_pos	Position of the motor
solenoid	1 if shot, else 0

4.46.1.3 send_reaction_time_to_node_1()

```
void send_reaction_time_to_node_1 (
            CAN_MESSAGE * msgToSend,
            uint16_t ms )
```

Send reaction time to node 1.

Parameters

msgToSend	Pointer to a CAN_MESSAGE allocated in memory
ms	Amount of ms used to react

4.46.1.4 send_time_to_node_1()

```
void send_time_to_node_1 (
            CAN_MESSAGE * msgToSend )
```

Send amount of time node 2 has been active.

Parameters

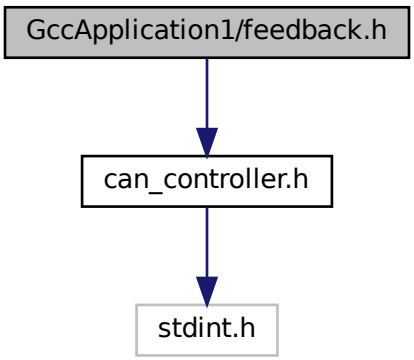
msgToSend	Pointer to a CAN_MESSAGE allocated in memory

4.47 GccApplication1/feedback.h File Reference

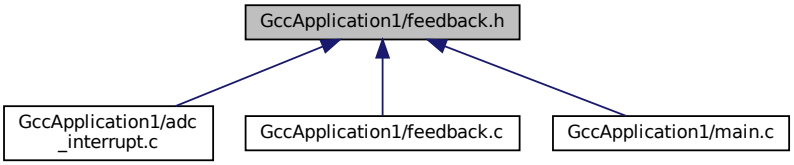
Module for sending CAN messages over CAN bus back to node 1.

```
#include "can_controller.h"
```

Include dependency graph for feedback.h:



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



Functions

- void send_time_to_node_1 (CAN_MESSAGE *msgToSend)

  Send amount of time node 2 has been active.
- void send_goal_to_node_1 (CAN_MESSAGE *msgToSend)

  Notifies node 1 that a goal has been scored and at which time.
- void send_motor_info_to_node_1 (CAN_MESSAGE *msgToSend, uint8_t y_pos, uint8_t solenoide)

  Send motor position to node 1.
- void send_reaction_time_to_node_1 (CAN_MESSAGE *msgToSend, uint16_t ms)

  Send reaction time to node 1.

4.47.1 Detailed Description

Module for sending CAN messages over CAN bus back to node 1.

4.47.2 Function Documentation

4.47.2.1 send_goal_to_node_1()

```
void send_goal_to_node_1 (
            CAN_MESSAGE * msgToSend )
```

Notifies node 1 that a goal has been scored and at which time.

Parameters

msgToSend	Pointer to a CAN_MESSAGE allocated in memory

4.47.2.2 send_motor_info_to_node_1()

```
void send_motor_info_to_node_1 (
            CAN_MESSAGE * msgToSend,
            uint8_t y_pos,
            uint8_t solenoide )
```

Send motor position to node 1.

Parameters

msgToSend	Pointer to a CAN_MESSAGE allocated in memory
y_pos	Position of the motor
solenoid	1 if shot, else 0

4.47.2.3 send_reaction_time_to_node_1()

```
void send_reaction_time_to_node_1 (
            CAN_MESSAGE * msgToSend,
            uint16_t ms )
```

Send reaction time to node 1.

Parameters

msgToSend	Pointer to a CAN_MESSAGE allocated in memory
ms	Amount of ms used to react

4.47.2.4 send_time_to_node_1()

```
void send_time_to_node_1 (
            CAN_MESSAGE * msgToSend )
```

Send amount of time node 2 has been active.

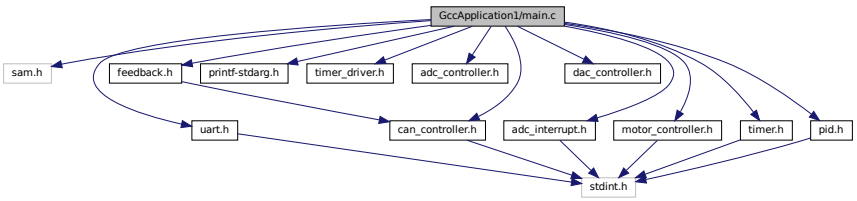
Parameters

msgToSend	Pointer to a CAN_MESSAGE allocated in memory

4.48 GccApplication1/main.c File Reference

```
#include "sam.h"
#include "can_controller.h"
#include "uart.h"
#include "printf-stdarg.h"
#include "timer_driver.h"
#include "adc_controller.h"
#include "adc_interrupt.h"
#include "dac_controller.h"
#include "motor_controller.h"
#include "timer.h"
#include "feedback.h"
#include "pid.h"
```

Include dependency graph for main.c:



Functions

- int main (void)

4.48.1 Function Documentation

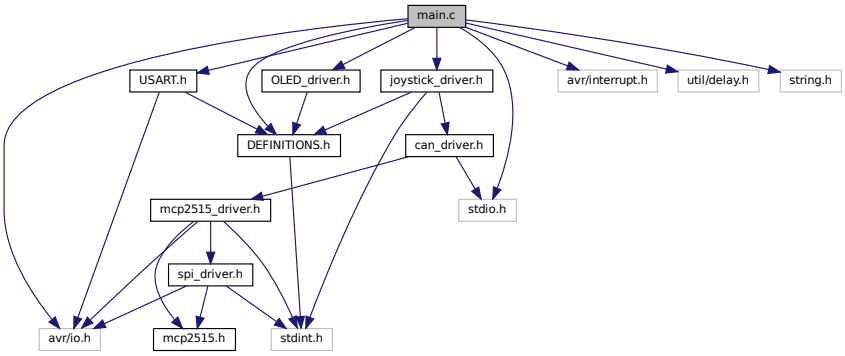
4.48.1.1 main()

```
int main (
            void  )
```

4.49 main.c File Reference

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "DEFINITIONS.h"
#include <util/delay.h>
#include "USART.h"
#include <stdio.h>
#include <string.h>
#include "joystick_driver.h"
#include "OLED_driver.h"
```

Include dependency graph for main.c:



Functions

- void led_test (void)
- int main (void)

4.49.1 Function Documentation

4.49.1.1 led_test()

```
void led_test (
            void  )
```

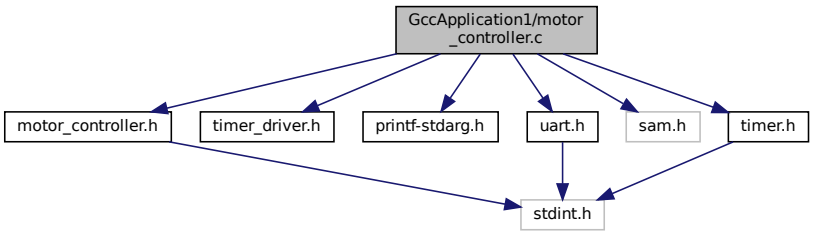
4.49.1.2 main()

```
int main (
            void  )
```

4.50 GccApplication1/motor_controller.c File Reference

```
#include "motor_controller.h"
#include "timer_driver.h"
#include "printf-stdarg.h"
#include "uart.h"
#include "sam.h"
#include "timer.h"
```

Include dependency graph for motor_controller.c:



Functions

- void set_pi_value (uint8_t val)

  Set pi value (motor position) to be used in pid regulator.
- uint8_t get_pi_value ()

  Return the current pi value (motor position)
- uint8_t get_solenoid_status ()

  Return whether solenoid has shot or not.
- void reset_solenoid_status ()

  Reset solenoid status.
- void move_servo ()

  Move servo based on joystick values.
- void check_solenoid_shot ()

  Check if the solenoid has shot.
- void change_motor_speed ()

  Change motor speed based on joystick position.
- void change_motor_speed_using_paadrag (int paadrag)

  Change motor speed using the paadrag from the pid regulator.
- void motor_box_init ()

  Initialize the motor box.
- void encoder_read ()

  Read the encoder values to find motor position.
- uint8_t button_check (uint8_t current)

  Check if the button is pressed.

Variables

- uint8_t previous = 1
- uint8_t y_value_pi = 0
- uint8_t solenoide_status = 0

4.50.1 Function Documentation

4.50.1.1 button_check()

```
uint8_t button_check (
            uint8_t current )
```

Check if the button is pressed.

Function to check if one of the three buttons is being pressed. Uses a local previous variable to make sure an input is registered as just one input.

Parameters

current	Current value of the button

Returns

1 if pressed, else 0

4.50.1.2 change_motor_speed()

```
void change_motor_speed ( )
```

Change motor speed based on joystick position.

4.50.1.3 change_motor_speed_using_paadrag()

```
void change_motor_speed_using_paadrag (
            int paadrag )
```

Change motor speed using the paadrag from the pid regulator.

Parameters

paadrag	Value to be sent to motor box, deciding the speed of the motor

4.50.1.4 check_solenoid_shot()

```
void check_solenoid_shot ( )
```

Check if the solenoid has shot.

4.50.1.5 encoder_read()

```
void encoder_read ( )
```

Read the encoder values to find motor position.

OE low

RST low

RST high

OE high

4.50.1.6 get_pi_value()

```
uint8_t get_pi_value ( )
```

Return the current pi value (motor position)

Returns

returns the current pi value

4.50.1.7 get_solenoid_status()

```
uint8_t get_solenoid_status ( )
```

Return whether solenoid has shot or not.

Returns

shot(1) else 0

4.50.1.8 motor_box_init()

```
void motor_box_init ( )
```

Initialize the motor box.

RST high

4.50.1.9 move_servo()

```
void move_servo ( )
```

Move servo based on joystick values.

4.50.1.10 reset_solenoid_status()

```
void reset_solenoid_status ( )
```

Reset solenoid status.

4.50.1.11 set_pi_value()

```
void set_pi_value (
            uint8_t val )
```

Set pi value (motor position) to be used in pid regulator.

Parameters

val	Value to be set

4.50.2 Variable Documentation

4.50.2.1 previous

```
uint8_t previous = 1
```

4.50.2.2 solenoide_status

```
uint8_t solenoide_status = 0
```

4.50.2.3 y_value_pi

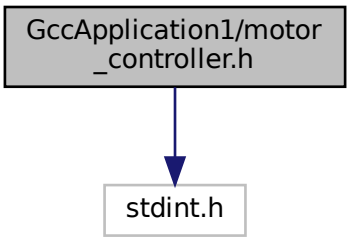
```
uint8_t y_value_pi = 0
```

4.51 GccApplication1/motor_controller.h File Reference

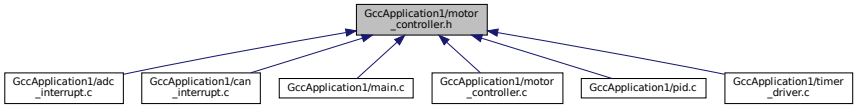
Module for handling all things related to controlling the motor.

```
#include <stdint.h>
```

Include dependency graph for motor_controller.h:



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



Classes

- struct joyVal

  Send amount of time node 2 has been active.

Functions

- void set_pi_value (uint8_t val)

  Set pi value (motor position) to be used in pid regulator.
- uint8_t get_pi_value ()

  Return the current pi value (motor position)
- uint8_t get_solenoid_status ()

  Return whether solenoid has shot or not.
- void reset_solenoid_status ()

  Reset solenoid status.
- void move_servo ()

  Move servo based on joystick values.
- void check_solenoid_shot ()

  Check if the solenoid has shot.

- void change_motor_speed ()

  Change motor speed based on joystick position.
- void change_motor_speed_using_paadrag (int paadrag)

  Change motor speed using the paadrag from the pid regulator.
- void motor_box_init ()

  Initialize the motor box.
- void encoder_read ()

  Read the encoder values to find motor position.
- uint8_t button_check (uint8_t current)

  Check if the button is pressed.

Variables

- joyVal joystick

4.51.1 Detailed Description

Module for handling all things related to controlling the motor.

4.51.2 Function Documentation

4.51.2.1 button_check()

```
uint8_t button_check (
            uint8_t current )
```

Check if the button is pressed.

Parameters

current	Current value of the button

Returns

1 if pressed, else 0

4.51.2.2 change_motor_speed()

```
void change_motor_speed ( )
```

Change motor speed based on joystick position.

4.51.2.3 change_motor_speed_using_paadrag()

```
void change_motor_speed_using_paadrag (
            int paadrag )
```

Change motor speed using the paadrag from the pid regulator.

Parameters

paadrag	Value to be sent to motor box, deciding the speed of the motor

4.51.2.4 check_solenoid_shot()

```
void check_solenoid_shot ( )
```

Check if the solenoid has shot.

4.51.2.5 encoder_read()

```
void encoder_read ( )
```

Read the encoder values to find motor position.

OE low

RST low

RST high

OE high

4.51.2.6 get_pi_value()

```
uint8_t get_pi_value ( )
```

Return the current pi value (motor position)

Returns

returns the current pi value

4.51.2.7 get_solenoid_status()

```
uint8_t get_solenoid_status ( )
```

Return whether solenoid has shot or not.

Returns

shot(1) else 0

4.51.2.8 motor_box_init()

```
void motor_box_init ( )
```

Initialize the motor box.

RST high

4.51.2.9 move_servo()

```
void move_servo ( )
```

Move servo based on joystick values.

4.51.2.10 reset_solenoid_status()

```
void reset_solenoid_status ( )
```

Reset solenoid status.

4.51.2.11 set_pi_value()

```
void set_pi_value (
            uint8_t val )
```

Set pi value (motor position) to be used in pid regulator.

Parameters

val	Value to be set

4.51.3 Variable Documentation

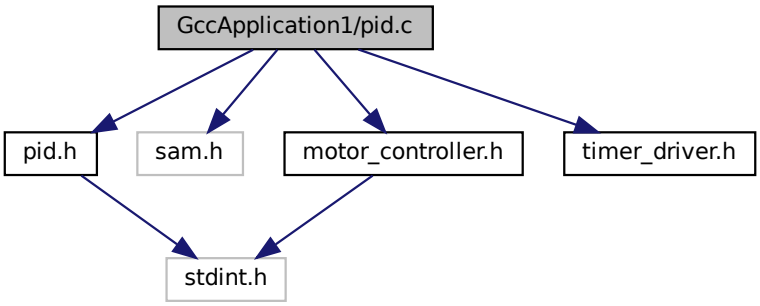
4.51.3.1 joystick

```
joyVal joystick
```

4.52 GccApplication1/pid.c File Reference

```
#include "pid.h"
#include "sam.h"
#include "motor_controller.h"
#include "timer_driver.h"
```

Include dependency graph for pid.c:



Functions

- void TC1_Handler (void)

  Function called x amount per second based on timer initialized in timer_driver, drives the pid regulator.
- uint8_t get_difficulty ()

  Return the current difficulty setting.
- void set_difficulty (uint8_t difficulty_to_set)

  Set the difficulty.
- void stop_pid ()
- void start_pid ()

Variables

- double prev_error = 0
- double error = 0
- int paadrag = 0
- double kp = 20
- double ki = 20
- double kd = 1
- double sum_error = 0
- double T_periode = 0.02
- int active = 0
- uint8_t difficulty = 0

4.52.1 Function Documentation

4.52.1.1 get_difficulty()

```
uint8_t get_difficulty ( )
```

Return the current difficulty setting.

Returns

Returns the current difficulty setting

4.52.1.2 set_difficulty()

```
void set_difficulty (
            uint8_t difficulty_to_set )
```

Set the difficulty.

Parameters

difficulty_to_set	Chosen difficulty to play with

4.52.1.3 start_pid()

```
void start_pid ( )
```

4.52.1.4 stop_pid()

```
void stop_pid ( )
```

4.52.1.5 TC1_Handler()

```
void TC1_Handler (
            void  )
```

Function called x amount per second based on timer initialized in timer_driver, drives the pid regulator.

4.52.2 Variable Documentation

4.52.2.1 active

```
int active = 0
```

4.52.2.2 difficulty

```
uint8_t difficulty = 0
```

4.52.2.3 error

```
double error = 0
```

4.52.2.4 kd

```
double kd = 1
```

4.52.2.5 ki

```
double ki = 20
```

4.52.2.6 kp

```
double kp = 20
```

4.52.2.7 paadrag

```
int paadrag = 0
```

4.52.2.8 prev_error

```
double prev_error = 0
```

4.52.2.9 sum_error

```
double sum_error = 0
```

4.52.2.10 T_periode

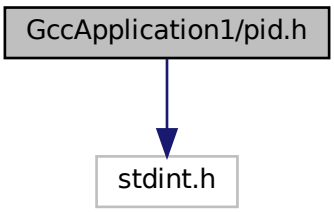
```
double T_periode = 0.02
```

4.53 GccApplication1/pid.h File Reference

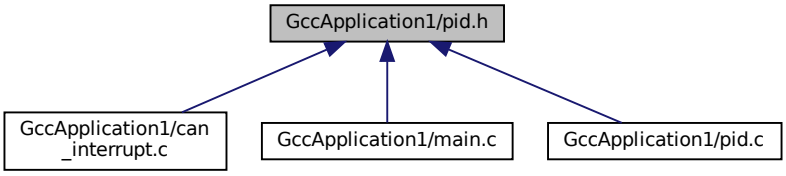
Module for handling the pid regulator.

```
#include <stdint.h>
```

Include dependency graph for pid.h:



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



Functions

- void TC1_Handler (void)

  Function called x amount per second based on timer initialized in timer_driver, drives the pid regulator.
- uint8_t get_difficulty ()

  Return the current difficulty setting.
- void set_difficulty (uint8_t difficulty_to_set)

  Set the difficulty.
- void stop_pid ()
- void start_pid ()

4.53.1 Detailed Description

Module for handling the pid regulator.

4.53.2 Function Documentation

4.53.2.1 get_difficulty()

```
uint8_t get_difficulty ( )
```

Return the current difficulty setting.

Returns

Returns the current difficulty setting

4.53.2.2 set_difficulty()

```
void set_difficulty (
            uint8_t difficulty_to_set )
```

Set the difficulty.

Parameters

difficulty_to_set	Chosen difficulty to play with

4.53.2.3 start_pid()

```
void start_pid ( )
```

4.53.2.4 stop_pid()

```
void stop_pid ( )
```

4.53.2.5 TC1_Handler()

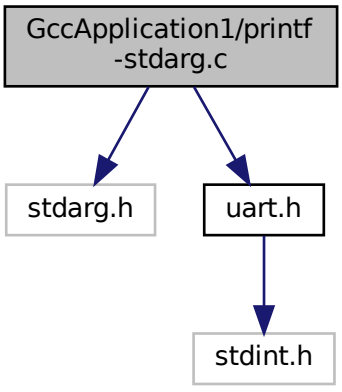
```
void TC1_Handler (
             void  )
```

Function called x amount per second based on timer initialized in timer_driver, drives the pid regulator.

4.54 GccApplication1/printf-stdarg.c File Reference

```
#include <stdarg.h>
#include "uart.h"
```

Include dependency graph for printf-stdarg.c:



Macros

- #define PAD_RIGHT 1
- #define PAD_ZERO 2
- #define PRINT_BUF_LEN 12

Functions

- int printf (const char *format,...)
- int sprintf (char *out, const char *format,...)
- int snprintf (char *buf, unsigned int count, const char *format,...)

4.54.1 Macro Definition Documentation

4.54.1.1 PAD_RIGHT

```
#define PAD_RIGHT 1
```

4.54.1.2 PAD_ZERO

```
#define PAD_ZERO 2
```

4.54.1.3 PRINT_BUF_LEN

```
#define PRINT_BUF_LEN 12
```

4.54.2 Function Documentation

4.54.2.1 printf()

```
int printf (
            const char * format,
             ...  )
```

4.54.2.2 snprintf()

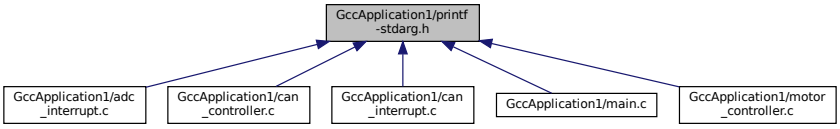
```
int snprintf (
            char * buf,
            unsigned int count,
            const char * format,
             ...  )
```

4.54.2.3 sprintf()

```
int sprintf (
            char * out,
            const char * format,
             ...  )
```

4.55 GccApplication1/printf-stdarg.h File Reference

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



Macros

- #define PRINTF -STDARG_H_

Functions

- int printf (const char ∗format,...)

4.55.1 Macro Definition Documentation

4.55.1.1 PRINTF

```
#define PRINTF -STDARG_H_
```

4.55.2 Function Documentation

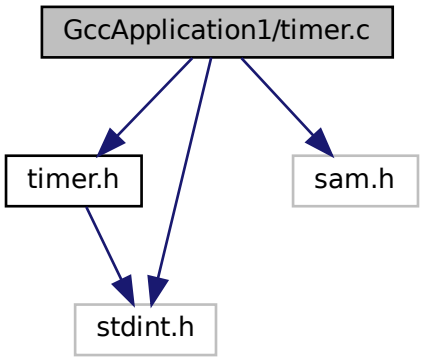
4.55.2.1 printf()

```
int printf (
            const char * format,
             ...  )
```

4.56 GccApplication1/timer.c File Reference

```
#include "timer.h"
#include <stdint.h>
#include "sam.h"
```

Include dependency graph for timer.c:



Functions

- void SysTick_init ()

  Function to set the systick registers on the atsam. Allows us to generate an interrupt at a set number of clock cycles.
- void SysTick_Handler (void)

  Function that is called by the systick interrupt is ideally called every millisecond and therefore increments a counter.
- uint32_t return_milliseconds ()

  Function to return milliseconds since program start.
- uint16_t return_seconds ()

  Function to return seconds since program start.
- uint16_t return_starttime ()

  Function to return a starttime defined by a CAN interrupt.
- void set_starttime ()

  Function to set starttime to the current number of seconds gone by.
- uint32_t return_trigger_time ()
- void set_trigger_time ()

Variables

- uint32_t trigger_time = 0

4.56.1 Function Documentation

4.56.1.1 return_milliseconds()

```
uint32_t return_milliseconds ( )
```

Function to return milliseconds since program start.

Returns

milliseconds gone by

4.56.1.2 return_seconds()

```
uint16_t return_seconds ( )
```

Function to return seconds since program start.

Returns

seconds gone by

4.56.1.3 return_starttime()

```
uint16_t return_starttime ( )
```

Function to return a starttime defined by a CAN interrupt.

Returns

the last set starttime

4.56.1.4 return_trigger_time()

```
uint32_t return_trigger_time ( )
```

4.56.1.5 set_starttime()

```
void set_starttime ( )
```

Function to set starttime to the current number of seconds gone by.

4.56.1.6 set_trigger_time()

```
void set_trigger_time ( )
```

4.56.1.7 SysTick_Handler()

```
void SysTick_Handler (
             void  )
```

Function that is called by the systick interrupt is ideally called every millisecond and therefore increments a counter.

4.56.1.8 SysTick_init()

```
void SysTick_init ( )
```

Function to set the systick registers on the atsam. Allows us to generate an interrupt at a set number of clock cycles.

4.56.2 Variable Documentation

4.56.2.1 trigger_time

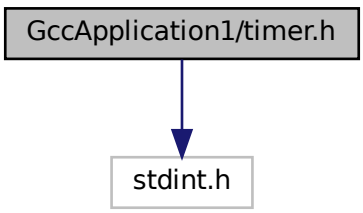
```
uint32_t trigger_time = 0
```

4.57 GccApplication1/timer.h File Reference

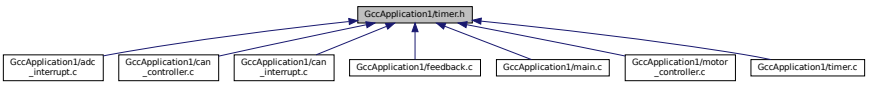
Driver module for asynchronous timer functionality.

```
#include <stdint.h>
```

Include dependency graph for timer.h:



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



Functions

- void SysTick_init ()

  Function to set the systick registers on the atsam. Allows us to generate an interrupt at a set number of clock cycles.
- void SysTick_Handler (void)

  Function that is called by the systick interrupt is ideally called every millisecond and therefore increments a counter.
- uint32_t return_milliseconds ()

  Function to return milliseconds since program start.
- uint16_t return_seconds ()

  Function to return seconds since program start.
- uint16_t return_starttime ()

  Function to return a starttime defined by a CAN interrupt.
- void set_starttime ()

  Function to set starttime to the current number of seconds gone by.
- uint32_t return_trigger_time ()
- void set_trigger_time ()

4.57.1 Detailed Description

Driver module for asynchronous timer functionality.

4.57.2 Function Documentation

4.57.2.1 return_milliseconds()

```
uint32_t return_milliseconds ( )
```

Function to return milliseconds since program start.

Returns

milliseconds gone by

4.57.2.2 return_seconds()

```
uint16_t return_seconds ( )
```

Function to return seconds since program start.

Returns

seconds gone by

4.57.2.3 return_starttime()

```
uint16_t return_starttime ( )
```

Function to return a starttime defined by a CAN interrupt.

Returns

the last set starttime

4.57.2.4 return_trigger_time()

```
uint32_t return_trigger_time ( )
```

4.57.2.5 set_starttime()

```
void set_starttime ( )
```

Function to set starttime to the current number of seconds gone by.

4.57.2.6 set_trigger_time()

```
void set_trigger_time ( )
```

4.57.2.7 SysTick_Handler()

```
void SysTick_Handler (
             void  )
```

Function that is called by the systick interrupt is ideally called every millisecond and therefore increments a counter.

4.57.2.8 SysTick_init()

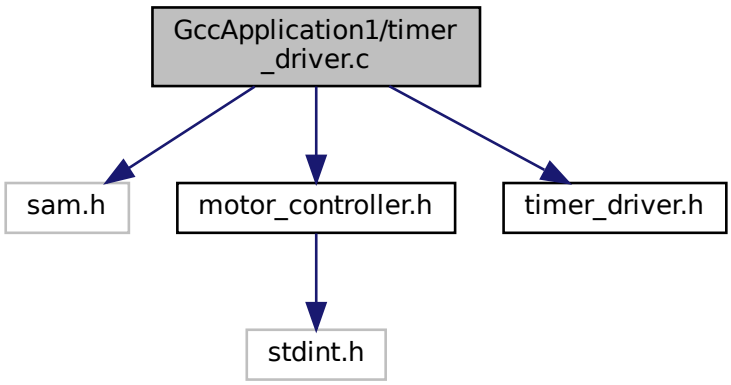
```
void SysTick_init ( )
```

Function to set the systick registers on the atsam. Allows us to generate an interrupt at a set number of clock cycles.

4.58 GccApplication1/timer_driver.c File Reference

```
#include "sam.h"
#include "motor_controller.h"
#include "timer_driver.h"
```

Include dependency graph for timer_driver.c:



Macros

- #define DEBUG_INTERRUPT 1

Functions

- uint8_t get_controller_runs ()
- void increment_controller_runs ()
- void reset_controller_runs ()
- void timer_init ()

  Function for setting up a timed pwm signal for the servo.
- void timer_change_duty (uint8_t dutyCycle)

  Function for changing the duty cycle of the servo pwm signal.
- void timer_change_duty_buzzer (uint8_t dutyCycle)
- void TC2_Handler (void)
- void init_ch1_PI ()
- void init_ch2 ()

Variables

- uint8_t ti_counter = 0

4.58.1 Macro Definition Documentation

4.58.1.1 DEBUG_INTERRUPT

```
#define DEBUG_INTERRUPT 1
```

4.58.2 Function Documentation

4.58.2.1 get_controller_runs()

```
uint8_t get_controller_runs ( )
```

4.58.2.2 increment_controller_runs()

```
void increment_controller_runs ( )
```

4.58.2.3 init_ch1_PI()

```
void init_ch1_PI ( )
```

4.58.2.4 init_ch2()

```
void init_ch2 ( )
```

4.58.2.5 reset_controller_runs()

```
void reset_controller_runs ( )
```

4.58.2.6 TC2_Handler()

```
void TC2_Handler (
             void  )
```

4.58.2.7 timer_change_duty()

```
void timer_change_duty (
             uint8_t dutyCycle )
```

Function for changing the duty cycle of the servo pwm signal.

Parameters

dutyCycle	new duty cycle

4.58.2.8 timer_change_duty_buzzer()

```
void timer_change_duty_buzzer (
             uint8_t dutyCycle )
```

4.58.2.9 timer_init()

```
void timer_init ( )
```

Function for setting up a timed pwm signal for the servo.

4.58.3 Variable Documentation

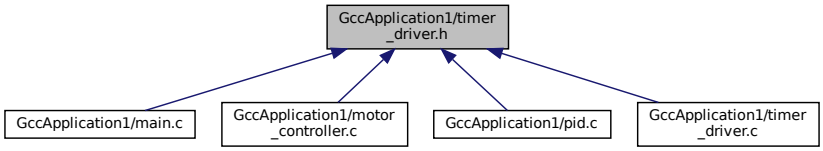
4.58.3.1 ti_counter

```
uint8_t ti_counter = 0
```

4.59 GccApplication1/timer_driver.h File Reference

Driver module for setting up and handling pwm timers and timer interrupt.

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



Functions

- void timer_init ()

  Function for setting up a timed pwm signal for the servo.
- void timer_change_duty (uint8_t dutyCycle)

  Function for changing the duty cycle of the servo pwm signal.
- void timer_change_duty_buzzer (uint8_t dutyCycle)
- void init_ch1_PI ()
- uint8_t get_controller_runs ()
- void reset_controller_runs ()
- void increment_controller_runs ()

4.59.1 Detailed Description

Driver module for setting up and handling pwm timers and timer interrupt.

Driver module for setting pwm timer data.

4.59.2 Function Documentation

4.59.2.1 get_controller_runs()

```
uint8_t get_controller_runs ( )
```

4.59.2.2 increment_controller_runs()

```
void increment_controller_runs ( )
```

4.59.2.3 init_ch1_PI()

```
void init_ch1_PI ( )
```

4.59.2.4 reset_controller_runs()

```
void reset_controller_runs ( )
```

4.59.2.5 timer_change_duty()

```
void timer_change_duty (
            uint8_t dutyCycle )
```

Function for changing the duty cycle of the servo pwm signal.

Parameters

dutyCycle	new duty cycle

4.59.2.6 timer_change_duty_buzzer()

```
void timer_change_duty_buzzer (
            uint8_t dutyCycle )
```

4.59.2.7 timer_init()

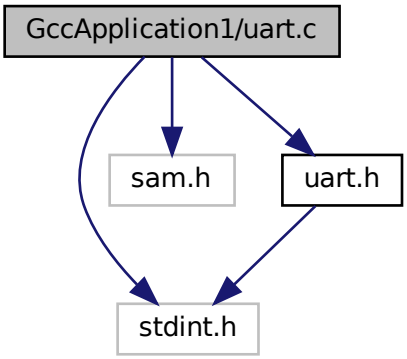
```
void timer_init ( )
```

Function for setting up a timed pwm signal for the servo.

4.60 GccApplication1/uart.c File Reference

```
#include <stdint.h>
#include "sam.h"
#include "uart.h"
```

Include dependency graph for uart.c:



Functions

- void configure_uart (void)

  Configure UART.
- int uart_getchar (uint8_t *c)

  Get character from UART.
- int uart_putchar (const uint8_t c)
- void UART_Handler (void)

  Handler for UART interrupts.

Variables

- uart_ringbuffer rx_buffer

4.60.1 Function Documentation

4.60.1.1 configure_uart()

```
void configure_uart (
             void  )
```

Configure UART.

Parameters

void	

Return values

void.	

4.60.1.2 uart_getchar()

```
int uart_getchar (
             uint8_t * c )
```

Get character from UART.

Parameters

*c	location of character

Return values

Success(0)	or failure(1)

4.60.1.3 UART_Handler()

```
void UART_Handler (
             void  )
```

Handler for UART interrupts.

Parameters

void	

Return values

void.	

4.60.1.4 uart_putchar()

```
int uart_putchar (
```

```
            const uint8_t c )
```

4.60.2 Variable Documentation

4.60.2.1 rx_buffer

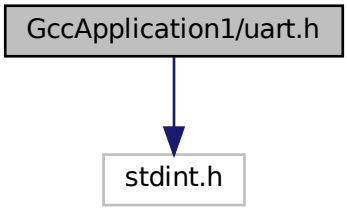
```
uart_ringbuffer rx_buffer
```

4.61 GccApplication1/uart.h File Reference

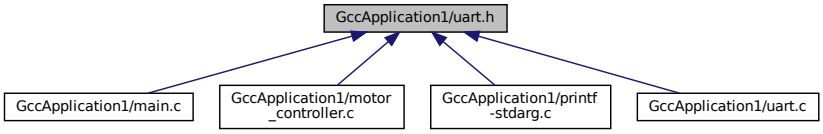
A simple interface for receiving and transmitting characters to a computer using UART via the on board USB-connector.

```
#include <stdint.h>
```

Include dependency graph for uart.h:



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



Classes

- struct uart_ringbuffer_t

Macros

- #define UART_RINGBUFFER_SIZE 64

Typedefs

- typedef struct uart_ringbuffer_t uart_ringbuffer

Functions

- void configure_uart (void)
  Configure UART.
- int uart_getchar (uint8_t *c)
  Get character from UART.
- int uart_putchar (const uint8_t c)
- void UART_Handler (void)
  Handler for UART interrupts.

4.61.1 Detailed Description

A simple interface for receiving and transmitting characters to a computer using UART via the on board USB-connector.

4.61.2 Macro Definition Documentation

4.61.2.1 UART_RINGBUFFER_SIZE

```
#define UART_RINGBUFFER_SIZE 64
```

4.61.3 Typedef Documentation

4.61.3.1 uart_ringbuffer

```
typedef struct uart_ringbuffer_t uart_ringbuffer
```

4.61.4 Function Documentation

4.61.4.1 configure_uart()

```
void configure_uart (
            void  )
```

Configure UART.

Parameters

void	

Return values

void.	

4.61.4.2 uart_getchar()

```
int uart_getchar (
            uint8_t * c )
```

Get character from UART.

Parameters

*c	location of character

Return values

Success(0)	or failure(1)

4.61.4.3 UART_Handler()

```
void UART_Handler (
            void  )
```

Handler for UART interrupts.

Parameters

void	

Return values

void.	

4.61.4.4 uart_putchar()

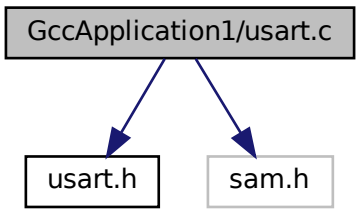
```
int uart_putchar (
```

```
            const uint8_t c )
```

4.62 GccApplication1/usart.c File Reference

```
#include "usart.h"
#include "sam.h"
```

Include dependency graph for usart.c:



Functions

- void usart_init ()

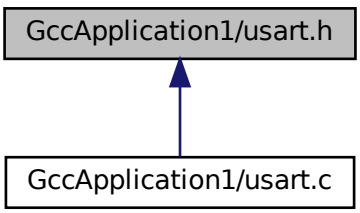
4.62.1 Function Documentation

4.62.1.1 usart_init()

```
void usart_init ( )
```

4.63 GccApplication1/usart.h File Reference

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



Functions

- void usart_init ()

4.63.1 Function Documentation

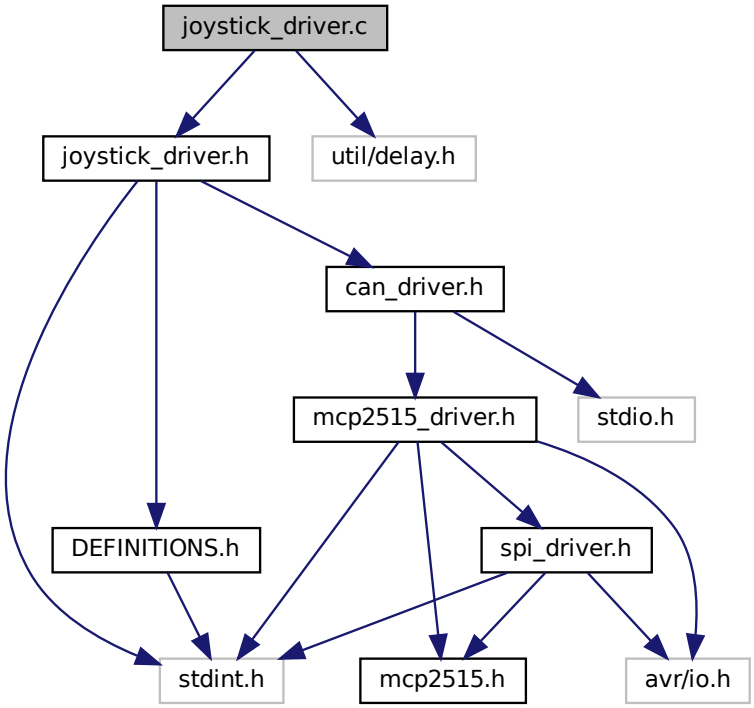
4.63.1.1 usart_init()

```
void usart_init ( )
```

4.64 joystick_driver.c File Reference

```
#include "joystick_driver.h"
#include <util/delay.h>
```

Include dependency graph for joystick_driver.c:



Functions

- joyVal get_joyvals ()

  Function to return joystick values for other modules.
- sliderVal get_slidervals ()

  Function to return slider values for other modules.
- void calc_offset ()

  Function that sets the x and y offsets to the joysticks current value This needs the joystick to be in a neutral position in order to work.
- uint8_t button_check (uint8_t current)

  Check if the button is pressed.
- void update_adc_values ()

  Updates all joystick and slider values from the adc Sets values to 0-100 or (-100)-100 for the joystick.
- DIRECTION joystick_direction (DIRECTION dir)

  Update a joysticks direction based on its x and y data.

Variables

- uint8_t x_offset = 160
- uint8_t y_offset = 160
- uint8_t previous = 1
- DIRECTION joydir = NEUTRAL
- DIRECTION direction

4.64.1 Function Documentation

4.64.1.1 button_check()

```
uint8_t button_check (
            uint8_t current )
```

Check if the button is pressed.

Function to check if one of the three buttons is being pressed. Uses a local previous variable to make sure an input is registered as just one input.

Parameters

current	Current value of the button

Returns

1 if pressed, else 0

4.64.1.2 calc_offset()

```
void calc_offset ( )
```

Function that sets the x and y offsets to the joysticks current value This needs the joystick to be in a neutral position in order to work.

4.64.1.3 get_joyvals()

```
joyVal get_joyvals ( )
```

Function to return joystick values for other modules.

Returns

joystick variable

4.64.1.4 get_slidervals()

```
sliderVal get_slidervals ( )
```

Function to return slider values for other modules.

Returns

slider variable

4.64.1.5 joystick_direction()

```
DIRECTION joystick_direction (
            DIRECTION dir )
```

Update a joysticks direction based on its x and y data.

Parameters

dir	previous direction of the joystick uses local joystick variable

Returns

the joysticks DIRECTION

4.64.1.6 update_adc_values()

```
void update_adc_values ( )
```

Updates all joystick and slider values from the adc Sets values to 0-100 or (-100)-100 for the joystick.

4.64.2 Variable Documentation

4.64.2.1 direction

```
DIRECTION direction
```

4.64.2.2 joydir

```
DIRECTION joydir = NEUTRAL
```

4.64.2.3 previous

```
uint8_t previous = 1
```

4.64.2.4 x_offset

```
uint8_t x_offset = 160
```

4.64.2.5 y_offset

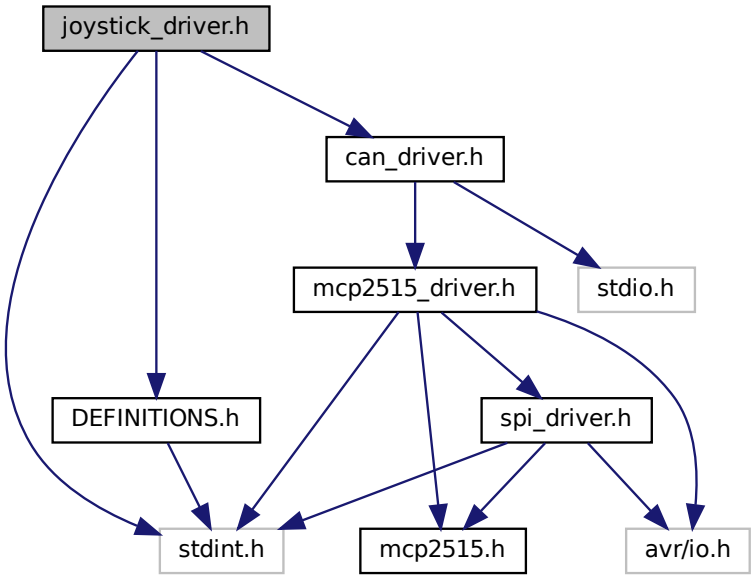
```
uint8_t y_offset = 160
```

4.65 joystick_driver.h File Reference

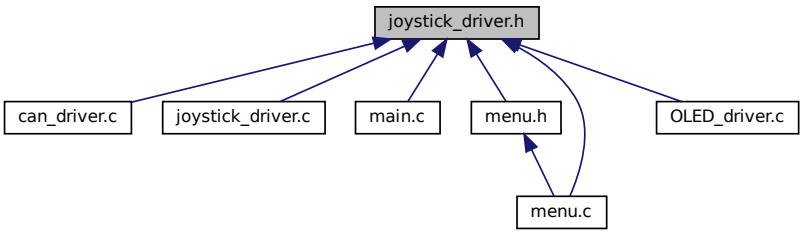
Driver module for handling data joystick and slider inputs.

```
#include <stdint.h>
#include "DEFINITIONS.h"
#include "can_driver.h"
```

Include dependency graph for joystick_driver.h:



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



Classes

- struct joyVal

  Send amount of time node 2 has been active.
- struct sliderVal

  Struct to store left and right slider values.

Enumerations

- enum DIRECTION {
  LEFT, RIGHT, UP, DOWN,
  NEUTRAL, WAITING }

  Struct for joystick directions.

Functions

- joyVal get_joyvals ()

  Function to return joystick values for other modules.
- sliderVal get_slidervals ()

  Function to return slider values for other modules.
- void calc_offset ()

  Function that sets the x and y offsets to the joysticks current value This needs the joystick to be in a neutral position in order to work.
- uint8_t button_check (uint8_t current)

  Function to check if one of the three buttons is being pressed. Uses a local previous variable to make sure an input is registered as just one input.
- void update_adc_values ()

  Updates all joystick and slider values from the adc Sets values to 0-100 or (-100)-100 for the joystick.
- DIRECTION joystick_direction (DIRECTION dir)

  Update a joysticks direction based on its x and y data.

4.65.1 Detailed Description

Driver module for handling data joystick and slider inputs.

4.65.2 Enumeration Type Documentation

4.65.2.1 DIRECTION

```
enum DIRECTION
```

Struct for joystick directions.

Enumerator

LEFT	
RIGHT	
UP	
DOWN	
NEUTRAL	
WAITING	

4.65.3 Function Documentation

4.65.3.1 button_check()

```
uint8_t button_check (
            uint8_t current )
```

Function to check if one of the three buttons is being pressed. Uses a local previous variable to make sure an input is registered as just one input.

Parameters

current	can be any button value represented by a boolean

Returns

1 if button is being pressed, 0 otherwise

Function to check if one of the three buttons is being pressed. Uses a local previous variable to make sure an input is registered as just one input.

Parameters

current	Current value of the button

Returns

1 if pressed, else 0

4.65.3.2 calc_offset()

```
void calc_offset ( )
```

Function that sets the x and y offsets to the joysticks current value This needs the joystick to be in a neutral position in order to work.

4.65.3.3 get_joyvals()

```
joyVal get_joyvals ( )
```

Function to return joystick values for other modules.

Returns

joystick variable

4.65.3.4 get_slidervals()

```
sliderVal get_slidervals ( )
```

Function to return slider values for other modules.

Returns

slider variable

4.65.3.5 joystick_direction()

```
DIRECTION joystick_direction (
            DIRECTION dir )
```

Update a joysticks direction based on its x and y data.

Parameters

dir	previous direction of the joystick uses local joystick variable

Returns

the joysticks DIRECTION

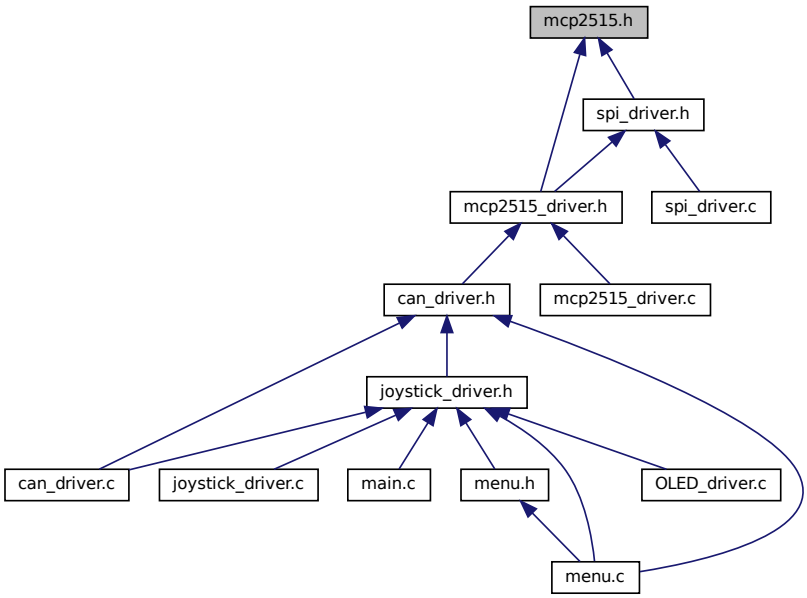
4.65.3.6 update_adc_values()

```
void update_adc_values ( )
```

Updates all joystick and slider values from the adc Sets values to 0-100 or (-100)-100 for the joystick.

4.66 mcp2515.h File Reference

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



Macros

- #define MCP_RXF0SIDH 0x00
- #define MCP_RXF0SIDL 0x01
- #define MCP_RXF0EID8 0x02
- #define MCP_RXF0EID0 0x03
- #define MCP_RXF1SIDH 0x04
- #define MCP_RXF1SIDL 0x05
- #define MCP_RXF1EID8 0x06
- #define MCP_RXF1EID0 0x07
- #define MCP_RXF2SIDH 0x08
- #define MCP_RXF2SIDL 0x09
- #define MCP_RXF2EID8 0x0A
- #define MCP_RXF2EID0 0x0B
- #define MCP_CANSTAT 0x0E
- #define MCP_CANCTRL 0x0F
- #define MCP_RXF3SIDH 0x10
- #define MCP_RXF3SIDL 0x11
- #define MCP_RXF3EID8 0x12
- #define MCP_RXF3EID0 0x13
- #define MCP_RXF4SIDH 0x14
- #define MCP_RXF4SIDL 0x15
- #define MCP_RXF4EID8 0x16
- #define MCP_RXF4EID0 0x17
- #define MCP_RXF5SIDH 0x18
- #define MCP_RXF5SIDL 0x19

- #define MCP_RXF5EID8 0x1A
- #define MCP_RXF5EID0 0x1B
- #define MCP_TEC 0x1C
- #define MCP_REC 0x1D
- #define MCP_RXM0SIDH 0x20
- #define MCP_RXM0SIDL 0x21
- #define MCP_RXM0EID8 0x22
- #define MCP_RXM0EID0 0x23
- #define MCP_RXM1SIDH 0x24
- #define MCP_RXM1SIDL 0x25
- #define MCP_RXM1EID8 0x26
- #define MCP_RXM1EID0 0x27
- #define MCP_CNF3 0x28
- #define MCP_CNF2 0x29
- #define MCP_CNF1 0x2A
- #define MCP_CANINTE 0x2B
- #define MCP_CANINTF 0x2C
- #define MCP_EFLG 0x2D
- #define MCP_TXB0CTRL 0x30
- #define MCP_TXB1CTRL 0x40
- #define MCP_TXB2CTRL 0x50
- #define MCP_RXB0CTRL 0x60
- #define MCP_RXB0SIDH 0x61
- #define MCP_RXB1CTRL 0x70
- #define MCP_RXB1SIDH 0x71
- #define MCP_TX_INT 0x1C
- #define MCP_TX01_INT 0x0C
- #define MCP_RX_INT 0x03
- #define MCP_NO_INT 0x00
- #define MCP_TX01_MASK 0x14
- #define MCP_TX_MASK 0x54
- #define MCP_WRITE 0x02
- #define MCP_READ 0x03
- #define MCP_BITMOD 0x05
- #define MCP_LOAD_TX0 0x40
- #define MCP_LOAD_TX1 0x42
- #define MCP_LOAD_TX2 0x44
- #define MCP_RTS_TX0 0x81
- #define MCP_RTS_TX1 0x82
- #define MCP_RTS_TX2 0x84
- #define MCP_RTS_ALL 0x87
- #define MCP_READ_RX0 0x90
- #define MCP_READ_RX1 0x94
- #define MCP_READ_STATUS 0xA0
- #define MCP_RX_STATUS 0xB0
- #define MCP_RESET 0xC0
- #define MODE_NORMAL 0x00
- #define MODE_SLEEP 0x20
- #define MODE_LOOPBACK 0x40
- #define MODE_LISTENONLY 0x60
- #define MODE_CONFIG 0x80
- #define MODE_POWERUP 0xE0
- #define MODE_MASK 0xE0
- #define ABORT_TX 0x10
- #define MODE_ONESHOT 0x08

- #define CLKOUT_ENABLE 0x04
- #define CLKOUT_DISABLE 0x00
- #define CLKOUT_PS1 0x00
- #define CLKOUT_PS2 0x01
- #define CLKOUT_PS4 0x02
- #define CLKOUT_PS8 0x03
- #define SJW1 0x00
- #define SJW2 0x40
- #define SJW3 0x80
- #define SJW4 0xC0
- #define BTLMODE 0x80
- #define SAMPLE_1X 0x00
- #define SAMPLE_3X 0x40
- #define SOF_ENABLE 0x80
- #define SOF_DISABLE 0x00
- #define WAKFIL_ENABLE 0x40
- #define WAKFIL_DISABLE 0x00
- #define MCP_RX0IF 0x01
- #define MCP_RX1IF 0x02
- #define MCP_TX0IF 0x04
- #define MCP_TX1IF 0x08
- #define MCP_TX2IF 0x10
- #define MCP_ERRIF 0x20
- #define MCP_WAKIF 0x40
- #define MCP_MERRF 0x80

4.66.1 Macro Definition Documentation

4.66.1.1 ABORT_TX

```
#define ABORT_TX 0x10
```

4.66.1.2 BTLMODE

```
#define BTLMODE 0x80
```

4.66.1.3 CLKOUT_DISABLE

```
#define CLKOUT_DISABLE 0x00
```

4.66.1.4 CLKOUT_ENABLE

```
#define CLKOUT_ENABLE 0x04
```

4.66.1.5 CLKOUT_PS1

```
#define CLKOUT_PS1 0x00
```

4.66.1.6 CLKOUT_PS2

```
#define CLKOUT_PS2 0x01
```

4.66.1.7 CLKOUT_PS4

```
#define CLKOUT_PS4 0x02
```

4.66.1.8 CLKOUT_PS8

```
#define CLKOUT_PS8 0x03
```

4.66.1.9 MCP_BITMOD

```
#define MCP_BITMOD 0x05
```

4.66.1.10 MCP_CANCTRL

```
#define MCP_CANCTRL 0x0F
```

4.66.1.11 MCP_CANINTE

```
#define MCP_CANINTE 0x2B
```

4.66.1.12 MCP_CANINTF

```
#define MCP_CANINTF 0x2C
```

4.66.1.13 MCP_CANSTAT

```
#define MCP_CANSTAT 0x0E
```

4.66.1.14 MCP_CNF1

```
#define MCP_CNF1 0x2A
```

4.66.1.15 MCP_CNF2

```
#define MCP_CNF2 0x29
```

4.66.1.16 MCP_CNF3

```
#define MCP_CNF3 0x28
```

4.66.1.17 MCP_EFLG

```
#define MCP_EFLG 0x2D
```

4.66.1.18 MCP_ERRIF

```
#define MCP_ERRIF 0x20
```

4.66.1.19 MCP_LOAD_TX0

```
#define MCP_LOAD_TX0 0x40
```

4.66.1.20 MCP_LOAD_TX1

```
#define MCP_LOAD_TX1 0x42
```

4.66.1.21 MCP_LOAD_TX2

```
#define MCP_LOAD_TX2 0x44
```

4.66.1.22 MCP_MERRF

```
#define MCP_MERRF 0x80
```

4.66.1.23 MCP_NO_INT

```
#define MCP_NO_INT 0x00
```

4.66.1.24 MCP_READ

```
#define MCP_READ 0x03
```

4.66.1.25 MCP_READ_RX0

```
#define MCP_READ_RX0 0x90
```

4.66.1.26 MCP_READ_RX1

```
#define MCP_READ_RX1 0x94
```

4.66.1.27 MCP_READ_STATUS

```
#define MCP_READ_STATUS 0xA0
```

4.66.1.28 MCP_REC

```
#define MCP_REC 0x1D
```

4.66.1.29 MCP_RESET

```
#define MCP_RESET 0xC0
```

4.66.1.30 MCP_RTS_ALL

```
#define MCP_RTS_ALL 0x87
```

4.66.1.31 MCP_RTS_TX0

```
#define MCP_RTS_TX0 0x81
```

4.66.1.32 MCP_RTS_TX1

```
#define MCP_RTS_TX1 0x82
```

4.66.1.33 MCP_RTS_TX2

```
#define MCP_RTS_TX2 0x84
```

4.66.1.34 MCP_RX0IF

```
#define MCP_RX0IF 0x01
```

4.66.1.35 MCP_RX1IF

```
#define MCP_RX1IF 0x02
```

4.66.1.36 MCP_RX_INT

```
#define MCP_RX_INT 0x03
```

4.66.1.37 MCP_RX_STATUS

```
#define MCP_RX_STATUS 0xB0
```

4.66.1.38 MCP_RXB0CTRL

```
#define MCP_RXB0CTRL 0x60
```

4.66.1.39 MCP_RXB0SIDH

```
#define MCP_RXB0SIDH 0x61
```

4.66.1.40 MCP_RXB1CTRL

```
#define MCP_RXB1CTRL 0x70
```

4.66.1.41 MCP_RXB1SIDH

```
#define MCP_RXB1SIDH 0x71
```

4.66.1.42 MCP_RXF0EID0

```
#define MCP_RXF0EID0 0x03
```

4.66.1.43 MCP_RXF0EID8

```
#define MCP_RXF0EID8 0x02
```

4.66.1.44 MCP_RXF0SIDH

```
#define MCP_RXF0SIDH 0x00
```

4.66.1.45 MCP_RXF0SIDL

```
#define MCP_RXF0SIDL 0x01
```

4.66.1.46 MCP_RXF1EID0

```
#define MCP_RXF1EID0 0x07
```

4.66.1.47 MCP_RXF1EID8

```
#define MCP_RXF1EID8 0x06
```

4.66.1.48 MCP_RXF1SIDH

```
#define MCP_RXF1SIDH 0x04
```

4.66.1.49 MCP_RXF1SIDL

```
#define MCP_RXF1SIDL 0x05
```

4.66.1.50 MCP_RXF2EID0

```
#define MCP_RXF2EID0 0x0B
```

4.66.1.51 MCP_RXF2EID8

```
#define MCP_RXF2EID8 0x0A
```

4.66.1.52 MCP_RXF2SIDH

```
#define MCP_RXF2SIDH 0x08
```

4.66.1.53 MCP_RXF2SIDL

```
#define MCP_RXF2SIDL 0x09
```

4.66.1.54 MCP_RXF3EID0

```
#define MCP_RXF3EID0 0x13
```

4.66.1.55 MCP_RXF3EID8

```
#define MCP_RXF3EID8 0x12
```

4.66.1.56 MCP_RXF3SIDH

```
#define MCP_RXF3SIDH 0x10
```

4.66.1.57 MCP_RXF3SIDL

```
#define MCP_RXF3SIDL 0x11
```

4.66.1.58 MCP_RXF4EID0

```
#define MCP_RXF4EID0 0x17
```

4.66.1.59 MCP_RXF4EID8

```
#define MCP_RXF4EID8 0x16
```

4.66.1.60 MCP_RXF4SIDH

```
#define MCP_RXF4SIDH 0x14
```

4.66.1.61 MCP_RXF4SIDL

```
#define MCP_RXF4SIDL 0x15
```

4.66.1.62 MCP_RXF5EID0

```
#define MCP_RXF5EID0 0x1B
```

4.66.1.63 MCP_RXF5EID8

```
#define MCP_RXF5EID8 0x1A
```

4.66.1.64 MCP_RXF5SIDH

```
#define MCP_RXF5SIDH 0x18
```

4.66.1.65 MCP_RXF5SIDL

```
#define MCP_RXF5SIDL 0x19
```

4.66.1.66 MCP_RXM0EID0

```
#define MCP_RXM0EID0 0x23
```

4.66.1.67 MCP_RXM0EID8

```
#define MCP_RXM0EID8 0x22
```

4.66.1.68 MCP_RXM0SIDH

```
#define MCP_RXM0SIDH 0x20
```

4.66.1.69 MCP_RXM0SIDL

```
#define MCP_RXM0SIDL 0x21
```

4.66.1.70 MCP_RXM1EID0

```
#define MCP_RXM1EID0 0x27
```

4.66.1.71 MCP_RXM1EID8

```
#define MCP_RXM1EID8 0x26
```

4.66.1.72 MCP_RXM1SIDH

```
#define MCP_RXM1SIDH 0x24
```

4.66.1.73 MCP_RXM1SIDL

```
#define MCP_RXM1SIDL 0x25
```

4.66.1.74 MCP_TEC

```
#define MCP_TEC 0x1C
```

4.66.1.75 MCP_TX01_INT

```
#define MCP_TX01_INT 0x0C
```

4.66.1.76 MCP_TX01_MASK

```
#define MCP_TX01_MASK 0x14
```

4.66.1.77 MCP_TX0IF

```
#define MCP_TX0IF 0x04
```

4.66.1.78 MCP_TX1IF

```
#define MCP_TX1IF 0x08
```

4.66.1.79 MCP_TX2IF

```
#define MCP_TX2IF 0x10
```

4.66.1.80 MCP_TX_INT

```
#define MCP_TX_INT 0x1C
```

4.66.1.81 MCP_TX_MASK

```
#define MCP_TX_MASK 0x54
```

4.66.1.82 MCP_TXB0CTRL

```
#define MCP_TXB0CTRL 0x30
```

4.66.1.83 MCP_TXB1CTRL

```
#define MCP_TXB1CTRL 0x40
```

4.66.1.84 MCP_TXB2CTRL

```
#define MCP_TXB2CTRL 0x50
```

4.66.1.85 MCP_WAKIF

```
#define MCP_WAKIF 0x40
```

4.66.1.86 MCP_WRITE

```
#define MCP_WRITE 0x02
```

4.66.1.87 MODE_CONFIG

```
#define MODE_CONFIG 0x80
```

4.66.1.88 MODE_LISTENONLY

```
#define MODE_LISTENONLY 0x60
```

4.66.1.89 MODE_LOOPBACK

```
#define MODE_LOOPBACK 0x40
```

4.66.1.90 MODE_MASK

```
#define MODE_MASK 0xE0
```

4.66.1.91 MODE_NORMAL

```
#define MODE_NORMAL 0x00
```

4.66.1.92 MODE_ONESHOT

```
#define MODE_ONESHOT 0x08
```

4.66.1.93 MODE_POWERUP

```
#define MODE_POWERUP 0xE0
```

4.66.1.94 MODE_SLEEP

```
#define MODE_SLEEP 0x20
```

4.66.1.95 SAMPLE_1X

```
#define SAMPLE_1X 0x00
```

4.66.1.96 SAMPLE_3X

```
#define SAMPLE_3X 0x40
```

4.66.1.97 SJW1

```
#define SJW1 0x00
```

4.66.1.98 SJW2

```
#define SJW2 0x40
```

4.66.1.99 SJW3

```
#define SJW3 0x80
```

4.66.1.100 SJW4

```
#define SJW4 0xC0
```

4.66.1.101 SOF_DISABLE

```
#define SOF_DISABLE 0x00
```

4.66.1.102 SOF_ENABLE

```
#define SOF_ENABLE 0x80
```

4.66.1.103 WAKFIL_DISABLE

```
#define WAKFIL_DISABLE 0x00
```

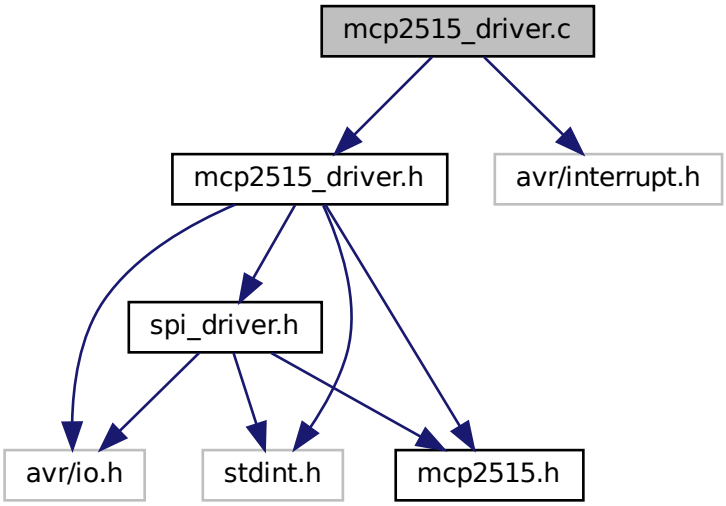
4.66.1.104 WAKFIL_ENABLE

```
#define WAKFIL_ENABLE 0x40
```

4.67 mcp2515_driver.c File Reference

```
#include "mcp2515_driver.h"
#include "avr/interrupt.h"
```

Include dependency graph for mcp2515_driver.c:



Functions

- void mcp2515_init ()

  Initializes spi and resets the mcp2515.
- uint8_t mcp2515_read (uint8_t address)

  Sets the mcp2515 in read mode and reads the selected address using spi.
- void mcp2515_write (uint8_t address, uint8_t data)

  Sets the mcp2515 in write mode and writes the selected address using spi.
- void mcp2515_request_to_send (uint8_t command)
- void mcp2515_bit_modify (uint8_t address, uint8_t mask, uint8_t data)

  Function to modify a bit at an address in the mcp2515.
- void mcp2515_reset ()

  Resets the mcp2515.
- uint8_t mcp2515_read_status ()

  Function to get the status of the mcp2515.

4.67.1 Function Documentation

4.67.1.1 mcp2515_bit_modify()

```
void mcp2515_bit_modify (
            uint8_t address,
            uint8_t mask,
            uint8_t data )
```

Function to modify a bit at an address in the mcp2515.

Parameters

address	address to read
mask	to choose which bit in the byte to change
data	to be written

4.67.1.2 mcp2515_init()

```
void mcp2515_init ( )
```

Initializes spi and resets the mcp2515.

4.67.1.3 mcp2515_read()

```
uint8_t mcp2515_read (
            uint8_t address )
```

Sets the mcp2515 in read mode and reads the selected address using spi.

Parameters

address	address to read

Returns

data at address

ss

4.67.1.4 mcp2515_read_status()

```
uint8_t mcp2515_read_status ( )
```

Function to get the status of the mcp2515.

Returns

MCP2515 status

4.67.1.5 mcp2515_request_to_send()

```
void mcp2515_request_to_send (
            uint8_t command )
```

Parameters

command	mcp2515 command to be sent according to datasheet

4.67.1.6 mcp2515_reset()

```
void mcp2515_reset ( )
```

Resets the mcp2515.

4.67.1.7 mcp2515_write()

```
void mcp2515_write (
            uint8_t address,
            uint8_t data )
```

Sets the mcp2515 in write mode and writes the selected address using spi.

Parameters

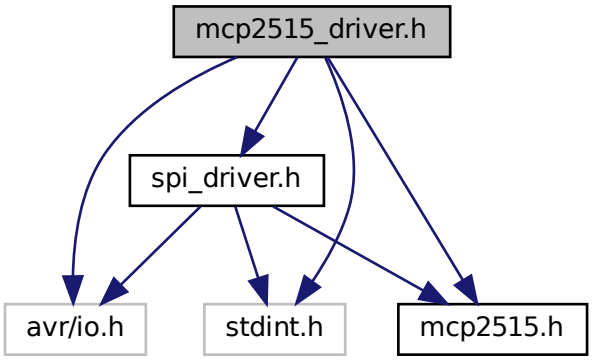
address	address to write
data	data to write

4.68 mcp2515_driver.h File Reference

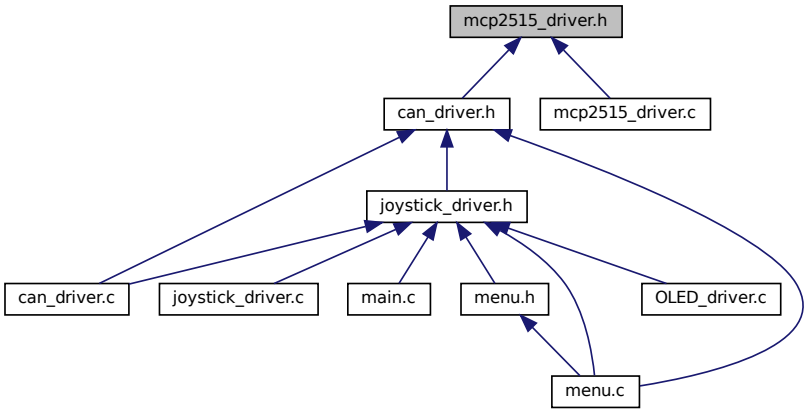
Driver module for for the mcp2515 CAN controller specifically.

```
#include <avr/io.h>
#include <stdint.h>
#include "mcp2515.h"
#include "spi_driver.h"
```

Include dependency graph for mcp2515_driver.h:



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



Classes

- struct can_message

  Struct for the contents of a CAN message.

Functions

- uint8_t mcp2515_read (uint8_t address)

  Sets the mcp2515 in read mode and reads the selected address using spi.
- void mcp2515_write (uint8_t address, uint8_t data)

  Sets the mcp2515 in write mode and writes the selected address using spi.
- void mcp2515_request_to_send (uint8_t command)
- void mcp2515_bit_modify (uint8_t address, uint8_t mask, uint8_t data)

  Function to modify a bit at an address in the mcp2515.
- void mcp2515_reset ()

  Resets the mcp2515.
- uint8_t mcp2515_read_status ()

  Function to get the status of the mcp2515.
- void mcp2515_init ()

  Initializes spi and resets the mcp2515.

4.68.1 Detailed Description

Driver module for for the mcp2515 CAN controller specifically.

4.68.2 Function Documentation

4.68.2.1 mcp2515_bit_modify()

```
void mcp2515_bit_modify (
            uint8_t address,
            uint8_t mask,
            uint8_t data )
```

Function to modify a bit at an address in the mcp2515.

Parameters

address	address to read
mask	to choose which bit in the byte to change
data	to be written

4.68.2.2 mcp2515_init()

```
void mcp2515_init ( )
```

Initializes spi and resets the mcp2515.

4.68.2.3 mcp2515_read()

```
uint8_t mcp2515_read (
            uint8_t address )
```

Sets the mcp2515 in read mode and reads the selected address using spi.

Parameters

address	address to read

Returns

data at address

ss

4.68.2.4 mcp2515_read_status()

```
uint8_t mcp2515_read_status ( )
```

Function to get the status of the mcp2515.

Returns

MCP2515 status

4.68.2.5 mcp2515_request_to_send()

```
void mcp2515_request_to_send (
            uint8_t command )
```

Parameters

command	mcp2515 command to be sent according to datasheet

4.68.2.6 mcp2515_reset()

```
void mcp2515_reset ( )
```

Resets the mcp2515.

4.68.2.7 mcp2515_write()

```
void mcp2515_write (
            uint8_t address,
            uint8_t data )
```

Sets the mcp2515 in write mode and writes the selected address using spi.

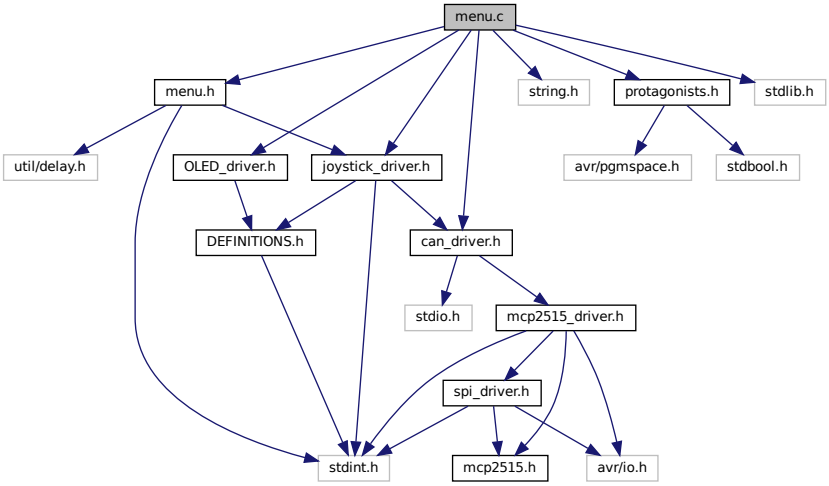
Parameters

address	address to write
data	data to write

4.69 menu.c File Reference

```
#include "menu.h"
#include "OLED_driver.h"
#include <string.h>
#include "protagonists.h"
#include "joystick_driver.h"
#include "can_driver.h"
#include <stdlib.h>
```

Include dependency graph for menu.c:



Functions

- menu * new_menu (menu *parent)

  Function to create a new menu page.
- void write_menu_to_screen (menu *menuPointer)

  Function to write the current menu to screen.
- void change_menu (menu *next_menu, menu **menuHead)

  Function to change the menu.
- void invert_selected (menu *menuPointer)

  Function to invert the selected row. Selected row is tracked by the "seleced" variable in the struct.
- void change_selected (menu **menuHead, DIRECTION d)

  Function to change which option in a menu is selected.
- void button_pressed (menu **menuHead)

  Function to check if the joystick button is pressed.
- void launch_menusystem ()

  Function to start our pretetermined menusystem.
- void wojakprinter ()

  Function that prints a familiar face to the OLED.
- void hello_world ()

  Function to write "hello world" to the OLED.
- void choose_character ()

  Function to let you choose between our two characters. The choice gets saved to SRAM and enables high-score tracking.
- void play_game ()

  Function to initialize ping-pong game uses CAN to send joystick info over to node 2 receives feedback over CAN and displays it on the OLED.
- void calibrate ()

  Function to calibrate joystick for offset. Gives user clear instructions.
- void set_easy ()

  Function to set PID regulator on node 2 to easy mode over CAN.
- void set_medium ()

  Function to set PID regulator on node 2 to medium mode over CAN.
- void set_hard ()

  Function to set PID regulator on node 2 to hard mode over CAN.
- void show_credits ()

  Function to credit the creators of the game on the OLED.
- void hiscore ()

  Function to show current high scores for both characters uses information from SRAM.
- void reset_scores ()

  Function to set high score values for both characters to 0 in SRAM.
- void reaction_test ()

  Function to launch a small minigame that lets you test your reaction time sends a start and a stop signal to node 2 which has the timer implemented. you lose if you press too early.

Variables

- volatile char * sram = (char *) 0x1800
- char * string_list [ ]

4.69.1 Function Documentation

4.69.1.1 button_pressed()

```
void button_pressed (
            menu ** menuHead )
```

Function to check if the joystick button is pressed.

Parameters

menuHead	pointer to current active menu

4.69.1.2 calibrate()

```
void calibrate ( )
```

Function to calibrate joystick for offset. Gives user clear instructions.

4.69.1.3 change_menu()

```
void change_menu (
            menu * next_menu,
            menu ** menuHead )
```

Function to change the menu.

Parameters

menuHead	is a pointer to our current placement in the linked list
next_menu	is a pointer to the menu we are navigating to

4.69.1.4 change_selected()

```
void change_selected (
            menu ** menuHead,
            DIRECTION d )
```

Function to change which option in a menu is selected.

Parameters

menuHead	pointer to current menu
d	joystick direction used to change the selected item according to user input

4.69.1.5 choose_character()

```
void choose_character ( )
```

Function to let you choose between our two characters. The choice gets saved to SRAM and enables high-score tracking.

4.69.1.6 hello_world()

```
void hello_world ( )
```

Function to write "hello world" to the OLED.

4.69.1.7 hiscore()

```
void hiscore ( )
```

Function to show current high scores for both characters uses information from SRAM.

4.69.1.8 invert_selected()

```
void invert_selected (
            menu * menuPointer )
```

Function to invert the selected row. Selected row is tracked by the "seleced" variable in the struct.

Parameters

menuPointer	pointer to current screen

4.69.1.9 launch_menusystem()

```
void launch_menusystem ( )
```

Function to start our pretetermined menusystem.

4.69.1.10 new_menu()

```
menu* new_menu (
            menu * parent )
```

Function to create a new menu page.

Warning

Uses Malloc to allocate space for the menu in memory

Parameters

parent	uses a pointer to its parent in order to create a "back" - option

4.69.1.11 play_game()

```
void play_game ( )
```

Function to initialize ping-pong game uses CAN to send joystick info over to node 2 receives feedback over CAN and displays it on the OLED.

4.69.1.12 reaction_test()

```
void reaction_test ( )
```

Function to launch a small minigame that lets you test your reaction time sends a start and a stop signal to node 2 which has the timer implemented. you lose if you press too early.

4.69.1.13 reset_scores()

```
void reset_scores ( )
```

Function to set high score values for both characters to 0 in SRAM.

4.69.1.14 set_easy()

```
void set_easy ( )
```

Function to set PID regulator on node 2 to easy mode over CAN.

4.69.1.15 set_hard()

```
void set_hard ( )
```

Function to set PID regulator on node 2 to hard mode over CAN.

4.69.1.16 set_medium()

```
void set_medium ( )
```

Function to set PID regulator on node 2 to medium mode over CAN.

4.69.1.17 show_credits()

```
void show_credits ( )
```

Function to credit the creators of the game on the OLED.

4.69.1.18 wojakprinter()

```
void wojakprinter ( )
```

Function that prints a familiar face to the OLED.

4.69.1.19 write_menu_to_screen()

```
void write_menu_to_screen (
            menu * menuPointer )
```

Function to write the current menu to screen.

Parameters

menuPointer	pointer to current screen

4.69.2 Variable Documentation

4.69.2.1 sram

```
volatile char* sram = (char *) 0x1800
```

4.69.2.2 string_list

```
char* string_list[]
```

Initial value:

```
= {
    "Option 1",
    "Option 2",
    "Option 3",
    "",
    "",
    "",
    "",
    "<- Back"
}
```

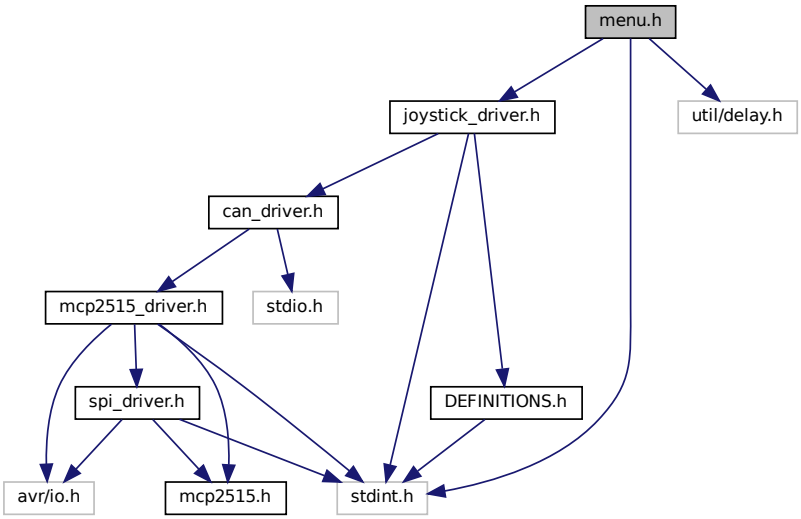
4.70 menu.h File Reference

Driver module for handling data transmission over the CAN-bus.

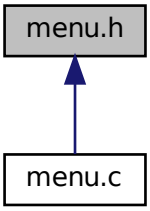
```
#include <stdint.h>
#include "joystick_driver.h"
```

```
#include <util/delay.h>
```

Include dependency graph for menu.h:



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



Classes

- struct menu

  Struct for content in a menu page labels are names of options, one for each line links are references to submenues (linked list) f are function pointers as an option to submenues selected keeps track of the selected option for each menu.

Functions

- menu ∗ new_menu (menu ∗parent)

  Function to create a new menu page.
- void write_menu_to_screen (menu ∗menuPointer)

Function to write the current menu to screen.

- void change_menu (menu *next_menu, menu **menuHead)

  Function to change the menu.
- void invert_selected (menu *menuPointer)

  Function to invert the selected row. Selected row is tracked by the "seleced" variable in the struct.
- void change_selected (menu **menuHead, DIRECTION d)

  Function to change which option in a menu is selected.
- void button_pressed (menu **menuHead)

  Function to check if the joystick button is pressed.
- void launch_menusystem ()

  Function to start our pretetermined menusystem.
- void wojakprinter ()

  Function that prints a familiar face to the OLED.
- void hello_world ()

  Function to write "hello world" to the OLED.
- void choose_character ()

  Function to let you choose between our two characters. The choice gets saved to SRAM and enables high-score tracking.
- void play_game ()

  Function to initialize ping-pong game uses CAN to send joystick info over to node 2 receives feedback over CAN and displays it on the OLED.
- void calibrate ()

  Function to calibrate joystick for offset. Gives user clear instructions.
- void set_easy ()

  Function to set PID regulator on node 2 to easy mode over CAN.
- void set_medium ()

  Function to set PID regulator on node 2 to medium mode over CAN.
- void set_hard ()

  Function to set PID regulator on node 2 to hard mode over CAN.
- void show_credits ()

  Function to credit the creators of the game on the OLED.
- void hiscore ()

  Function to show current high scores for both characters uses information from SRAM.
- void reset_scores ()

  Function to set high score values for both characters to 0 in SRAM.
- void reaction_test ()

  Function to launch a small minigame that lets you test your reaction time sends a start and a stop signal to node 2 which has the timer implemented. you lose if you press too early.

4.70.1 Detailed Description

Driver module for handling data transmission over the CAN-bus.

4.70.2 Function Documentation

4.70.2.1 button_pressed()

```
void button_pressed (
            menu ** menuHead )
```

Function to check if the joystick button is pressed.

Parameters

menuHead	pointer to current active menu

4.70.2.2 calibrate()

```
void calibrate ( )
```

Function to calibrate joystick for offset. Gives user clear instructions.

4.70.2.3 change_menu()

```
void change_menu (
            menu * next_menu,
            menu ** menuHead )
```

Function to change the menu.

Parameters

menuHead	is a pointer to our current placement in the linked list
next_menu	is a pointer to the menu we are navigating to

4.70.2.4 change_selected()

```
void change_selected (
            menu ** menuHead,
            DIRECTION d )
```

Function to change which option in a menu is selected.

Parameters

menuHead	pointer to current menu
d	joystick direction used to change the selected item according to user input

4.70.2.5 choose_character()

```
void choose_character ( )
```

Function to let you choose between our two characters. The choice gets saved to SRAM and enables high-score tracking.

4.70.2.6 hello_world()

```
void hello_world ( )
```

Function to write "hello world" to the OLED.

4.70.2.7 hiscore()

```
void hiscore ( )
```

Function to show current high scores for both characters uses information from SRAM.

4.70.2.8 invert_selected()

```
void invert_selected (
            menu * menuPointer )
```

Function to invert the selected row. Selected row is tracked by the "seleced" variable in the struct.

Parameters

menuPointer	pointer to current screen

4.70.2.9 launch_menusystem()

```
void launch_menusystem ( )
```

Function to start our pretetermined menusystem.

4.70.2.10 new_menu()

```
menu* new_menu (
            menu * parent )
```

Function to create a new menu page.

Warning

Uses Malloc to allocate space for the menu in memory

Parameters

parent	uses a pointer to its parent in order to create a "back" - option

4.70.2.11 play_game()

```
void play_game ( )
```

Function to initialize ping-pong game uses CAN to send joystick info over to node 2 receives feedback over CAN and displays it on the OLED.

4.70.2.12 reaction_test()

```
void reaction_test ( )
```

Function to launch a small minigame that lets you test your reaction time sends a start and a stop signal to node 2 which has the timer implemented. you lose if you press too early.

4.70.2.13 reset_scores()

```
void reset_scores ( )
```

Function to set high score values for both characters to 0 in SRAM.

4.70.2.14 set_easy()

```
void set_easy ( )
```

Function to set PID regulator on node 2 to easy mode over CAN.

4.70.2.15 set_hard()

```
void set_hard ( )
```

Function to set PID regulator on node 2 to hard mode over CAN.

4.70.2.16 set_medium()

```
void set_medium ( )
```

Function to set PID regulator on node 2 to medium mode over CAN.

4.70.2.17 show_credits()

```
void show_credits ( )
```

Function to credit the creators of the game on the OLED.

4.70.2.18 wojakprinter()

```
void wojakprinter ( )
```

Function that prints a familiar face to the OLED.

4.70.2.19 write_menu_to_screen()

```
void write_menu_to_screen (
            menu * menuPointer )
```

Function to write the current menu to screen.

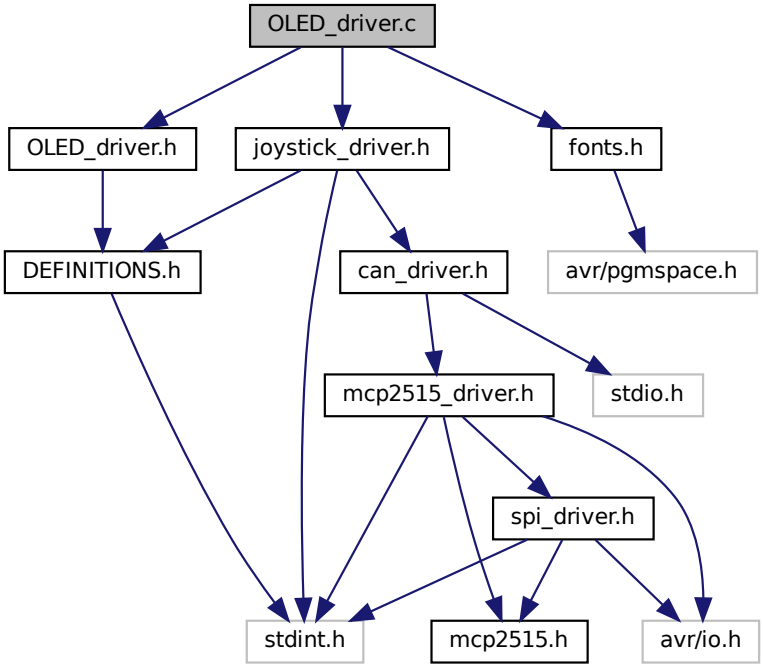
Parameters

menuPointer	pointer to current screen

4.71 OLED_driver.c File Reference

```
#include "OLED_driver.h"
#include "fonts.h"
#include "joystick_driver.h"
```

Include dependency graph for OLED_driver.c:



Functions

- void oled_write_command (char c)

  array to keep track of which pixels on the screen are on and which are off uint8_t oled_array[8][100];
- void oled_write_data (char c)

  Function to write data to OLED display. Data goes in a different memory location than commands.
- void oled_init ()

  Function to initialize OLED display. Writes a series of commands to enable use of this modules other functions,.
- void go_to_line (uint8_t line)

  Function to select a line to write to.
- void go_to_column (uint8_t column)

  Function to select a column to write to.
- void oled_start_write_at (amap ∗atmelMap, uint8_t page, uint8_t lowerCol, uint8_t upperCol)
- void oled_write (amap ∗atmelMap)
- void clear_oled ()

  Function to set all pixels black.
- void clear_oled_new ()
- void oled_write_string (uint8_t startline, char ∗c, uint8_t n)

  Function to write a string at a selected line using a selected font.
- void oled_write_string_inverted (uint8_t startline, char ∗c, uint8_t n)

  Function to write a black string on a white background at a selected line using a selected font.
- void oled_write_char_using_font (char c, uint8_t n)

  Function to write a char using a selected font.
- void oled_write_inverted_char_using_font (char c, uint8_t n)

Function to write a black char on a white background using a selected font.

- void oled_write_char8 (char c)

  Function to write a char using the biggest font.

- void character_printer (uint8_t arr[ ], int width, int height, uint8_t x_offset, uint8_t y_offset, uint8_t inverted)

  Fancy function to write a bit matrix where height and width is divisible by 8 to the OLED. Any picture can be converted to a 0/1 matrix using internet tools and therefore be printed to the screen.

- void oled_drawing_sram (char ∗sram, uint8_t l_slider, uint8_t r_slider, uint8_t write)

  Function to draw on the screen and store the resulting "painting" in sram instead of a bit matrix Otherwise equivalent to oled_drawing.

- void draw_sram ()

  Function to draw on the screen using the sliders.

- void reset_oled_array_sram (char ∗sram)

  Function to null out the array that keeps track of the screen pixels in the sram.

4.71.1 Function Documentation

4.71.1.1 character_printer()

```
void character_printer (
            uint8_t arr[],
            int width,
            int height,
            uint8_t x_offset,
            uint8_t y_offset,
            uint8_t inverted )
```

Fancy function to write a bit matrix where height and width is divisible by 8 to the OLED. Any picture can be converted to a 0/1 matrix using internet tools and therefore be printed to the screen.

Parameters

arr	bit matrix to write
width	width of the matrix (divisible by 8)
height	height of the matrix (divisible by 8)
x_offset	how far from the left border of the screen to place picture
y_offset	how far from the top of the screen to place picture
inverted	1 for an inverted picture, 0 for white on black

4.71.1.2 clear_oled()

```
void clear_oled ( )
```

Function to set all pixels black.

4.71.1.3 clear_oled_new()

```
void clear_oled_new ( )
```

4.71.1.4 draw_sram()

```
void draw_sram ( )
```

Function to draw on the screen using the sliders.

Parameters

l_slider	left slider value
r_slider	right slider value
write	whether to write white or black. 1 for "pencil", 0 for "rubber"

4.71.1.5 go_to_column()

```
void go_to_column (
            uint8_t column )
```

Function to select a column to write to.

Parameters

column	to write. There are 128 colums.

4.71.1.6 go_to_line()

```
void go_to_line (
            uint8_t line )
```

Function to select a line to write to.

Parameters

line	to write. There are 8 lines each consisting of one byte.

4.71.1.7 oled_drawing_sram()

```
void oled_drawing_sram (
             char * sram,
             uint8_t l_slider,
             uint8_t r_slider,
             uint8_t write )
```

Function to draw on the screen and store the resulting "painting" in sram instead of a bit matrix Otherwise equivalent to oled_drawing.

Parameters

sram	sram location in memory

4.71.1.8 oled_init()

```
void oled_init ( )
```

Function to initialize OLED display. Writes a series of commands to enable use of this modules other functions,.

4.71.1.9 oled_start_write_at()

```
void oled_start_write_at (
             amap * atmelMap,
             uint8_t page,
             uint8_t lowerCol,
             uint8_t upperCol )
```

4.71.1.10 oled_write()

```
void oled_write (
             amap * atmelMap )
```

4.71.1.11 oled_write_char8()

```
void oled_write_char8 (
             char c )
```

Function to write a char using the biggest font.

Parameters

c	char to write

4.71.1.12 oled_write_char_using_font()

```
void oled_write_char_using_font (
            char c,
            uint8_t n )
```

Function to write a char using a selected font.

Parameters

c	char to write
n	font to use. 4, 5 or 8 are the usable options. They describe their own size.

4.71.1.13 oled_write_command()

```
void oled_write_command (
            char c )
```

array to keep track of which pixels on the screen are on and which are off uint8_t oled_array[8][100];

/** Function to write commands to OLED display. Commands go in a different memory location than data.

Parameters

c	byte to write

4.71.1.14 oled_write_data()

```
void oled_write_data (
            char c )
```

Function to write data to OLED display. Data goes in a different memory location than commands.

Parameters

c	byte to write

4.71.1.15 oled_write_inverted_char_using_font()

```
void oled_write_inverted_char_using_font (
            char c,
            uint8_t n )
```

Function to write a black char on a white background using a selected font.

Parameters

c	char to write
n	font to use. 4, 5 or 8 are the usable options. They describe their own size.

4.71.1.16 oled_write_string()

```
void oled_write_string (
            uint8_t startline,
            char * c,
            uint8_t n )
```

Function to write a string at a selected line using a selected font.

Parameters

startline	line to write at
c	string to write
n	font to use. 4, 5 or 8 are the usable options. They describe their own size.

4.71.1.17 oled_write_string_inverted()

```
void oled_write_string_inverted (
            uint8_t startline,
            char * c,
            uint8_t n )
```

Function to write a black string on a white background at a selected line using a selected font.

Parameters

startline	line to write at
c	string to write
n	font to use. 4, 5 or 8 are the usable options. They describe their own size.

4.71.1.18 reset_oled_array_sram()

```
void reset_oled_array_sram (
            char * sram )
```

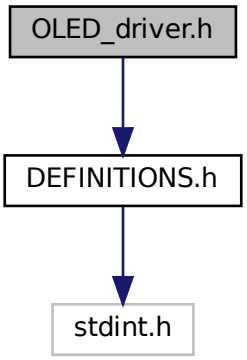
Function to null out the array that keeps track of the screen pixels in the sram.

4.72 OLED_driver.h File Reference

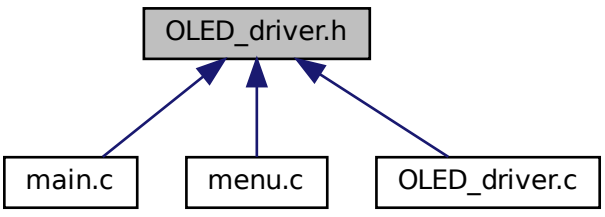
Driver module for handling writing to the OLED display.

```
#include "DEFINITIONS.h"
```

Include dependency graph for OLED_driver.h:



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



Functions

- void oled_write_command (char c)

  array to keep track of which pixels on the screen are on and which are off uint8_t oled_array[8][100];
- void oled_write_data (char c)

  Function to write data to OLED display. Data goes in a different memory location than commands.
- void oled_init ()

  Function to initialize OLED display. Writes a series of commands to enable use of this modules other functions,.
- void go_to_line (uint8_t line)

  Function to select a line to write to.
- void go_to_column (uint8_t column)

  Function to select a column to write to.
- void oled_start_write_at (amap *atmelMap, uint8_t page, uint8_t lowerCol, uint8_t upperCol)
- void oled_write (amap *atmelMap)
- void clear_oled ()

  Function to set all pixels black.
- void clear_oled_new ()
- void oled_write_string (uint8_t startline, char *c, uint8_t n)

  Function to write a string at a selected line using a selected font.
- void oled_write_string_inverted (uint8_t startline, char *c, uint8_t n)

  Function to write a black string on a white background at a selected line using a selected font.
- void oled_write_char_using_font (char c, uint8_t n)

  Function to write a char using a selected font.
- void oled_write_inverted_char_using_font (char c, uint8_t n)

  Function to write a black char on a white background using a selected font.
- void oled_write_char8 (char c)

  Function to write a char using the biggest font.
- void character_printer (uint8_t arr[ ], int width, int height, uint8_t x_offset, uint8_t y_offset, uint8_t inverted)

  Fancy function to write a bit matrix where height and width is divisible by 8 to the OLED. Any picture can be converted to a 0/1 matrix using internet tools and therefore be printed to the screen.
- void draw_sram ()

  Function to draw on the screen using the sliders.
- void reset_oled_array ()

  Function to null out the array that keeps track of the screen pixels.
- void oled_drawing_sram (char *sram, uint8_t l_slider, uint8_t r_slider, uint8_t write)

  Function to draw on the screen and store the resulting "painting" in sram instead of a bit matrix Otherwise equivalent to oled_drawing.
- void reset_oled_array_sram (char *sram)

  Function to null out the array that keeps track of the screen pixels in the sram.

4.72.1 Detailed Description

Driver module for handling writing to the OLED display.

4.72.2 Function Documentation

4.72.2.1 character_printer()

```
void character_printer (
            uint8_t arr[],
            int width,
            int height,
            uint8_t x_offset,
            uint8_t y_offset,
            uint8_t inverted )
```

Fancy function to write a bit matrix where height and width is divisible by 8 to the OLED. Any picture can be converted to a 0/1 matrix using internet tools and therefore be printed to the screen.

Parameters

arr	bit matrix to write
width	width of the matrix (divisible by 8)
height	height of the matrix (divisible by 8)
x_offset	how far from the left border of the screen to place picture
y_offset	how far from the top of the screen to place picture
inverted	1 for an inverted picture, 0 for white on black

4.72.2.2 clear_oled()

```
void clear_oled ( )
```

Function to set all pixels black.

4.72.2.3 clear_oled_new()

```
void clear_oled_new ( )
```

4.72.2.4 draw_sram()

```
void draw_sram ( )
```

Function to draw on the screen using the sliders.

Parameters

l_slider	left slider value
r_slider	right slider value
write	whether to write white or black. 1 for "pencil", 0 for "rubber"

4.72.2.5 go_to_column()

```
void go_to_column (
            uint8_t column )
```

Function to select a column to write to.

Parameters

column	to write. There are 128 colums.

4.72.2.6 go_to_line()

```
void go_to_line (
            uint8_t line )
```

Function to select a line to write to.

Parameters

line	to write. There are 8 lines each consisting of one byte.

4.72.2.7 oled_drawing_sram()

```
void oled_drawing_sram (
            char * sram,
            uint8_t l_slider,
            uint8_t r_slider,
            uint8_t write )
```

Function to draw on the screen and store the resulting "painting" in sram instead of a bit matrix Otherwise equivalent to oled_drawing.

Parameters

sram	sram location in memory

4.72.2.8 oled_init()

```
void oled_init ( )
```

Function to initialize OLED display. Writes a series of commands to enable use of this modules other functions,.

4.72.2.9 oled_start_write_at()

```
void oled_start_write_at (
            amap * atmelMap,
            uint8_t page,
            uint8_t lowerCol,
            uint8_t upperCol )
```

4.72.2.10 oled_write()

```
void oled_write (
            amap * atmelMap )
```

4.72.2.11 oled_write_char8()

```
void oled_write_char8 (
            char c )
```

Function to write a char using the biggest font.

Parameters

c	char to write

4.72.2.12 oled_write_char_using_font()

```
void oled_write_char_using_font (
            char c,
            uint8_t n )
```

Function to write a char using a selected font.

Parameters

c	char to write
n	font to use. 4, 5 or 8 are the usable options. They describe their own size.

4.72.2.13 oled_write_command()

```
void oled_write_command (
            char c )
```

array to keep track of which pixels on the screen are on and which are off uint8_t oled_array[8][100];

/** Function to write commands to OLED display. Commands go in a different memory location than data.

Parameters

c	byte to write

4.72.2.14 oled_write_data()

```
void oled_write_data (
            char c )
```

Function to write data to OLED display. Data goes in a different memory location than commands.

Parameters

c	byte to write

4.72.2.15 oled_write_inverted_char_using_font()

```
void oled_write_inverted_char_using_font (
            char c,
            uint8_t n )
```

Function to write a black char on a white background using a selected font.

Parameters

c	char to write
n	font to use. 4, 5 or 8 are the usable options. They describe their own size.

4.72.2.16 oled_write_string()

```
void oled_write_string (
            uint8_t startline,
```

```
            char * c,
            uint8_t n )
```

Function to write a string at a selected line using a selected font.

Parameters

startline	line to write at
c	string to write
n	font to use. 4, 5 or 8 are the usable options. They describe their own size.

4.72.2.17 oled_write_string_inverted()

```
void oled_write_string_inverted (
            uint8_t startline,
            char * c,
            uint8_t n )
```

Function to write a black string on a white background at a selected line using a selected font.

Parameters

startline	line to write at
c	string to write
n	font to use. 4, 5 or 8 are the usable options. They describe their own size.

4.72.2.18 reset_oled_array()

```
void reset_oled_array ( )
```

Function to null out the array that keeps track of the screen pixels.

4.72.2.19 reset_oled_array_sram()

```
void reset_oled_array_sram (
            char * sram )
```

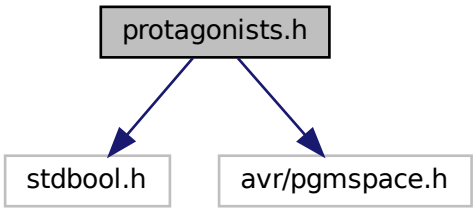
Function to null out the array that keeps track of the screen pixels in the sram.

4.73 protagonists.h File Reference

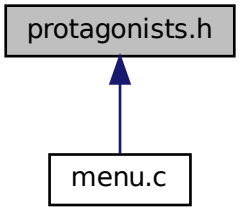
Bit matrixes used to represent pixtures on the screen.

```
#include "stdbool.h"
#include <avr/pgmspace.h>
```

Include dependency graph for protagonists.h:



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



Variables

- const uint8_t PROGMEM wojak [56 ∗40]
- const uint8_t PROGMEM pepe [56 ∗40]

4.73.1 Detailed Description

Bit matrixes used to represent pixtures on the screen.

4.73.2 Variable Documentation

4.73.2.1 pepe

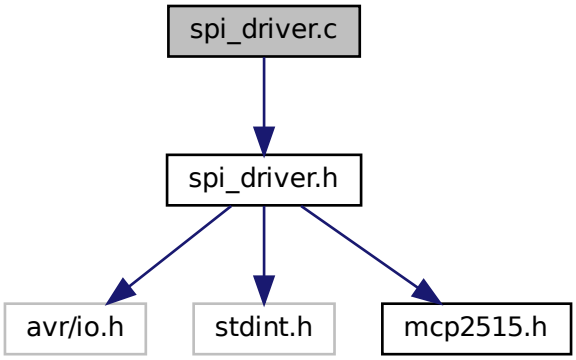
```
const uint8_t PROGMEM pepe[56 *40]
```

4.73.2.2 wojak

```
const uint8_t PROGMEM wojak[56 *40]
```

4.74 spi_driver.c File Reference

```
#include "spi_driver.h"
```
Include dependency graph for spi_driver.c:



Functions

- uint8_t spi_read ()

  Function to read data sent using spi.
- void spi_init ()

  Function to initialize SPI on the AtMega.
- void spi_write (char data)

  Function to write a byte of data using spi.

4.74.1 Function Documentation

4.74.1.1 spi_init()

```
void spi_init ( )
```

Function to initialize SPI on the AtMega.

4.74.1.2 spi_read()

```
uint8_t spi_read ( )
```

Function to read data sent using spi.

4.74.1.3 spi_write()

```
void spi_write (
            char data )
```

Function to write a byte of data using spi.

Parameters

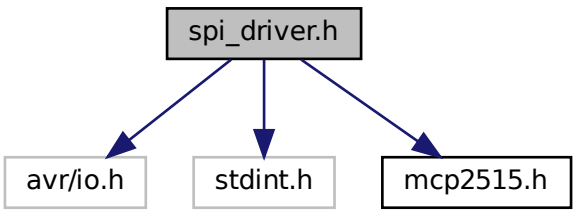
data	byte to write

4.75 spi_driver.h File Reference

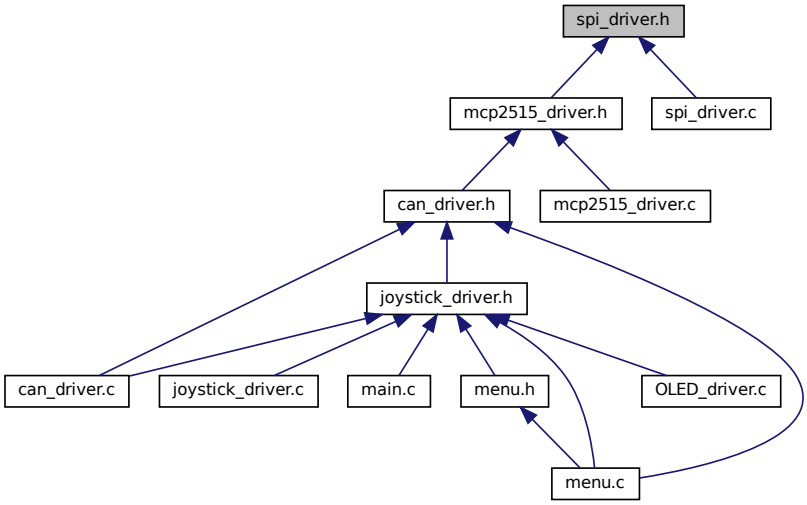
Driver module for SPI communications.

```
#include <avr/io.h>
#include <stdint.h>
#include "mcp2515.h"
```

Include dependency graph for spi_driver.h:



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



Functions

- void spi_init ()

  Function to initialize SPI on the AtMega.
- void spi_write (char data)

  Function to write a byte of data using spi.
- uint8_t spi_read ()

  Function to read data sent using spi.

4.75.1 Detailed Description

Driver module for SPI communications.

4.75.2 Function Documentation

4.75.2.1 spi_init()

```
void spi_init ( )
```

Function to initialize SPI on the AtMega.

4.75.2.2 spi_read()

```
uint8_t spi_read ( )
```

Function to read data sent using spi.

4.75.2.3 spi_write()

```
void spi_write (
            char data )
```

Function to write a byte of data using spi.

Parameters

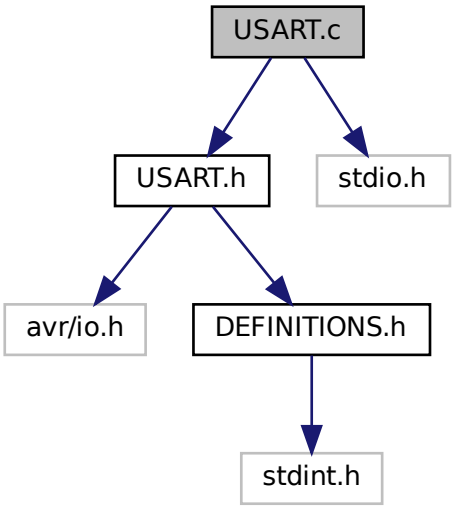
data	byte to write

4.76 sram.h File Reference

4.77 USART.c File Reference

```
#include "USART.h"
#include <stdio.h>
```

Include dependency graph for USART.c:



Macros

- #define clear_bit(reg, bit) (reg &= ~(1 << bit))

Functions

- void USART_Init (unsigned int ubrr)

  Function to initialize USART.
- void USART_Transmit (unsigned char data)

  Function to transmit a 5 to 8 bit char.
- void USART_Transmit9 (unsigned char data)

  Function to transmit a 9 bit char.
- unsigned int USART_Receive9 (void)

  Function to receive a 9 bit char.
- unsigned int USART_Receive (void)

  Function to receive a 5 to 8 bit char.

4.77.1 Macro Definition Documentation

4.77.1.1 clear_bit

```
#define clear_bit(
              reg,
              bit ) (reg &= ~(1 << bit))
```

4.77.2 Function Documentation

4.77.2.1 USART_Init()

```
void USART_Init (
             unsigned int ubrr )
```

Function to initialize USART.

Parameters

ubrr	Value used to choose baudrate, ubrr = F_CPU/16/BAUD-1

4.77.2.2 USART_Receive()

```
unsigned int USART_Receive (
             void  )
```

Function to receive a 5 to 8 bit char.

Returns

data char to send using USART.

4.77.2.3 USART_Receive9()

```
unsigned int USART_Receive9 (
             void  )
```

Function to receive a 9 bit char.

Returns

data char to send using USART.

4.77.2.4 USART_Transmit()

```
void USART_Transmit (
             unsigned char data )
```

Function to transmit a 5 to 8 bit char.

Parameters

data	char to send using USART.

4.77.2.5 USART_Transmit9()

```
void USART_Transmit9 (
             unsigned char data )
```

Function to transmit a 9 bit char.

Parameters

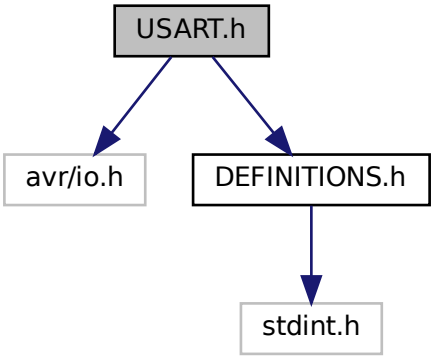
data	char to send using USART.

4.78 USART.h File Reference

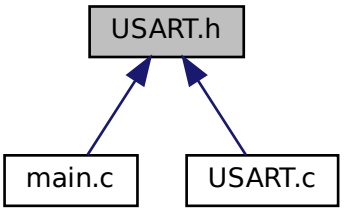
Driver module for handling data transmission using USART.

```
#include <avr/io.h>
#include "DEFINITIONS.h"
```

Include dependency graph for USART.h:



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



Functions

- void USART_Init (unsigned int ubrr)

  Function to initialize USART.
- void USART_Transmit (unsigned char data)

  Function to transmit a 5 to 8 bit char.
- void USART_Transmit9 (unsigned char data)

  Function to transmit a 9 bit char.
- unsigned int USART_Receive (void)

  Function to receive a 5 to 8 bit char.
- unsigned int USART_Receive9 (void)

  Function to receive a 9 bit char.

4.78.1 Detailed Description

Driver module for handling data transmission using USART.

4.78.2 Function Documentation

4.78.2.1 USART_Init()

```
void USART_Init (
            unsigned int ubrr )
```

Function to initialize USART.

Parameters

ubrr	Value used to choose baudrate, ubrr = F_CPU/16/BAUD-1

4.78.2.2 USART_Receive()

```
unsigned int USART_Receive (
            void  )
```

Function to receive a 5 to 8 bit char.

Returns

data char to send using USART.

4.78.2.3 USART_Receive9()

```
unsigned int USART_Receive9 (
            void  )
```

Function to receive a 9 bit char.

Returns

data char to send using USART.

4.78.2.4 USART_Transmit()

```
void USART_Transmit (
            unsigned char data )
```

Function to transmit a 5 to 8 bit char.

Parameters

data	char to send using USART.

4.78.2.5 USART_Transmit9()

```
void USART_Transmit9 (
            unsigned char data )
```

Function to transmit a 9 bit char.

Parameters

data	char to send using USART.

Index