

GRIGORA

1.0.0

Generated by Doxygen 1.9.1

1 Todo List	1
2 Data Structure Index	1
2.1 Data Structures	1
3 File Index	2
3.1 File List	2
4 Data Structure Documentation	5
4.1 BatteryConfigs_t Struct Reference	5
4.1.1 Detailed Description	5
4.1.2 Field Documentation	5
4.2 BUTTON_CONFIGS_t Struct Reference	6
4.2.1 Detailed Description	6
4.2.2 Field Documentation	6
4.3 CONNECTIONS_t Struct Reference	7
4.3.1 Detailed Description	7
4.3.2 Field Documentation	7
4.4 IR_CONFIG_t Struct Reference	8
4.4.1 Detailed Description	8
4.4.2 Field Documentation	8
4.5 LED_CONFIGS_t Struct Reference	9
4.5.1 Detailed Description	9
4.5.2 Field Documentation	9
4.6 LIFTER_CONFIGS_t Struct Reference	10
4.6.1 Detailed Description	10
4.6.2 Field Documentation	10
4.7 NRF24_t Struct Reference	12
4.7.1 Detailed Description	13
4.7.2 Field Documentation	13
4.8 ORDER_INFO_t Struct Reference	15
4.8.1 Detailed Description	15
4.8.2 Field Documentation	16
4.9 PinConfig_t Struct Reference	16
4.9.1 Detailed Description	16
4.9.2 Field Documentation	17
4.10 PINS_t Struct Reference	17
4.10.1 Detailed Description	18
4.10.2 Field Documentation	18
4.11 SPI_CONFIG_t Struct Reference	19
4.11.1 Detailed Description	19
4.11.2 Field Documentation	19
4.12 SPI_CONNECTIONS_t Struct Reference	21

4.12.1 Detailed Description	21
4.12.2 Field Documentation	21
4.13 SPI_PINS_t Struct Reference	22
4.13.1 Detailed Description	22
4.13.2 Field Documentation	22
4.14 UART_CFG_t Struct Reference	23
4.14.1 Detailed Description	23
4.14.2 Field Documentation	23
4.15 WHEELS_CONFIG_t Struct Reference	24
4.15.1 Detailed Description	24
4.15.2 Field Documentation	24
5 File Documentation	26
5.1 code/include/app.h File Reference	26
5.1.1 Detailed Description	28
5.1.2 Enumeration Type Documentation	28
5.1.3 Function Documentation	31
5.2 code/include/app_cfg.h File Reference	34
5.2.1 Detailed Description	34
5.2.2 Macro Definition Documentation	34
5.2.3 Enumeration Type Documentation	35
5.3 code/include/BATTERY.h File Reference	36
5.3.1 Detailed Description	36
5.3.2 Function Documentation	37
5.4 code/include/BATTERY_cfg.h File Reference	37
5.4.1 Detailed Description	38
5.4.2 Macro Definition Documentation	38
5.4.3 Variable Documentation	39
5.5 code/include/BIT_MATH.h File Reference	39
5.5.1 Detailed Description	40
5.5.2 Macro Definition Documentation	40
5.6 code/include/BUTTON.h File Reference	44
5.6.1 Detailed Description	45
5.6.2 Enumeration Type Documentation	45
5.6.3 Function Documentation	46
5.7 code/include/BUTTON_cfg.h File Reference	47
5.7.1 Detailed Description	47
5.7.2 Enumeration Type Documentation	47
5.7.3 Variable Documentation	48
5.8 code/include/DIO.h File Reference	48
5.8.1 Enumeration Type Documentation	49
5.8.2 Function Documentation	50

5.9 code/include/DIO_cfg.h File Reference	55
5.9.1 Detailed Description	55
5.9.2 Variable Documentation	56
5.10 code/include/DIO_reg.h File Reference	56
5.10.1 Detailed Description	57
5.10.2 Macro Definition Documentation	57
5.11 code/include/EXTI.h File Reference	60
5.11.1 Detailed Description	61
5.11.2 Enumeration Type Documentation	61
5.11.3 Function Documentation	62
5.12 code/include/EXTI_cfg.h File Reference	64
5.12.1 Detailed Description	64
5.12.2 Macro Definition Documentation	65
5.13 code/include/EXTI_reg.h File Reference	65
5.13.1 Detailed Description	66
5.13.2 Macro Definition Documentation	66
5.13.3 Enumeration Type Documentation	67
5.14 code/include/GIE.h File Reference	69
5.14.1 Detailed Description	69
5.14.2 Function Documentation	70
5.15 code/include/IR.h File Reference	71
5.15.1 Detailed Description	72
5.15.2 Enumeration Type Documentation	72
5.15.3 Function Documentation	72
5.16 code/include/IR_cfg.h File Reference	75
5.16.1 Detailed Description	76
5.16.2 Variable Documentation	76
5.17 code/include/LED.h File Reference	77
5.17.1 Detailed Description	77
5.17.2 Enumeration Type Documentation	77
5.17.3 Function Documentation	78
5.18 code/include/LED_cfg.h File Reference	80
5.18.1 Detailed Description	81
5.18.2 Variable Documentation	81
5.19 code/include/LIFTER.h File Reference	82
5.19.1 Detailed Description	82
5.19.2 Function Documentation	83
5.20 code/include/LIFTER_cfg.h File Reference	85
5.20.1 Detailed Description	86
5.20.2 Macro Definition Documentation	87
5.20.3 Enumeration Type Documentation	87
5.20.4 Variable Documentation	87

5.21 code/include/MATH.h File Reference	88
5.21.1 Detailed Description	89
5.21.2 Macro Definition Documentation	89
5.22 code/include/NRF24.h File Reference	100
5.22.1 Detailed Description	101
5.22.2 Function Documentation	101
5.23 code/include/NRF24_cfg.h File Reference	103
5.23.1 Detailed Description	103
5.23.2 Enumeration Type Documentation	104
5.23.3 Variable Documentation	104
5.24 code/include/NRF24_reg.h File Reference	104
5.24.1 Detailed Description	106
5.24.2 Macro Definition Documentation	106
5.24.3 Enumeration Type Documentation	111
5.25 code/include/registers.h File Reference	116
5.25.1 Detailed Description	116
5.25.2 Macro Definition Documentation	117
5.26 code/include/SPI.h File Reference	118
5.26.1 Detailed Description	119
5.26.2 Function Documentation	119
5.27 code/include/SPI_cfg.h File Reference	122
5.27.1 Detailed Description	123
5.27.2 Enumeration Type Documentation	123
5.27.3 Variable Documentation	124
5.28 code/include/SPI_reg.h File Reference	125
5.28.1 Detailed Description	125
5.28.2 Macro Definition Documentation	126
5.28.3 Enumeration Type Documentation	126
5.29 code/include/SREG.h File Reference	127
5.29.1 Detailed Description	127
5.29.2 Macro Definition Documentation	127
5.29.3 Enumeration Type Documentation	127
5.30 code/include/STD_TYPES.h File Reference	128
5.30.1 Detailed Description	129
5.30.2 Macro Definition Documentation	129
5.30.3 Typedef Documentation	129
5.30.4 Enumeration Type Documentation	130
5.31 code/include/TIMER.h File Reference	131
5.31.1 Detailed Description	133
5.31.2 Enumeration Type Documentation	133
5.31.3 Function Documentation	136
5.32 code/include/TIMER_cfg.h File Reference	144

5.32.1 Detailed Description	144
5.33 code/include/TIMER_reg.h File Reference	144
5.33.1 Detailed Description	146
5.33.2 Macro Definition Documentation	146
5.33.3 Enumeration Type Documentation	151
5.34 code/include/UART.h File Reference	156
5.34.1 Detailed Description	158
5.34.2 Function Documentation	159
5.35 code/include/UART_cfg.h File Reference	172
5.35.1 Detailed Description	173
5.35.2 Macro Definition Documentation	173
5.35.3 Enumeration Type Documentation	173
5.35.4 Variable Documentation	175
5.36 code/include/UART_reg.h File Reference	176
5.36.1 Detailed Description	177
5.36.2 Macro Definition Documentation	177
5.36.3 Enumeration Type Documentation	178
5.37 code/include/UART_service.h File Reference	179
5.37.1 Detailed Description	181
5.37.2 Function Documentation	181
5.38 code/include/WHEELS.h File Reference	189
5.38.1 Detailed Description	190
5.38.2 Enumeration Type Documentation	190
5.38.3 Function Documentation	190
5.39 code/include/WHEELS_cfg.h File Reference	195
5.39.1 Detailed Description	195
5.39.2 Enumeration Type Documentation	195
5.39.3 Variable Documentation	196
5.40 code/src/appTest.c File Reference	196
5.40.1 Function Documentation	197
5.41 code/src/BATTERY.c File Reference	198
5.41.1 Detailed Description	198
5.42 code/src/BATTERY_cfg.c File Reference	198
5.42.1 Detailed Description	199
5.42.2 Variable Documentation	199
5.43 code/src/BUTTON.c File Reference	200
5.43.1 Detailed Description	200
5.43.2 Function Documentation	200
5.44 code/src/BUTTON_cfg.c File Reference	202
5.44.1 Variable Documentation	202
5.45 code/src/DIO.c File Reference	203
5.45.1 Detailed Description	204

5.45.2 Macro Definition Documentation	204
5.45.3 Function Documentation	205
5.46 code/src/DIO_cfg.c File Reference	208
5.46.1 Detailed Description	208
5.46.2 Variable Documentation	209
5.47 code/src/EXTI.c File Reference	209
5.47.1 Detailed Description	210
5.47.2 Function Documentation	211
5.48 code/src/GIE.c File Reference	214
5.48.1 Detailed Description	214
5.48.2 Function Documentation	215
5.49 code/src/IR.c File Reference	216
5.49.1 Function Documentation	217
5.50 code/src/IR_cfg.c File Reference	219
5.50.1 Variable Documentation	219
5.51 code/src/LED.c File Reference	220
5.51.1 Detailed Description	221
5.51.2 Function Documentation	221
5.52 code/src/LED_cfg.c File Reference	223
5.52.1 Detailed Description	223
5.52.2 Variable Documentation	224
5.53 code/src/LIFTER.c File Reference	224
5.53.1 Detailed Description	225
5.53.2 Function Documentation	225
5.54 code/src/LIFTER_cfg.c File Reference	228
5.54.1 Detailed Description	228
5.54.2 Variable Documentation	229
5.55 code/src/NRF24.c File Reference	230
5.55.1 Detailed Description	230
5.55.2 Function Documentation	231
5.56 code/src/NRF24_cfg.c File Reference	232
5.56.1 Detailed Description	233
5.56.2 Variable Documentation	233
5.57 code/src/SPI.c File Reference	233
5.57.1 Detailed Description	234
5.57.2 Function Documentation	234
5.57.3 Variable Documentation	238
5.58 code/src/SPI_cfg.c File Reference	238
5.58.1 Detailed Description	238
5.58.2 Variable Documentation	239
5.59 code/src/TIMER.c File Reference	239
5.59.1 Detailed Description	240

5.59.2 Function Documentation	241
5.60 code/src/UART.c File Reference	254
5.60.1 Detailed Description	256
5.60.2 Typedef Documentation	256
5.60.3 Function Documentation	256
5.61 code/src/UART_cfg.c File Reference	271
5.61.1 Detailed Description	271
5.61.2 Variable Documentation	272
5.62 code/src/UART_service.c File Reference	273
5.62.1 Function Documentation	274
5.63 code/src/WHEELS.c File Reference	283
5.63.1 Detailed Description	284
5.63.2 Function Documentation	285
5.64 code/src/WHEELS_cfg.c File Reference	289
5.64.1 Detailed Description	289
5.64.2 Variable Documentation	290
6 Example Documentation	290
6.1 LED_Toggle	290
Index	291

1 Todo List

File [EXTI.c](#)

Debug configuration of INT4 to INT7 as edge triggered not working.

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

BatteryConfigs_t	5
BUTTON_CONFIGS_t	6
CONNECTIONS_t	7
IR_CONFIG_t	8
LED_CONFIGS_t	9
LIFTER_CONFIGS_t	10
NRF24_t	12

ORDER_INFO_t	15
Information of an order	
PinConfig_t	16
PINS_t	17
SPI_CONFIG_t	19
SPI_CONNECTIONS_t	21
SPI_PINS_t	22
UART_CFG_t	23
WHEELS_CONFIG_t	24

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

code/include/app.h	26
Types and APIs for application app.c	
code/include/app_cfg.h	34
Configuration header file for app.c	
code/include/BATTERY.h	36
Interfaces header file for BATTERY.c	
code/include/BATTERY_cfg.h	37
Configuration header file for BATTERY.c	
code/include/BIT_MATH.h	39
Common bit manipulation operations	
code/include/BUTTON.h	44
Interfaces header file for BUTTON.c	
code/include/BUTTON_cfg.h	47
Configuration header file for BUTTON.c	
code/include/DIO.h	48
code/include/DIO_cfg.h	55
Configuration header file for DIO.c	
code/include/DIO_reg.h	56
DIO Registers of ATmega328p microcontroller	
code/include/EXTI.h	60
Interfaces header file for EXTI.c	
code/include/EXTI_cfg.h	64
Configuration header file for EXTI.c	

code/include/ EXTI_reg.h EXTI Registers of ATmega328p MCU	65
code/include/ GIE.h Interfaces header file for GIE.c	69
code/include/ IR.h Interfaces header file for IR.c	71
code/include/ IR_cfg.h Configuration header file for IR.c	75
code/include/ LED.h Interfaces header file for LED.c	77
code/include/ LED_cfg.h Configuration header file for LED.c	80
code/include/ LIFTER.h Interfaces header file for LIFTER.c	82
code/include/ LIFTER_cfg.h Configuration header file for LIFTER.c	85
code/include/ MATH.h Common math functions and constants	88
code/include/ NRF24.h Interfaces header file for NRF24.c	100
code/include/ NRF24_cfg.h Configuration header file for NRF24.c	103
code/include/ NRF24_reg.h Registers of NRF24L01 wireless transceiver module	104
code/include/ registers.h Registers of ATmega328p MCU	116
code/include/ SPI.h Interfaces header file for SPI.c	118
code/include/ SPI_cfg.h Configuration header file for SPI.c	122
code/include/ SPI_reg.h SPI Registers of ATmega328p MCU	125
code/include/ SREG.h Status Registers of ATmega328p MCU	127
code/include/ STD_TYPES.h Standard data types For AVR Microcontrollers	128
code/include/ TIMER.h Interfaces header file for TIMER.c	131
code/include/ TIMER_cfg.h Configuration header file for TIMER.c	144
code/include/ TIMER_reg.h Registers of Timers of ATmega328p MCU	144

<code>code/include/UART.h</code> Interfaces header file for <code>UART.c</code>	156
<code>code/include/UART_cfg.h</code> Configuration header file for <code>UART.c</code>	172
<code>code/include/UART_reg.h</code> Registers of UART of ATmega328p MCU	176
<code>code/include/UART_service.h</code> UART services interfaces file	179
<code>code/include/WHEELS.h</code> Interfaces header file for <code>WHEELS.c</code>	189
<code>code/include/WHEELS_cfg.h</code> Configuration header file for <code>WHEELS.c</code>	195
<code>code/src/appTest.c</code>	196
<code>code/src/BATTERY.c</code> Battery Management Module	198
<code>code/src/BATTERY_cfg.c</code> Configuration source file for <code>BATTERY.c</code>	198
<code>code/src/BUTTON.c</code> Button Management Module	200
<code>code/src/BUTTON_cfg.c</code>	202
<code>code/src/DIO.c</code> Digital Input Output (DIO) driver for Atmega128 microcontroller	203
<code>code/src/DIO_cfg.c</code> Configuration source file for <code>DIO.c</code>	208
<code>code/src/EXTI.c</code> External Interrupts (EXTI) driver for Atmega128 microcontroller	209
<code>code/src/GIE.c</code> Global Interrupt Enable (GIE)	214
<code>code/src/IR.c</code>	216
<code>code/src/IR_cfg.c</code>	219
<code>code/src/LED.c</code> LED Driver	220
<code>code/src/LED_cfg.c</code> Configuration source file for <code>LED.c</code>	223
<code>code/src/LIFTER.c</code> Lifter module that control the movement of the lifter actuator	224
<code>code/src/LIFTER_cfg.c</code> Configuration source file for <code>LIFTER.c</code>	228
<code>code/src/NRF24.c</code> NRF24 wireless transceiver module driver	230

code/src/NRF24_cfg.c	
Configuration source file for NRF24.c	232
code/src/SPI.c	
SPI driver for ATMEGA128 microcontroller	233
code/src/SPI_cfg.c	
Configuration source file for SPI.c	238
code/src/TIMER.c	
Timer driver for ATMEGA128 microcontroller	239
code/src/UART.c	
UART driver for ATMEGA128 microcontroller	254
code/src/UART_cfg.c	
Configuration source file for UART.c	271
code/src/UART_service.c	273
code/src/WHEELS.c	
This is the source file for the WHEELS module based on the WHEELS module	283
code/src/WHEELS_cfg.c	
Configuration source file for WHEELS.c	289

4 Data Structure Documentation

4.1 BatteryConfigs_t Struct Reference

```
#include <BATTERY_cfg.h>
```

Data Fields

- [PIN_t pin](#)
- [PORT_t port](#)
- [f32 minVoltage](#)
- [f32 maxVoltage](#)

4.1.1 Detailed Description

Definition at line 21 of file BATTERY_cfg.h.

4.1.2 Field Documentation

4.1.2.1 maxVoltage [f32 maxVoltage](#)

Definition at line 25 of file BATTERY_cfg.h.

4.1.2.2 minVoltage [f32](#) minVoltage

Definition at line 24 of file BATTERY_cfg.h.

4.1.2.3 pin [PIN_t](#) pin

Definition at line 22 of file BATTERY_cfg.h.

4.1.2.4 port [PORT_t](#) port

Definition at line 23 of file BATTERY_cfg.h.

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

- [code/include/BATTERY_cfg.h](#)

4.2 BUTTON_CONFIGS_t Struct Reference

```
#include <BUTTON_cfg.h>
```

Data Fields

- [BUTTON_t](#) button
- [PIN_t](#) pin
- [PORT_t](#) port
- [ACTIVATION_STATUS_t](#) ActiveHighOrLow
- [DEBOUNCE_t](#) debounceStatus

4.2.1 Detailed Description

Definition at line 17 of file BUTTON_cfg.h.

4.2.2 Field Documentation

4.2.2.1 ActiveHighOrLow [ACTIVATION_STATUS_t](#) ActiveHighOrLow

Definition at line 21 of file BUTTON_cfg.h.

4.2.2.2 button [BUTTON_t](#) button

Definition at line 18 of file [BUTTON_cfg.h](#).

4.2.2.3 debounceStatus [DEBOUNCE_t](#) debounceStatus

Definition at line 22 of file [BUTTON_cfg.h](#).

4.2.2.4 pin [PIN_t](#) pin

Definition at line 19 of file [BUTTON_cfg.h](#).

4.2.2.5 port [PORT_t](#) port

Definition at line 20 of file [BUTTON_cfg.h](#).

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

- [code/include/BUTTON_cfg.h](#)

4.3 CONNECTIONS_t Struct Reference

```
#include <NRF24_cfg.h>
```

Data Fields

- [PIN_t](#) pin
- [PORT_t](#) port

4.3.1 Detailed Description

Definition at line 12 of file [NRF24_cfg.h](#).

4.3.2 Field Documentation

4.3.2.1 pin [PIN_t](#) pin

Definition at line 13 of file [NRF24_cfg.h](#).

4.3.2.2 port [PORT_t](#) port

Definition at line 14 of file NRF24_cfg.h.

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

- [code/include/NRF24_cfg.h](#)

4.4 IR_CONFIG_t Struct Reference

```
#include <IR_cfg.h>
```

Data Fields

- [IR_SENSOR_t](#) sensor
- [PIN_t](#) pin
- [PORT_t](#) port
- [ACTIVATION_STATUS_t](#) ActiveHighOrLow

4.4.1 Detailed Description

Definition at line 12 of file IR_cfg.h.

4.4.2 Field Documentation

4.4.2.1 ActiveHighOrLow [ACTIVATION_STATUS_t](#) ActiveHighOrLow

Definition at line 16 of file IR_cfg.h.

4.4.2.2 pin [PIN_t](#) pin

Definition at line 14 of file IR_cfg.h.

4.4.2.3 port [PORT_t](#) port

Definition at line 15 of file IR_cfg.h.

4.4.2.4 sensor [IR_SENSOR_t](#) sensor

Definition at line 13 of file IR_cfg.h.

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

- [code/include/IR_cfg.h](#)

4.5 LED_CONFIGS_t Struct Reference

```
#include <LED_cfg.h>
```

Data Fields

- [LED_t](#) led
- [PIN_t](#) pin
- [PORT_t](#) port

4.5.1 Detailed Description

Definition at line 12 of file LED_cfg.h.

4.5.2 Field Documentation

4.5.2.1 led [LED_t](#) led

Definition at line 13 of file LED_cfg.h.

4.5.2.2 pin [PIN_t](#) pin

Definition at line 14 of file LED_cfg.h.

4.5.2.3 port [PORT_t](#) port

Definition at line 15 of file LED_cfg.h.

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

- [code/include/LED_cfg.h](#)

4.6 LIFTER_CONFIGS_t Struct Reference

```
#include <LIFTER_cfg.h>
```

Data Fields

- [PIN_t directionPin](#)
- [PORT_t directionPinPort](#)
- [PIN_t pulsePin](#)
- [PORT_t pulsePinPort](#)
- [PIN_t enablePin](#)
- [PORT_t enablePinPort](#)
- [u8 overallStroke](#)
- [u16 pulsesPerRevolution](#)
- [u8 revolutionStroke](#)
- [u8 speed](#)

4.6.1 Detailed Description

Definition at line 25 of file LIFTER_cfg.h.

4.6.2 Field Documentation

4.6.2.1 directionPin [PIN_t](#) directionPin

Definition at line 26 of file LIFTER_cfg.h.

4.6.2.2 directionPinPort [PORT_t](#) directionPinPort

Definition at line 27 of file LIFTER_cfg.h.

4.6.2.3 enablePin [PIN_t](#) enablePin

Definition at line 30 of file LIFTER_cfg.h.

4.6.2.4 enablePinPort [PORT_t](#) enablePinPort

Definition at line 31 of file LIFTER_cfg.h.

4.6.2.5 overallStroke `u8 overallStroke`

Definition at line 32 of file LIFTER_cfg.h.

4.6.2.6 pulsePin `PIN_t pulsePin`

Definition at line 28 of file LIFTER_cfg.h.

4.6.2.7 pulsePinPort `PORT_t pulsePinPort`

Definition at line 29 of file LIFTER_cfg.h.

4.6.2.8 pulsesPerRevolution `u16 pulsesPerRevolution`

Definition at line 33 of file LIFTER_cfg.h.

4.6.2.9 revolutionStroke `u8 revolutionStroke`

Definition at line 34 of file LIFTER_cfg.h.

4.6.2.10 speed `u8 speed`

Definition at line 35 of file LIFTER_cfg.h.

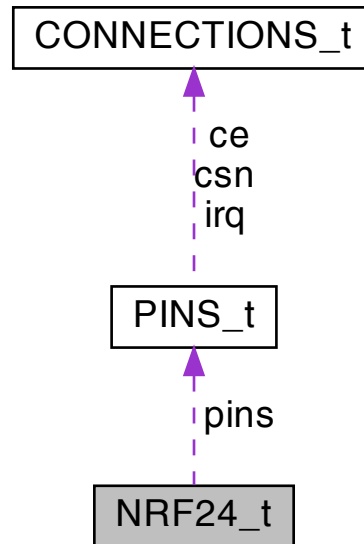
The documentation for this struct was generated from the following file:

- [code/include/LIFTER_cfg.h](#)

4.7 NRF24_t Struct Reference

```
#include <NRF24_cfg.h>
```

Collaboration diagram for NRF24_t:



Data Fields

- [PINS_t pins](#)
- [u8 channel](#)
- [u8 payload_len](#)
- [u8 addressWidth](#)
- [u8 txAddress \[5\]](#)
- [u8 txPayload \[32\]](#)
- [u8 rx0Address \[5\]](#)
- [u8 rx0Payload \[32\]](#)
- [u8 rx1Address \[5\]](#)
- [u8 rx1Payload \[32\]](#)
- [u8 rx2Address \[5\]](#)
- [u8 rx2Payload \[32\]](#)
- [u8 rx3Address \[5\]](#)
- [u8 rx3Payload \[32\]](#)
- [u8 rx4Address \[5\]](#)
- [u8 rx4Payload \[32\]](#)
- [u8 rx5Address \[5\]](#)
- [u8 rx5Payload \[32\]](#)
- [PIPE_t rxPipe](#)

4.7.1 Detailed Description

Definition at line 32 of file NRF24_cfg.h.

4.7.2 Field Documentation

4.7.2.1 addressWidth `u8 addressWidth`

Definition at line 36 of file NRF24_cfg.h.

4.7.2.2 channel `u8 channel`

Definition at line 34 of file NRF24_cfg.h.

4.7.2.3 payload_len `u8 payload_len`

Definition at line 35 of file NRF24_cfg.h.

4.7.2.4 pins `PINS_t pins`

Definition at line 33 of file NRF24_cfg.h.

4.7.2.5 rx0Address `u8 rx0Address[5]`

Definition at line 39 of file NRF24_cfg.h.

4.7.2.6 rx0Payload `u8 rx0Payload[32]`

Definition at line 40 of file NRF24_cfg.h.

4.7.2.7 rx1Address `u8 rx1Address[5]`

Definition at line 41 of file NRF24_cfg.h.

4.7.2.8 rx1Payload `u8 rx1Payload[32]`

Definition at line 42 of file NRF24_cfg.h.

4.7.2.9 rx2Address `u8 rx2Address[5]`

Definition at line 43 of file NRF24_cfg.h.

4.7.2.10 rx2Payload `u8 rx2Payload[32]`

Definition at line 44 of file NRF24_cfg.h.

4.7.2.11 rx3Address `u8 rx3Address[5]`

Definition at line 45 of file NRF24_cfg.h.

4.7.2.12 rx3Payload `u8 rx3Payload[32]`

Definition at line 46 of file NRF24_cfg.h.

4.7.2.13 rx4Address `u8 rx4Address[5]`

Definition at line 47 of file NRF24_cfg.h.

4.7.2.14 rx4Payload `u8 rx4Payload[32]`

Definition at line 48 of file NRF24_cfg.h.

4.7.2.15 rx5Address `u8 rx5Address[5]`

Definition at line 49 of file NRF24_cfg.h.

4.7.2.16 rx5Payload `u8 rx5Payload[32]`

Definition at line 50 of file NRF24_cfg.h.

4.7.2.17 rxPipe `PIPE_t rxPipe`

Definition at line 51 of file NRF24_cfg.h.

4.7.2.18 txAddress `u8 txAddress[5]`

Definition at line 37 of file NRF24_cfg.h.

4.7.2.19 txPayload `u8 txPayload[32]`

Definition at line 38 of file NRF24_cfg.h.

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

- code/include/NRF24_cfg.h

4.8 ORDER_INFO_t Struct Reference

Information of an order.

```
#include <app.h>
```

Data Fields

- [ORDER_AVAILABILITY_t](#) recievedOrNot
- [COLUMN_t](#) column
- [ROW_t](#) row
- [ORDER_TRACKING_t](#) deliveryState

4.8.1 Detailed Description

Information of an order.

The order is composed of the following information:

- Order ID: The ID of the order.
- Availability: If the order has been received or not.
- Column: The column coordinate of the order.
- Row: The row coordinate of the order.
- Tracking state: states of the order, member of [ORDER_TRACKING_t](#).

Definition at line 91 of file app.h.

4.8.2 Field Documentation

4.8.2.1 column [COLUMN_t](#) column

Definition at line 93 of file app.h.

4.8.2.2 deliveryState [ORDER_TRACKING_t](#) deliveryState

Definition at line 95 of file app.h.

4.8.2.3 recievedOrNot [ORDER_AVAILABILITY_t](#) recievedOrNot

Definition at line 92 of file app.h.

4.8.2.4 row [ROW_t](#) row

Definition at line 94 of file app.h.

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

- [code/include/app.h](#)

4.9 PinConfig_t Struct Reference

```
#include <DIO_cfg.h>
```

Data Fields

- [PIN_t](#) pin
- [PORT_t](#) port
- [DIR_t](#) direction
- [PULLUP_t](#) pullup

4.9.1 Detailed Description

Note

pins a of SSegment connected to MSB and dot point to LSB if no BCD is used

Definition at line 16 of file DIO_cfg.h.

4.9.2 Field Documentation

4.9.2.1 direction `DIR_t` direction

Definition at line 19 of file DIO_cfg.h.

4.9.2.2 pin `PIN_t` pin

Definition at line 17 of file DIO_cfg.h.

4.9.2.3 port `PORT_t` port

Definition at line 18 of file DIO_cfg.h.

4.9.2.4 pullup `PULLUP_t` pullup

Definition at line 20 of file DIO_cfg.h.

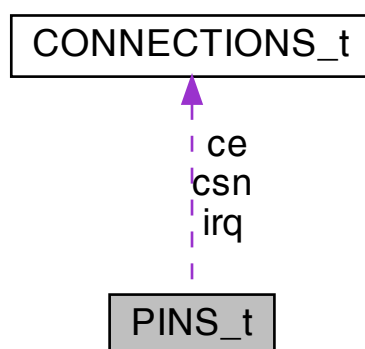
The documentation for this struct was generated from the following file:

- code/include/DIO_cfg.h

4.10 PINS_t Struct Reference

```
#include <NRF24_cfg.h>
```

Collaboration diagram for PINS_t:



Data Fields

- [CONNECTIONS_t ce](#)
- [CONNECTIONS_t csn](#)
- [CONNECTIONS_t irq](#)

4.10.1 Detailed Description

Definition at line 17 of file NRF24_cfg.h.

4.10.2 Field Documentation

4.10.2.1 **ce** [CONNECTIONS_t ce](#)

Definition at line 18 of file NRF24_cfg.h.

4.10.2.2 **csn** [CONNECTIONS_t csn](#)

Definition at line 19 of file NRF24_cfg.h.

4.10.2.3 **irq** [CONNECTIONS_t irq](#)

Definition at line 20 of file NRF24_cfg.h.

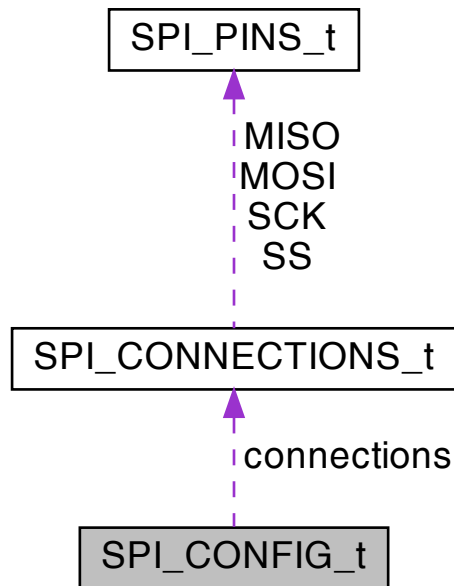
The documentation for this struct was generated from the following file:

- code/include/[NRF24_cfg.h](#)

4.11 SPI_CONFIG_t Struct Reference

```
#include <SPI_cfg.h>
```

Collaboration diagram for SPI_CONFIG_t:



Data Fields

- [SPI_CONNECTIONS_t connections](#)
- [SPI_MODE_t mode](#)
- [SPI_DATA_ORDER_t dataOrder](#)
- [SPI_CLOCK_MODE_t clockMode](#)
- [SPI_PRESCALER_t clockDivider](#)
- [SPI_DOUBLE_SPEED_t doubleSpeed](#)

4.11.1 Detailed Description

Definition at line 56 of file `SPI_cfg.h`.

4.11.2 Field Documentation

4.11.2.1 clockDivider [SPI_PRESCALER_t](#) clockDivider

Definition at line 61 of file SPI_cfg.h.

4.11.2.2 clockMode [SPI_CLOCK_MODE_t](#) clockMode

Definition at line 60 of file SPI_cfg.h.

4.11.2.3 connections [SPI_CONNECTIONS_t](#) connections

Definition at line 57 of file SPI_cfg.h.

4.11.2.4 dataOrder [SPI_DATA_ORDER_t](#) dataOrder

Definition at line 59 of file SPI_cfg.h.

4.11.2.5 doubleSpeed [SPI_DOUBLE_SPEED_t](#) doubleSpeed

Definition at line 62 of file SPI_cfg.h.

4.11.2.6 mode [SPI_MODE_t](#) mode

Definition at line 58 of file SPI_cfg.h.

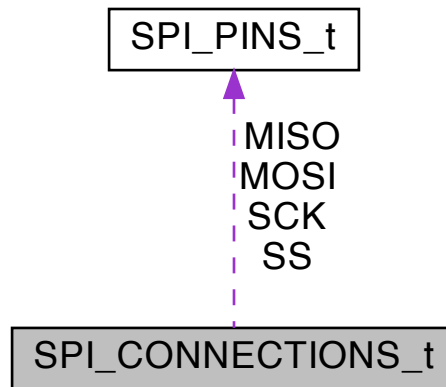
The documentation for this struct was generated from the following file:

- [code/include/SPI_cfg.h](#)

4.12 SPI_CONNECTIONS_t Struct Reference

```
#include <SPI_cfg.h>
```

Collaboration diagram for SPI_CONNECTIONS_t:



Data Fields

- [SPI_PINS_t SS](#)
- [SPI_PINS_t MOSI](#)
- [SPI_PINS_t MISO](#)
- [SPI_PINS_t SCK](#)

4.12.1 Detailed Description

Definition at line 49 of file `SPI_cfg.h`.

4.12.2 Field Documentation

4.12.2.1 MISO [SPI_PINS_t MISO](#)

Definition at line 52 of file `SPI_cfg.h`.

4.12.2.2 MOSI [SPI_PINS_t MOSI](#)

Definition at line 51 of file `SPI_cfg.h`.

4.12.2.3 SCK `SPI_PINS_t` SCK

Definition at line 53 of file SPI_cfg.h.

4.12.2.4 SS `SPI_PINS_t` SS

Definition at line 50 of file SPI_cfg.h.

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

- code/include/SPI_cfg.h

4.13 SPI_PINS_t Struct Reference

```
#include <SPI_cfg.h>
```

Data Fields

- `PIN_t` pin
- `PORT_t` port

4.13.1 Detailed Description

Definition at line 44 of file SPI_cfg.h.

4.13.2 Field Documentation

4.13.2.1 pin `PIN_t` pin

Definition at line 45 of file SPI_cfg.h.

4.13.2.2 port `PORT_t` port

Definition at line 46 of file SPI_cfg.h.

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

- code/include/SPI_cfg.h

4.14 UART_CFG_t Struct Reference

```
#include <UART_cfg.h>
```

Data Fields

- `const u32 baud_rate`
- `UART_DATA_BITS_t data_bits`
- `UART_STOP_BITS_t stop_bits`
- `UART_PARITY_t parity`
- `UART_MODE_t mode`
- `UART_MODE_TYPE_t mode_type`
- `UART_CLOCK_POLARITY_t clock_polarity`

4.14.1 Detailed Description

Definition at line 54 of file UART_cfg.h.

4.14.2 Field Documentation

4.14.2.1 baud_rate `const u32 baud_rate`

Definition at line 55 of file UART_cfg.h.

4.14.2.2 clock_polarity `UART_CLOCK_POLARITY_t clock_polarity`

Definition at line 61 of file UART_cfg.h.

4.14.2.3 data_bits `UART_DATA_BITS_t data_bits`

Definition at line 56 of file UART_cfg.h.

4.14.2.4 mode `UART_MODE_t mode`

Definition at line 59 of file UART_cfg.h.

4.14.2.5 **mode_type** [UART_MODE_TYPE_t](#) mode_type

Definition at line 60 of file UART_cfg.h.

4.14.2.6 **parity** [UART_PARITY_t](#) parity

Definition at line 58 of file UART_cfg.h.

4.14.2.7 **stop_bits** [UART_STOP_BITS_t](#) stop_bits

Definition at line 57 of file UART_cfg.h.

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

- [code/include/UART_cfg.h](#)

4.15 **WHEELS_CONFIG_t** Struct Reference

```
#include <WHEELS_cfg.h>
```

Data Fields

- [PIN_t](#) ENA_pin
- [PORT_t](#) ENA_port
- [PIN_t](#) ENB_pin
- [PORT_t](#) ENB_port
- [PWM_t](#) IN1A_channel
- [PWM_t](#) IN2A_channel
- [PWM_t](#) IN1B_channel
- [PWM_t](#) IN2B_channel
- [PIN_t](#) CTA_pin
- [PORT_t](#) CTA_port
- [PIN_t](#) CTB_pin
- [PORT_t](#) CTB_port
- [u8](#) currentSensitivity
- [u8](#) SpeedPercentage
- [WHEELS_POSITION_t](#) WHEELS_Position

4.15.1 Detailed Description

Definition at line 12 of file WHEELS_cfg.h.

4.15.2 Field Documentation

4.15.2.1 CTA_pin `PIN_t CTA_pin`

Definition at line 23 of file WHEELS_cfg.h.

4.15.2.2 CTA_port `PORT_t CTA_port`

Definition at line 24 of file WHEELS_cfg.h.

4.15.2.3 CTB_pin `PIN_t CTB_pin`

Definition at line 25 of file WHEELS_cfg.h.

4.15.2.4 CTB_port `PORT_t CTB_port`

Definition at line 26 of file WHEELS_cfg.h.

4.15.2.5 currentSensitivity `u8 currentSensitivity`

Definition at line 28 of file WHEELS_cfg.h.

4.15.2.6 ENA_pin `PIN_t ENA_pin`

Definition at line 13 of file WHEELS_cfg.h.

4.15.2.7 ENA_port `PORT_t ENA_port`

Definition at line 14 of file WHEELS_cfg.h.

4.15.2.8 ENB_pin `PIN_t ENB_pin`

Definition at line 15 of file WHEELS_cfg.h.

4.15.2.9 ENB_port [PORT_t](#) ENB_port

Definition at line 16 of file WHEELS_cfg.h.

4.15.2.10 IN1A_channel [PWM_t](#) IN1A_channel

Definition at line 18 of file WHEELS_cfg.h.

4.15.2.11 IN1B_channel [PWM_t](#) IN1B_channel

Definition at line 20 of file WHEELS_cfg.h.

4.15.2.12 IN2A_channel [PWM_t](#) IN2A_channel

Definition at line 19 of file WHEELS_cfg.h.

4.15.2.13 IN2B_channel [PWM_t](#) IN2B_channel

Definition at line 21 of file WHEELS_cfg.h.

4.15.2.14 SpeedPercentage [u8](#) SpeedPercentage

Definition at line 29 of file WHEELS_cfg.h.

4.15.2.15 WHEELS_Position [WHEELS_POSITION_t](#) WHEELS_Position

Definition at line 30 of file WHEELS_cfg.h.

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

- [code/include/WHEELS_cfg.h](#)

5 File Documentation

5.1 [code/include/app.h](#) File Reference

Types and APIs for application app.c.

Data Structures

- struct [ORDER_INFO_t](#)
Information of an order.

Enumerations

- enum [NAVIGATION_MODE_t](#) { [AUTONOMOUS_MODE](#) , [MANUAL_MODE](#) }
Navigation modes of the robot.
- enum [ROBOT_STATE_t](#) { [ROBOT_AVAILABLE](#) , [ROBOT_PROCESSING_ORDER](#) , [ROBOT_CHARGING](#) , [ROBOT_OBSTACLE_AVOIDANCE](#) }
Different states of the robot.
- enum [ORDER_AVAILABILITY_t](#) { [NO_ORDER_RECEIVED](#) , [ORDER_RECEIVED](#) }
Availability of new orders.
- enum [ORDER_TRACKING_t](#) { [ORDER_PROCESSING](#) , [ORDER_OUT_FOR_DELIVER](#) , [ORDER_DELIVERED](#) , [ORDER_GOING_BACK](#) , [ORDER_RETURNED](#) }
Different states of the order.
- enum [CLIENT_PICKUP_t](#) { [CLIENT_PICKUP_NO](#) , [CLIENT_PICKUP_YES](#) }
Pickup state of the item. If the client has picked up the item or not.
- enum [ORIENTATION_t](#) { [NORTH](#) = 0 , [EAST](#) = 90 , [SOUTH](#) = 180 , [WEST](#) = 270 }
Orientation of the robot face.

Functions

- [NAVIGATION_MODE_t](#) [checkAutoOrManualNavigation](#) (void)
- void [autonomousMode](#) (void)
- void [manualNavigation](#) (void)
- void [monitorBattery](#) (void)
- void [handleOrder](#) ([ORDER_INFO_t](#) order)
- [ORDER_INFO_t](#) [getOrderInfo](#) (void)
- void [navigate](#) ([COLUMN_t](#) column, [ROW_t](#) row)
- void [navigateColumn](#) ([COLUMN_t](#) column)
- void [navigateRow](#) (const [ROW_t](#) row)
- void [modifyOrientation](#) (const [ORIENTATION_t](#) orientation)
- [STATE_t](#) [onTrack](#) (void)
- [STATE_t](#) [onLeftSmooth](#) (void)
- [STATE_t](#) [onLeftSharp](#) (void)
- [STATE_t](#) [onRightSmooth](#) (void)
- [STATE_t](#) [onRightSharp](#) (void)
- void [gotoNextJunction](#) (void)
- void [sendPosition](#) (const [COLUMN_t](#) column, const [ROW_t](#) row)
- [BOOL_t](#) [onJunction](#) (void)
- [BOOL_t](#) [onObstacle](#) (void)
- void [sendRobotState](#) (const [ROBOT_STATE_t](#) robotState)
- void [sendOrderTrackInfo](#) (const [ORDER_TRACKING_t](#) orderTrackInfo)
- void [waitForClientPickup](#) (void)
- [STATE_t](#) [MASTER_PickItem](#) (void)
- [u8](#) [readSerial](#) (void)
- void [avoidObstacle](#) (void)

5.1.1 Detailed Description

Types and APIs for application app.c.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.1.2 Enumeration Type Documentation

5.1.2.1 **CLIENT_PICKUP_t** enum `CLIENT_PICKUP_t`

Pickup state of the item. If the client has picked up the item or not.

The item can be in one of the following states:

- Picked up: The item has been picked up by the client.
- Not picked up: The item has not been picked up by the client.

Enumerator

CLIENT_PICKUP_NO	Not picked up state
CLIENT_PICKUP_YES	Picked up state

Definition at line 77 of file app.h.

5.1.2.2 **NAVIGATION_MODE_t** enum `NAVIGATION_MODE_t`

Navigation modes of the robot.

The robot can be in one of the following modes:

- Manual mode: The robot is controlled by the user either by the joystick or the mobile app.
- Autonomous mode: The robot moves autonomously by the sensors.

Enumerator

AUTONOMOUS_MODE	Autonomous mode
MANUAL_MODE	Manual mode

Definition at line 22 of file app.h.

5.1.2.3 ORDER_AVAILABILITY_t enum [ORDER_AVAILABILITY_t](#)

Availability of new orders.

The order can either be:

- Received: The order has been received and is ready to be processed.
- Not received: The order has not been received yet.

Enumerator

NO_ORDER_RECIEVED	No order received
ORDER_RECIEVED	Order received

Definition at line 49 of file app.h.

5.1.2.4 ORDER_TRACKING_t enum [ORDER_TRACKING_t](#)

Different states of the order.

The order can be in one of the following states:

- Processing: The order is being processed. The robot is on its way to the shelf to get the item.
- Out for delivery: The order is out for delivery. The robot has picked the shelf and is on its way to the client.
- Delivered: The order has been delivered to the client and is ready to be picked up.
- Going back: The client has picked the item, hence the robot will return the shelf to its original position.
- Returned: The robot has returned the shelf to its original position.

Enumerator

ORDER_PROCESSING	Processing state
ORDER_OUT_FOR_DELIVER	Out for delivery state
ORDER_DELIVERED	Delivered state
ORDER_GOING_BACK	Going back state
ORDER_RETURNED	Returned state

Definition at line 63 of file app.h.

5.1.2.5 ORIENTATION_t enum [ORIENTATION_t](#)

Orientation of the robot face.

The robot can be in one of the following orientations:

- North: The robot is facing north.
- East: The robot is facing east.
- South: The robot is facing south.
- West: The robot is facing west.

Enumerator

NORTH	
EAST	
SOUTH	
WEST	

Definition at line 106 of file app.h.

5.1.2.6 ROBOT_STATE_t enum [ROBOT_STATE_t](#)

Different states of the robot.

The robot can be in one of the following states:

- Available: The robot is available to receive commands.
- Processing: The robot is processing a command (e.g. processing an order to get an item). It is not available to receive commands until the current command is completed.
- Charging: The robot is charging. It is not available to receive commands until the charging is completed.
- Obstacle: The robot is in an obstacle. It is not available to receive commands until the obstacle is removed.

Enumerator

ROBOT_AVAILABLE	Available state
ROBOT_PROCESSING_ORDER	Processing state
ROBOT_CHARGING	Charging state
ROBOT_OBSTACLE_AVOIDANCE	Obstacle state

Definition at line 36 of file app.h.

5.1.3 Function Documentation

5.1.3.1 autonomousMode() `void autonomousMode (`
`void)`

5.1.3.2 avoidObstacle() `void avoidObstacle (`
`void)`

5.1.3.3 checkAutoOrManualNavigation() `NAVIGATION_MODE_t checkAutoOrManualNavigation (`
`void)`

5.1.3.4 getOrderInfo() `ORDER_INFO_t getOrderInfo (`
`void)`

5.1.3.5 gotoNextJunction() `void gotoNextJunction (`
`void)`

5.1.3.6 handleOrder() `void handleOrder (`
`ORDER_INFO_t order)`

5.1.3.7 manualNavigation() `void manualNavigation (`
 `void)`

5.1.3.8 MASTER_PickItem() `STATE_t MASTER_PickItem (`
 `void)`

5.1.3.9 modifyOrientation() `void modifyOrientation (`
 `const ORIENTATION_t orientation)`

5.1.3.10 monitorBattery() `void monitorBattery (`
 `void)`

5.1.3.11 navigate() `void navigate (`
 `COLUMN_t column,`
 `ROW_t row)`

5.1.3.12 navigateColumn() `void navigateColumn (`
 `COLUMN_t column)`

5.1.3.13 navigateRow() `void navigateRow (`
 `const ROW_t row)`

5.1.3.14 onJunction() `BOOL_t onJunction (`
 `void)`

5.1.3.15 onLeftSharp() `STATE_t onLeftSharp (`
 `void)`

5.1.3.16 onLeftSmooth() `STATE_t onLeftSmooth (`
`void)`

5.1.3.17 onObstacle() `BOOL_t onObstacle (`
`void)`

5.1.3.18 onRightSharp() `STATE_t onRightSharp (`
`void)`

5.1.3.19 onRightSmooth() `STATE_t onRightSmooth (`
`void)`

5.1.3.20 onTrack() `STATE_t onTrack (`
`void)`

5.1.3.21 readSerial() `u8 readSerial (`
`void)`

5.1.3.22 sendOrderTrackInfo() `void sendOrderTrackInfo (`
`const ORDER_TRACKING_t orderTrackInfo)`

5.1.3.23 sendPosition() `void sendPosition (`
`const COLUMN_t column,`
`const ROW_t row)`

5.1.3.24 sendRobotState() `void sendRobotState (`
`const ROBOT_STATE_t robotState)`

5.1.3.25 waitForClientPickup() `void waitForClientPickup (`
`void)`

5.2 code/include/app_cfg.h File Reference

Configuration header file for app.c.

Macros

- `#define NAVIGATION_MODE_BUTTON BUTTON_0`
- `#define ITEM_PICKUP_BUTTON BUTTON_1`
- `#define NEON_LED LED_0`
- `#define OBSTACLE_DISTANCE_THRESHOLD 100`
- `#define NUM_OF_COLUMNS 2`
- `#define NUM_OF_ROWS 2`

Enumerations

- enum `COLUMN_t` { `DEFAULT_COLUMN = 0` , `CLIENT_COLUMN = 0` , `CHARGER_COLUMN = 0` }
- enum `ROW_t` { `CLIENT_ROW = 0` , `DEFAULT_ROW = 1` , `CHARGER_ROW = 2` }

5.2.1 Detailed Description

Configuration header file for app.c.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.2.2 Macro Definition Documentation

5.2.2.1 ITEM_PICKUP_BUTTON `#define ITEM_PICKUP_BUTTON BUTTON_1`

Button used to pick up an item is connected to `BUTTON_1`

Definition at line 13 of file `app_cfg.h`.

5.2.2.2 NAVIGATION_MODE_BUTTON `#define NAVIGATION_MODE_BUTTON BUTTON_0`

Button used to navigate between modes is connected to BUTTON_0

Definition at line 12 of file app_cfg.h.

5.2.2.3 NEON_LED `#define NEON_LED LED_0`

LED_0 is connected to the NEON_LED

Definition at line 15 of file app_cfg.h.

5.2.2.4 NUM_OF_COLUMNS `#define NUM_OF_COLUMNS 2`

Number of columns on the arena

Definition at line 31 of file app_cfg.h.

5.2.2.5 NUM_OF_ROWS `#define NUM_OF_ROWS 2`

Number of rows on the arena

Definition at line 32 of file app_cfg.h.

5.2.2.6 OBSTACLE_DISTANCE_THRESHOLD `#define OBSTACLE_DISTANCE_THRESHOLD 100`

Distance threshold in cm for obstacle avoidance

Definition at line 29 of file app_cfg.h.

5.2.3 Enumeration Type Documentation

5.2.3.1 COLUMN_t `enum COLUMN_t`

Enumerator

DEFAULT_COLUMN	Default column of the robot at reset
CLIENT_COLUMN	Column of the client
CHARGER_COLUMN	Column of the charger

Definition at line 17 of file app_cfg.h.

5.2.3.2 ROW_t enum ROW_t

Enumerator

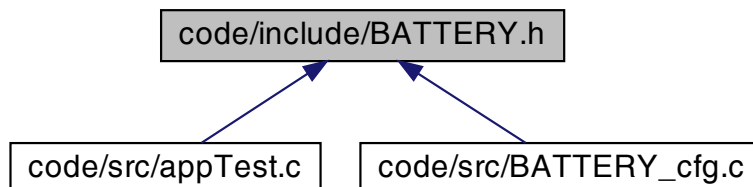
CLIENT_ROW	Row of the client
DEFAULT_ROW	Default row of the robot at reset
CHARGER_ROW	Row of the charger

Definition at line 23 of file app_cfg.h.

5.3 code/include/BATTERY.h File Reference

Interfaces header file for [BATTERY.c](#).

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



Functions

- void [BATTERY_Init](#) (void)
- [STATE_t BATTERY_GetState](#) (void)

5.3.1 Detailed Description

Interfaces header file for [BATTERY.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.3.2 Function Documentation

5.3.2.1 BATTERY_GetState() `STATE_t BATTERY_GetState (`
`void)`

5.3.2.2 BATTERY_Init() `void BATTERY_Init (`
`void)`

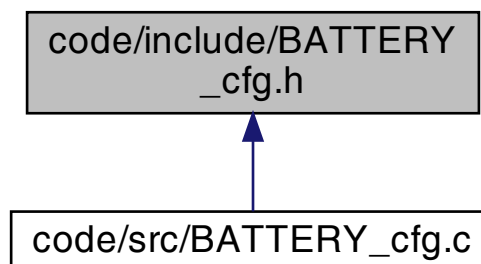
Here is the caller graph for this function:



5.4 code/include/BATTERY_cfg.h File Reference

Configuration header file for [BATTERY.c](#).

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



Data Structures

- struct [BatteryConfigs_t](#)

Macros

- `#define BATTERY_CAPACITY 18.0f /* 18 AH */`
- `#define BATTERY_VOLTAGE 12.0f /* 12 V */`
- `#define BATTERY_FULL_VOLTAGE 12.0f /* 12 V */`
- `#define BATTERY_EMPTY_VOLTAGE 6.0f /* 6 V */`
- `#define BATTERY_VOLTAGE_DIVIDER_RATIO (12 / 4.0f) /* 12V : 4V */`

Variables

- [BatteryConfigs_t](#) `BatteryConfigs`
Configurations for the Battery Management Module.

5.4.1 Detailed Description

Configuration header file for [BATTERY.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.4.2 Macro Definition Documentation

5.4.2.1 BATTERY_CAPACITY `#define BATTERY_CAPACITY 18.0f /* 18 AH */`

Definition at line 12 of file `BATTERY_cfg.h`.

5.4.2.2 BATTERY_EMPTY_VOLTAGE `#define BATTERY_EMPTY_VOLTAGE 6.0f /* 6 V */`

Definition at line 15 of file BATTERY_cfg.h.

5.4.2.3 BATTERY_FULL_VOLTAGE `#define BATTERY_FULL_VOLTAGE 12.0f /* 12 V */`

Definition at line 14 of file BATTERY_cfg.h.

5.4.2.4 BATTERY_VOLTAGE `#define BATTERY_VOLTAGE 12.0f /* 12 V */`

Definition at line 13 of file BATTERY_cfg.h.

5.4.2.5 BATTERY_VOLTAGE_DIVIDER_RATIO `#define BATTERY_VOLTAGE_DIVIDER_RATIO (12 / 4.0f) /* 12V : 4V */`

Definition at line 19 of file BATTERY_cfg.h.

5.4.3 Variable Documentation

5.4.3.1 BatteryConfigs `BatteryConfigs_t BatteryConfigs [extern]`

Configurations for the Battery Management Module.

This is the configuration for the Battery Management Module. The configuration is done by the user. The user can change the configuration according to his needs.

Note

The configuration is done in the [BATTERY_cfg.h](#) and [BATTERY_cfg.c](#) files.

Definition at line 21 of file BATTERY_cfg.c.

5.5 code/include/BIT_MATH.h File Reference

Common bit manipulation operations.

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



Macros

- `#define GET_BIT(REGISTER, BIT) (1 & ((REGISTER) >> (BIT)))`
Read state of a specific bit.
- `#define SET_BIT(REGISTER, BIT) ((REGISTER) |= (1 << (BIT)))`
Set state of a specific bit (set to 1)
- `#define CLR_BIT(REGISTER, BIT) ((REGISTER) &= ~(1 << (BIT)))`
Clear state of a specific bit (set to 0)
- `#define TOG_BIT(REGISTER, BIT) ((REGISTER) ^= (1 << (BIT)))`
Toggle state of a specific bit (set to 0)
- `#define BIT_IS_SET(REGISTER, Bit) ((REGISTER) & (1 << (Bit)))`
Check if state of a specific bit is set (state = 1)
- `#define BIT_IS_CLEAR(REGISTER, Bit) (!((REGISTER) & (1 << (Bit))))`
Check if state of a specific bit is Cleared (state = 0)
- `#define CONCAT_8BITS(b7, b6, b5, b4, b3, b2, b1, b0) (0b##b7##b6##b5##b4##b3##b2##b1##b0)`
- `#define CONCAT_7BITS(b6, b5, b4, b3, b2, b1, b0) (0b##b6##b5##b4##b3##b2##b1##b0)`
- `#define CONCAT_6BITS(b5, b4, b3, b2, b1, b0) (0b##b5##b4##b3##b2##b1##b0)`
- `#define CONCAT_5BITS(b4, b3, b2, b1, b0) (0b##b4##b3##b2##b1##b0)`
- `#define CONCAT_4BITS(b3, b2, b1, b0) (0b##b3##b2##b1##b0)`
- `#define CONCAT_3BITS(b2, b1, b0) (0b##b2##b1##b0)`
- `#define CONCAT_2BITS(b1, b0) (0b##b1##b0)`

5.5.1 Detailed Description

Common bit manipulation operations.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2021-07-31

5.5.2 Macro Definition Documentation

5.5.2.1 BIT_IS_CLEAR `#define BIT_IS_CLEAR(
REGISTER,
Bit) (!((REGISTER) & (1 << (Bit))))`

Check if state of a specific bit is Cleared (state = 0)

Parameters

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be set

Returns

1 or 0: 1 if the bit is cleared, 0 if the bit is set

For example:

`BIT_IS_CLEAR(PORT_A, PIN0)` will return 1 if bit 0 of PORT_A is LOW or 0 if it is HIGH

Definition at line 67 of file BIT_MATH.h.

```
5.5.2.2 BIT_IS_SET #define BIT_IS_SET(  
    REGISTER,  
    Bit ) ( (REGISTER) & (1 << (Bit)) )
```

Check if state of a specific bit is set (state = 1)

Parameters

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be set

Returns

1 or 0: 1 if the bit is set, 0 if the bit is cleared

For example:

`BIT_IS_SET(PORT_A, PIN0)` will return 1 if bit 0 of PORT_A is HIGH or 0 if it is LOW

Definition at line 56 of file BIT_MATH.h.

```
5.5.2.3 CLR_BIT #define CLR_BIT(  
    REGISTER,  
    BIT ) ( (REGISTER) &= ~(1 << (BIT)) )
```

Clear state of a specific bit (set to 0)

Parameters

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be cleared

For example:

CLEAR_BIT(PORT_A, PIN0) will set bit 0 of PORT_A to LOW (0)

Definition at line 37 of file BIT_MATH.h.

5.5.2.4 CONCAT_2BITS `#define CONCAT_2BITS(
 b1,
 b0) (0b##b1##b0)`

Definition at line 75 of file BIT_MATH.h.

5.5.2.5 CONCAT_3BITS `#define CONCAT_3BITS(
 b2,
 b1,
 b0) (0b##b2##b1##b0)`

Definition at line 74 of file BIT_MATH.h.

5.5.2.6 CONCAT_4BITS `#define CONCAT_4BITS(
 b3,
 b2,
 b1,
 b0) (0b##b3##b2##b1##b0)`

Definition at line 73 of file BIT_MATH.h.

5.5.2.7 CONCAT_5BITS `#define CONCAT_5BITS(
 b4,
 b3,
 b2,
 b1,
 b0) (0b##b4##b3##b2##b1##b0)`

Definition at line 72 of file BIT_MATH.h.

5.5.2.8 CONCAT_6BITS `#define CONCAT_6BITS(
 b5,
 b4,
 b3,
 b2,
 b1,
 b0) (0b##b5##b4##b3##b2##b1##b0)`

Definition at line 71 of file BIT_MATH.h.

5.5.2.9 CONCAT_7BITS `#define CONCAT_7BITS(`
`b6,`
`b5,`
`b4,`
`b3,`
`b2,`
`b1,`
`b0) (0b##b6##b5##b4##b3##b2##b1##b0)`

Definition at line 70 of file BIT_MATH.h.

5.5.2.10 CONCAT_8BITS `#define CONCAT_8BITS(`
`b7,`
`b6,`
`b5,`
`b4,`
`b3,`
`b2,`
`b1,`
`b0) (0b##b7##b6##b5##b4##b3##b2##b1##b0)`

Definition at line 69 of file BIT_MATH.h.

5.5.2.11 GET_BIT `#define GET_BIT(`
`REGISTER,`
`BIT) (1 & ((REGISTER) >> (BIT)))`

Read state of a specific bit.

Parameters

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be read

Returns

state of the bit: 1 or 0

For example:

`GET_BIT(PORT_A, PIN0)` will return 1 if bit 0 of PORT_A is HIGH or 0 if it is LOW

Definition at line 19 of file BIT_MATH.h.

5.5.2.12 SET_BIT `#define SET_BIT(`
`REGISTER,`
`BIT) ((REGISTER) |= (1 << (BIT)))`

Set state of a specific bit (set to 1)

Parameters

in	<i>REGISTER</i>	the register includes the bit
in	<i>BIT</i>	the required bit number to be set

For example:

`SET_BIT(PORT_A, PIN0)` will set bit 0 of PORT_A to HIGH (1)

Definition at line 28 of file BIT_MATH.h.

5.5.2.13 TOG_BIT `#define TOG_BIT(
REGISTER,
BIT) ((REGISTER) ^= (1 << (BIT)))`

Toggle state of a specific bit (set to 0)

Parameters

in	<i>REGISTER</i>	is the register includes the bit
in	<i>BIT</i>	the required bit number to be toggled

For example:

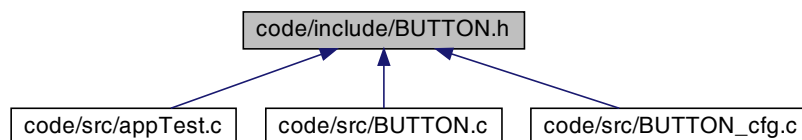
`TOG_BIT(PORT_A, PIN0)` will toggle bit 0 of PORT_A. So if it was HIGH, it will be LOW, and if it was LOW, it will be HIGH.

Definition at line 46 of file BIT_MATH.h.

5.6 code/include/BUTTON.h File Reference

Interfaces header file for `BUTTON.c`.

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

**Enumerations**

- enum `BUTTON_t` {
`BUTTON_0` , `BUTTON_1` , `BUTTON_2` , `BUTTON_3` ,
`BUTTON_4` , `BUTTON_5` , `BUTTON_6` , `BUTTON_7` }

Functions

- void `BUTTON_Init` (void)
Initializes Buttons connected to DIO.
- `STATE_t BUTTON_GetStatus` (`BUTTON_t` button)
Check whether a specific button is pressed or not.

5.6.1 Detailed Description

Interfaces header file for `BUTTON.c`.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.6.2 Enumeration Type Documentation

5.6.2.1 `BUTTON_t` enum `BUTTON_t`

Enumerator

<code>BUTTON_0</code>	
<code>BUTTON_1</code>	
<code>BUTTON_2</code>	
<code>BUTTON_3</code>	
<code>BUTTON_4</code>	
<code>BUTTON_5</code>	
<code>BUTTON_6</code>	
<code>BUTTON_7</code>	

Definition at line 15 of file BUTTON.h.

5.6.3 Function Documentation

5.6.3.1 **BUTTON_GetStatus()** `STATE_t BUTTON_GetStatus (BUTTON_t button)`

Check whether a specific button is pressed or not.

Parameters

in	<i>button</i>	Button number: options from <code>BUTTON_t</code> enum in <code>BUTTON.h</code> file
----	---------------	--

Returns

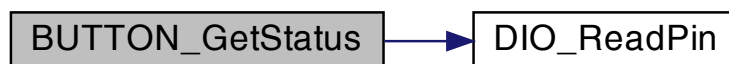
State of the button: `HIGH` if pressed and `LOW` if not, member of `STATE_t` enum

For example:

```
BUTTON_GetStatus(BUTTON_0); //returns \ref HIGH if button 0 is pressed and \ref LOW otherwise
```

Definition at line 18 of file BUTTON.c.

Here is the call graph for this function:



5.6.3.2 **BUTTON_Init()** `void BUTTON_Init (void)`

Initializes Buttons connected to DIO.

Definition at line 14 of file BUTTON.c.

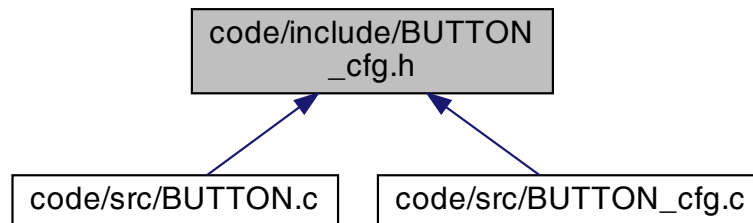
Here is the caller graph for this function:



5.7 code/include/BUTTON_cfg.h File Reference

Configuration header file for [BUTTON.c](#).

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



Data Structures

- struct [BUTTON_CONFIGS_t](#)

Enumerations

- enum [DEBOUNCE_t](#) { [DEBOUNCE_OFF](#) , [DEBOUNCE_ON](#) }

Variables

- [BUTTON_CONFIGS_t](#) buttonsConfigs []
- const [u8](#) countButtonsConfigured

5.7.1 Detailed Description

Configuration header file for [BUTTON.c](#).

Configuration source file for [BUTTON.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.7.2 Enumeration Type Documentation

5.7.2.1 [DEBOUNCE_t](#) enum [DEBOUNCE_t](#)

Definition at line 12 of file BUTTON_cfg.h.

5.7.3.1 buttonsConfigs `BUTTON_CONFIGS_t` buttonsConfigs[] [extern]

ACTIVE_LOW means that the pin is:

- Definition at line 23 of file BUTTON_cfg.c.

5.7.3.2 countButtonsConfigured

```
const u8 countButtonsConfigured [extern]
```

Definition at line 28 of file BUTTON_cfg.c.

5.8 code/include/DIO.h File Reference

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



- enum PIN_t {
PIN_0, PIN_1, PIN_2, PIN_3,
PIN_4, PIN_5, PIN_6, PIN_7 }
- enum PORT_t {
PORT_A, PORT_B, PORT_C, PORT_D,
PORT_E, PORT_F, PORT_G }
- enum DIR_t { INPUT, OUTPUT }
- enum PULLUP_t { PULLUP_TRUE, PULLUP_FALSE }

Functions

- void `DIO_Init` ()
Initialize DIO configurations based on user configurations in `DIO_cfg.h` and `DIO_cfg.c`.
- void `DIO_InitPin` (`PIN_t` pin, `PORT_t` port, `DIR_t` direction, `PULLUP_t` pullup)
Initialize a pin as input or output.
- void `DIO_WritePin` (const `PIN_t` pin, const `PORT_t` port, const `STATE_t` pinState)
write a value on the output pins, options are defined in `STD_TYPES.h` in the enum `STATE_t`
- void `DIO_WritePort` (const `PORT_t` port, const `u8` value)
write a value on a specific port (value of 8-bits ranges from 0 to 255)
- `STATE_t` `DIO_ReadPin` (const `PIN_t` pin, const `PORT_t` port)
Read the state of a pin.
- `u8` `DIO_ReadPort` (const `PORT_t` port)
Read the state of the port (8 bits --> 0-255)

5.8.1 Enumeration Type Documentation

5.8.1.1 `DIR_t` enum `DIR_t`

Enumerator

INPUT	
OUTPUT	

Definition at line 36 of file DIO.h.

5.8.1.2 `PIN_t` enum `PIN_t`

Enumerator

<code>PIN↔</code> _0	
<code>PIN↔</code> _1	
<code>PIN↔</code> _2	
<code>PIN↔</code> _3	
<code>PIN↔</code> _4	
<code>PIN↔</code> _5	
<code>PIN↔</code> _6	
<code>PIN↔</code> _7	

Definition at line 15 of file DIO.h.

5.8.1.3 **PORT_t** enum [PORT_t](#)

Enumerator

PORT_A	
PORT_B	
PORT_C	
PORT_D	
PORT_E	
PORT_F	
PORT_G	

Definition at line 26 of file DIO.h.

5.8.1.4 **PULLUP_t** enum [PULLUP_t](#)

Enumerator

PULLUP_TRUE	
PULLUP_FALSE	

Definition at line 41 of file DIO.h.

5.8.2 **Function Documentation**

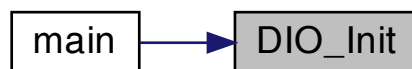
5.8.2.1 **DIO_Init()** void [DIO_Init](#) (void)

Initialize DIO configurations based on user configurations in [DIO_cfg.h](#) and [DIO_cfg.c](#).

Initialize DIO pins to a specific direction (input or output), pullup or not according to the configuration in the [DIO_cfg.h](#) file.

Definition at line 67 of file DIO.c.

Here is the caller graph for this function:



5.8.2.2 DIO_InitPin() void DIO_InitPin (
 const [PIN_t](#) pin,
 const [PORT_t](#) port,
 const [DIR_t](#) direction,
 const [PULLUP_t](#) pullup)

Initialize a pin as input or output.

Parameters

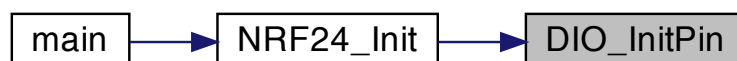
in	<i>pin</i>	The pin to be initialized: PIN0 ... PIN7
in	<i>port</i>	The port of the pin: PORT_A ... PORT_G.
in	<i>direction</i>	The direction of the pin: INPUT or OUTPUT.
in	<i>pullup</i>	The pullup of the pin: PULLUP_TRUE or PULLUP_FALSE.

If:

- DIO pin is not configured in the [DIO_cfg.h](#) file
- Or the pin is configurations need to be modified Then, this function will be called to modify the pin configuration.

Definition at line 110 of file DIO.c.

Here is the caller graph for this function:



5.8.2.3 DIO_ReadPin() `STATE_t DIO_ReadPin (`
`const PIN_t pin,`
`const PORT_t port)`

Read the state of a pin.

Parameters

in	<i>pin</i>	the pin to be initialized, PIN0, PIN1, ..., PIN7
in	<i>port</i>	the port of the pin to be initialized, PORT_A, PORT_B, ..., PORT_G

Returns

state of the pin, HIGH or LOW

Example:

```
DIO_ReadPin(PIN0, PORT_A); // return the state of PIN0 on PORT A
DIO_ReadPin(PIN6, PORT_B); // return the state of PIN6 on PORT B
```

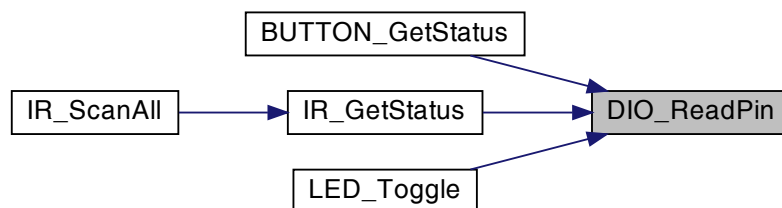
Warning

If the pin is not configured in the [DIO_cfg.h](#) file, then this function will do nothing.

If the pin is configured in the [DIO_cfg.h](#) file, then this function can be called to read the pin (1 or 0).

Definition at line 273 of file DIO.c.

Here is the caller graph for this function:



5.8.2.4 DIO_ReadPort() `u8 DIO_ReadPort (`
`const PORT_t port)`

Read the state of the port (8 bits --> 0-255)

Parameters

in	<i>port</i>	the port to be read, PORT_A, PORT_B, ..., PORT_G
----	-------------	--

Returns

value of the port (8 bits --> 0-255)

Example:

```
DIO_ReadPort(PORT_A); // return 0xFF when all pins of PORT A are high

DIO_ReadPort(PORT_A); // return 0x00 when all pins of PORT A are low
DIO_ReadPort(PORT_A); // return 0xF0 when pins (4~7) of PORT A are high
```

Warning

If the pin is not configured in the [DIO_cfg.h](#) file, then this function return LOW.

Read the value of the port (all pins).

Definition at line 321 of file DIO.c.

5.8.2.5 DIO_WritePin() void DIO_WritePin (
const [PIN_t](#) pin,
const [PORT_t](#) port,
const [STATE_t](#) pinState)

write a value on the output pins, options are defined in [STD_TYPES.h](#) in the enum STATE_t

Parameters

in	<i>pin</i>	the pin to be initialized, PIN0, PIN1, ..., PIN7
in	<i>port</i>	the port of the pin to be initialized, PORT_A, PORT_B, ..., PORT_G
in	<i>pinState</i>	state of the pin, high or low, options are defined in STD_TYPES.h in the enum STATE_t

Example:

```
DIO_WritePin(PIN0, PORT_A, HIGH); // turns on pin 0 of port A
DIO_WritePin(PIN0, PORT_A, LOW);  // turns off pin 0 of port A
```

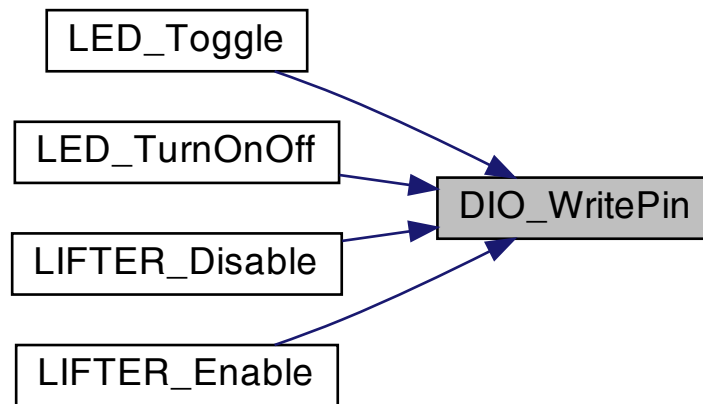
Warning

If the pin is not configured in the [DIO_cfg.h](#) file, then this function will do nothing.

If the pin is configured in the [DIO_cfg.h](#) file, then this function can be called to write to the pin (1 or 0).

Definition at line 149 of file DIO.c.

Here is the caller graph for this function:



5.8.2.6 DIO_WritePort() `void DIO_WritePort (`
 `const PORT_t port,`
 `const u8 value)`

write a value on a specific port (value of 8-bits ranges from 0 to 255)

Parameters

in	<i>port</i>	The port to be modified, PORT_A, PORT_B, ..., PORT_G
in	<i>value</i>	value to set on the port (8 bits --> 0-255)

Example:

```
DIO_WritePort(PORT_A, 0xFF); // sets all pins of port A to high
DIO_WritePort(PORT_A, 0x00); // sets all pins of port A to low
DIO_WritePort(PORT_A, 0xF0); // sets pins 4, 5, 6, 7 of port A to high
```

Warning

If the pin is not configured in the [DIO_cfg.h](#) file, then this function will do nothing.

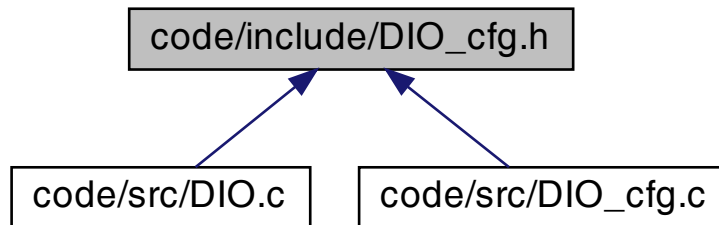
Write a value to the port (all pins).

Definition at line 234 of file DIO.c.

5.9 code/include/DIO_cfg.h File Reference

Configuration header file for [DIO.c](#).

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



Data Structures

- struct [PinConfig_t](#)

Variables

- [PinConfig_t](#) pinConfigs []
- const [u8](#) countPinsConfigured

5.9.1 Detailed Description

Configuration header file for [DIO.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.9.2 Variable Documentation

5.9.2.1 countPinsConfigured `const u8 countPinsConfigured [extern]`

Definition at line 53 of file DIO_cfg.c.

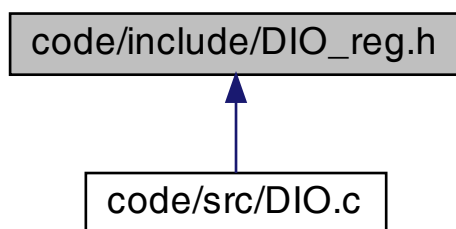
5.9.2.2 pinConfigs `PinConfig_t pinConfigs[] [extern]`

Definition at line 14 of file DIO_cfg.c.

5.10 code/include/DIO_reg.h File Reference

DIO Registers of ATmega328p microcontroller.

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



Macros

- #define PINA (* ((volatile u8 *) 0x39))
- #define DDRA (* ((volatile u8 *) 0x3A))
- #define PORTA (* ((volatile u8 *) 0x3B))
- #define PINB (* ((volatile u8 *) 0x36))
- #define DDRB (* ((volatile u8 *) 0x37))
- #define PORTB (* ((volatile u8 *) 0x38))
- #define PINC (* ((volatile u8 *) 0x33))
- #define DDRC (* ((volatile u8 *) 0x34))
- #define PORTC (* ((volatile u8 *) 0x35))
- #define PIND (* ((volatile u8 *) 0x30))
- #define DDRD (* ((volatile u8 *) 0x31))
- #define PORTD (* ((volatile u8 *) 0x32))
- #define PINE (* ((volatile u8 *) 0x21))
- #define DDRE (* ((volatile u8 *) 0x22))
- #define PORTE (* ((volatile u8 *) 0x23))
- #define PINF (* ((volatile u8 *) 0x20))
- #define DDRF (* ((volatile u8 *) 0x61))
- #define PORTF (* ((volatile u8 *) 0x62))
- #define PING (* ((volatile u8 *) 0x63))
- #define DDRG (* ((volatile u8 *) 0x64))
- #define PORTG (* ((volatile u8 *) 0x65))

5.10.1 Detailed Description

DIO Registers of ATmega328p microcontroller.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2021-07-31

5.10.2 Macro Definition Documentation

5.10.2.1 DDRA `#define DDRA (* ((volatile u8 *) 0x3A))`

Port A Data Direction Register

Definition at line 12 of file DIO_reg.h.

5.10.2.2 DDRB `#define DDRB (* ((volatile u8 *) 0x37))`

Port B Data Direction Register

Definition at line 16 of file DIO_reg.h.

5.10.2.3 DDRC `#define DDRC (* ((volatile u8 *) 0x34))`

Port C Data Direction Register

Definition at line 20 of file DIO_reg.h.

5.10.2.4 DDRD `#define DDRD (* ((volatile u8 *) 0x31))`

Port D Data Direction Register

Definition at line 24 of file DIO_reg.h.

5.10.2.5 DDRE `#define DDRE (* ((volatile u8 *) 0x22))`

Port E Data Direction Register

Definition at line 28 of file DIO_reg.h.

5.10.2.6 DDRF `#define DDRF (* ((volatile u8 *) 0x61))`

Port F Data Direction Register

Definition at line 32 of file DIO_reg.h.

5.10.2.7 DDRG `#define DDRG (* ((volatile u8 *) 0x64))`

Port G Data Direction Register

Definition at line 36 of file DIO_reg.h.

5.10.2.8 PINA `#define PINA (* ((volatile u8 *) 0x39))`

Port A Input Pins

Definition at line 11 of file DIO_reg.h.

5.10.2.9 PINB `#define PINB (* ((volatile u8 *) 0x36))`

Port B Input Pins

Definition at line 15 of file DIO_reg.h.

5.10.2.10 PINC `#define PINC (* ((volatile u8 *) 0x33))`

Port C Input Pins

Definition at line 19 of file DIO_reg.h.

5.10.2.11 PIND `#define PIND (* ((volatile u8 *) 0x30))`

Port D Input Pins

Definition at line 23 of file DIO_reg.h.

5.10.2.12 PINE `#define PINE (* ((volatile u8 *) 0x21))`

Port E Input Pins

Definition at line 27 of file DIO_reg.h.

5.10.2.13 PINF `#define PINF (* ((volatile u8 *) 0x20))`

Port F Input Pins

Definition at line 31 of file DIO_reg.h.

5.10.2.14 PING `#define PING (* ((volatile u8 *) 0x63))`

Port G Input Pins

Definition at line 35 of file DIO_reg.h.

5.10.2.15 PORTA `#define PORTA (* ((volatile u8 *) 0x3B))`

Port A Data Register

Definition at line 13 of file DIO_reg.h.

5.10.2.16 PORTB `#define PORTB (* ((volatile u8 *) 0x38))`

Port B Data Register

Definition at line 17 of file DIO_reg.h.

5.10.2.17 PORTC `#define PORTC (* ((volatile u8 *) 0x35))`

Port C Data Register

Definition at line 21 of file DIO_reg.h.

5.10.2.18 PORTD `#define PORTD (* ((volatile u8 *) 0x32))`

Port D Data Register

Definition at line 25 of file DIO_reg.h.

5.10.2.19 PORTE `#define PORTE (* ((volatile u8 *) 0x23))`

Port E Data Register

Definition at line 29 of file DIO_reg.h.

5.10.2.20 PORTF `#define PORTF (* ((volatile u8 *) 0x62))`

Port F Data Register

Definition at line 33 of file DIO_reg.h.

5.10.2.21 PORTG `#define PORTG (* ((volatile u8 *) 0x65))`

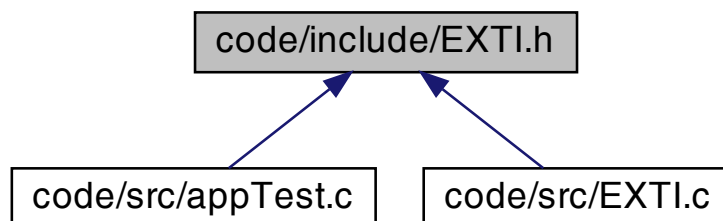
Port G Data Register

Definition at line 37 of file DIO_reg.h.

5.11 code/include/EXTI.h File Reference

Interfaces header file for [EXTI.c](#).

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



Enumerations

- enum `EXTI_SENSITIVITY_t` { `FALLING_EDGE` , `RISING_EDGE` , `LOW_LEVEL_DETECT` , `LOGIC_CHANGE` }
- enum `EXTI_t` { `EXTI_0` , `EXTI_1` , `EXTI_2` , `EXTI_3` , `EXTI_4` , `EXTI_5` , `EXTI_6` , `EXTI_7` }

Functions

- void `EXTI_Init` (const `EXTI_t` extiNumber, const `EXTI_SENSITIVITY_t` sensitivity, void(*const callback↵ Ptr)(void))
Initialize an EXTI pin with a given sensitivity and callback function.
- void `EXTI_EnableExternalInterrupt` (const `EXTI_t` extiNumber)
Enable an EXTI pin (EXTI0 - EXTI7)
- void `EXTI_DisableExternalInterrupt` (const `EXTI_t` extiNumber)
Disable an EXTI pin (EXTI0 - EXTI7)

5.11.1 Detailed Description

Interfaces header file for `EXTI.c`.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.11.2 Enumeration Type Documentation

5.11.2.1 `EXTI_SENSITIVITY_t` enum `EXTI_SENSITIVITY_t`

Enumerator

<code>FALLING_EDGE</code>	
<code>RISING_EDGE</code>	
<code>LOW_LEVEL_DETECT</code>	
<code>LOGIC_CHANGE</code>	

Definition at line 15 of file EXTI.h.

5.11.2.2 EXTI_t enum EXTI_t

Enumerator

EXTI↔ _0	
EXTI↔ _1	
EXTI↔ _2	
EXTI↔ _3	
EXTI↔ _4	
EXTI↔ _5	
EXTI↔ _6	
EXTI↔ _7	

Definition at line 22 of file EXTI.h.

5.11.3 Function Documentation

5.11.3.1 EXTI_DisableExternalInterrupt() `void EXTI_DisableExternalInterrupt (`
 `const EXTI_t extiNumber)`

Disable an EXTI pin (EXTI0 - EXTI7)

Parameters

in	<i>extiNumber</i>	The EXTI pin to disable (EXTI0 - EXTI7)
----	-------------------	---

Definition at line 501 of file EXTI.c.

5.11.3.2 EXTI_EnableExternalInterrupt() `void EXTI_EnableExternalInterrupt (`
 `const EXTI_t extiNumber)`

Enable an EXTI pin (EXTI0 - EXTI7)

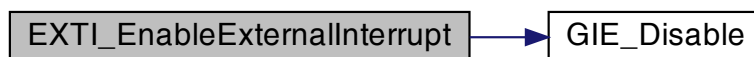
Parameters

in	<i>extiNumber</i>	The EXTI pin to enable (EXTI0 - EXTI7)
----	-------------------	--

Enable external interrupt pin

Definition at line 453 of file EXTI.c.

Here is the call graph for this function:



5.11.3.3 EXTI_Init() void EXTI_Init (

```

const EXTI_t extiNumber,
const EXTI_SENSITIVITY_t sensitivity,
void(*) (void) callbackPtr )

```

Initialize an EXTI pin with a given sensitivity and callback function.

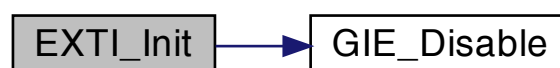
Parameters

in	<i>extiNumber</i>	The EXTI pin to initialize (EXTI0 - EXTI7)
in	<i>sensitivity</i>	The sensitivity of the EXTI pin (FALLING_EDGE, RISING_EDGE, LOW_LEVEL_DETECT, LOGIC_CHANGE)
in	<i>callbackPtr</i>	The callback function to be called when the EXTI pin is triggered

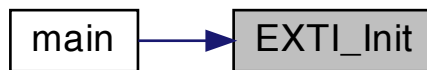
Initialize an external interrupt pin as input pin and set the

Definition at line 434 of file EXTI.c.

Here is the call graph for this function:



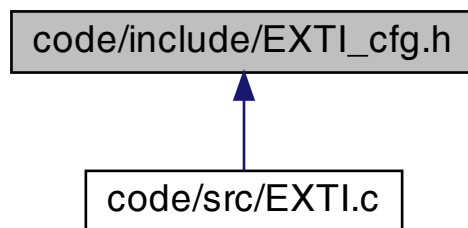
Here is the caller graph for this function:



5.12 code/include/EXTI_cfg.h File Reference

Configuration header file for [EXTI.c](#).

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



Macros

- `#define NESTING_NESTING_ENABLED`
Determine if nesting is allowed in the interrupt service routines. Options are: NESTING_DISABLED --> nesting is not allowed NESTING_ENABLED --> nesting is allowed.
- `#define NESTING_ENABLED 1`
- `#define NESTING_DISABLED 0`

5.12.1 Detailed Description

Configuration header file for [EXTI.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.12.2 Macro Definition Documentation**5.12.2.1 NESTING** `#define NESTING NESTING_ENABLED`

Determine if nesting is allowed in the interrupt service routines. Options are: NESTING_DISABLED --> nesting is not allowed NESTING_ENABLED --> nesting is allowed.

Definition at line 18 of file EXTI_cfg.h.

5.12.2.2 NESTING_DISABLED `#define NESTING_DISABLED 0`

Definition at line 25 of file EXTI_cfg.h.

5.12.2.3 NESTING_ENABLED `#define NESTING_ENABLED 1`

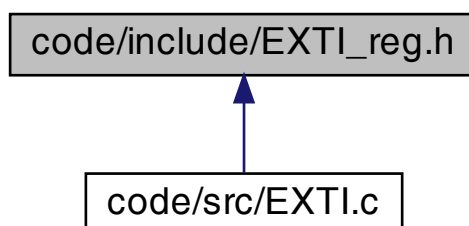
DO NOT CHANGE ANYTHING BELOW

Definition at line 24 of file EXTI_cfg.h.

5.13 code/include/EXTI_reg.h File Reference

EXTI Registers of ATmega328p MCU.

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



Macros

- `#define MCUCR (* ((volatile u8 *) 0x55)) /* MCU Control Register (Used with bootloader only) */`
- `#define EICRA (* ((volatile u8 *) 0x6A)) /* External Interrupt Control Register A --> INT3:0 mode: falling, rising, or Low Level */`
- `#define EICRB (* ((volatile u8 *) 0x5B)) /* External Interrupt Control Register B --> INT7:4 mode: falling, rising, or Low Level */`
- `#define EIMSK (* ((volatile u8 *) 0x59)) /* External Interrupt Mask Register: Enable or Disable External Interrupts */`
- `#define EIFR (* ((volatile u8 *) 0x58)) /* External Interrupt Flag Register: Clear External Interrupts Flag */`

Enumerations

- `enum {
 IVCE , IVSEL , SM2 , SM0 ,
 SM1 , SE , SRW10 , SRE }`
- `enum {
 ISC00 , ISC01 , ISC10 , ISC11 ,
 ISC20 , ISC21 , ISC30 , ISC31 }`
- `enum {
 ISC40 , ISC41 , ISC50 , ISC51 ,
 ISC60 , ISC61 , ISC70 , ISC71 }`
- `enum {
 INT0 , INT1 , INT2 , INT3 ,
 INT4 , INT5 , INT6 , INT7 }`
- `enum {
 INTF0 , INTF1 , INTF2 , INTF3 ,
 INTF4 , INTF5 , INTF6 , INTF7 }`

5.13.1 Detailed Description

EXTI Registers of ATmega328p MCU.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2021-07-31

5.13.2 Macro Definition Documentation

5.13.2.1 EICRA `#define EICRA (* ((volatile u8 *) 0x6A)) /* External Interrupt Control Register A --> INT3:0 mode: falling, rising, or Low Level */`

Definition at line 12 of file EXTI_reg.h.

5.13.2.2 EICRB `#define EICRB (* ((volatile u8 *) 0x5B)) /* External Interrupt Control Register B --> INT7:4 mode: falling, rising, or Low Level */`

Definition at line 13 of file EXTI_reg.h.

5.13.2.3 EIFR `#define EIFR (* ((volatile u8 *) 0x58)) /* External Interrupt Flag Register↔ : Clear External Interrupts Flag */`

Definition at line 15 of file EXTI_reg.h.

5.13.2.4 EIMSK `#define EIMSK (* ((volatile u8 *) 0x59)) /* External Interrupt Mask Register: Enable or Disable External Interrupts */`

Definition at line 14 of file EXTI_reg.h.

5.13.2.5 MCUCR `#define MCUCR (* ((volatile u8 *) 0x55)) /* MCU Control Register (Used with bootloader only) */`

Definition at line 11 of file EXTI_reg.h.

5.13.3 Enumeration Type Documentation

5.13.3.1 anonymous enum anonymous enum

Enumerator

IVCE	
IVSEL	
SM2	
SM0	
SM1	
SE	
SRW10	
SRE	

Definition at line 17 of file EXTI_reg.h.

5.13.3.2 anonymous enum `anonymous_enum`

Enumerator

ISC00	
ISC01	
ISC10	
ISC11	
ISC20	
ISC21	
ISC30	
ISC31	

Definition at line 28 of file EXTI_reg.h.

5.13.3.3 anonymous enum `anonymous_enum`

Enumerator

ISC40	
ISC41	
ISC50	
ISC51	
ISC60	
ISC61	
ISC70	
ISC71	

Definition at line 39 of file EXTI_reg.h.

5.13.3.4 anonymous enum `anonymous_enum`

Enumerator

INT0	
INT1	
INT2	
INT3	
INT4	
INT5	
INT6	
INT7	

Definition at line 50 of file EXTI_reg.h.

5.13.3.5 anonymous enum `anonymous enum`

Enumerator

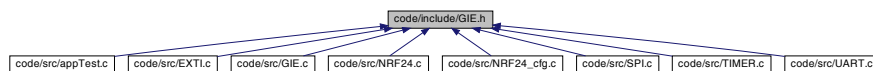
INTF0	
INTF1	
INTF2	
INTF3	
INTF4	
INTF5	
INTF6	
INTF7	

Definition at line 61 of file EXTI_reg.h.

5.14 code/include/GIE.h File Reference

Interfaces header file for [GIE.c](#).

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



Functions

- void [GIE_Enable](#) (void)
Global Interrupt Enable (GIE)
- void [GIE_Disable](#) (void)
Global Interrupt Disable (GID)

5.14.1 Detailed Description

Interfaces header file for [GIE.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

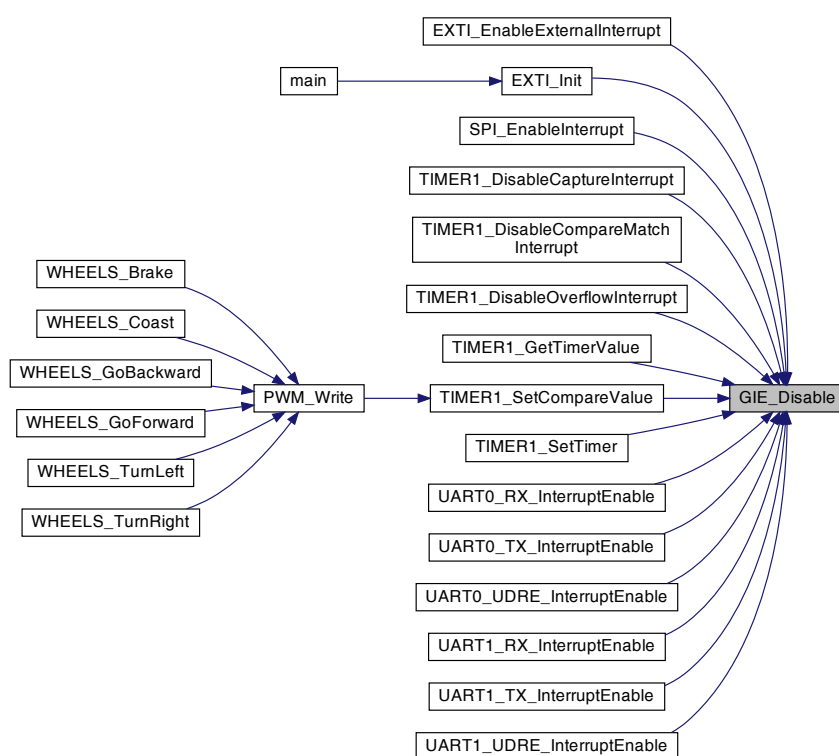
5.14.2 Function Documentation

5.14.2.1 GIE_Disable() `void GIE_Disable (void)`

Global Interrupt Disable (GID)

Definition at line 14 of file GIE.c.

Here is the caller graph for this function:

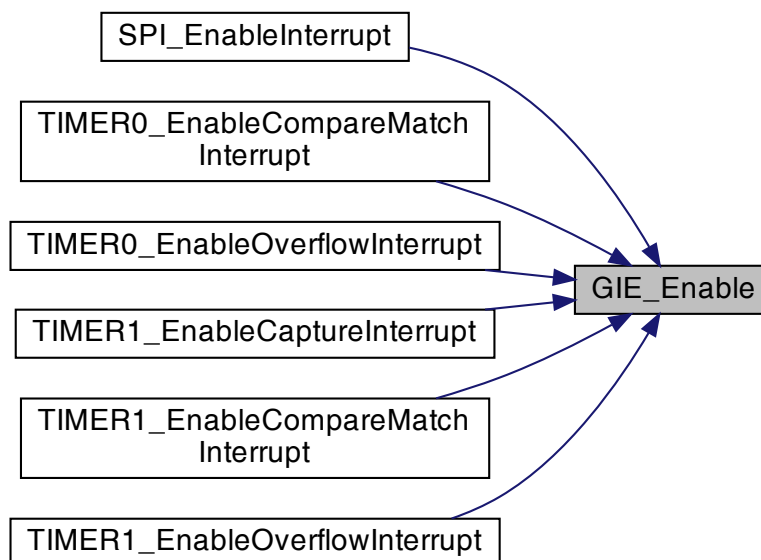


5.14.2.2 GIE_Enable() `void GIE_Enable (void)`

Global Interrupt Enable (GIE)

Definition at line 18 of file GIE.c.

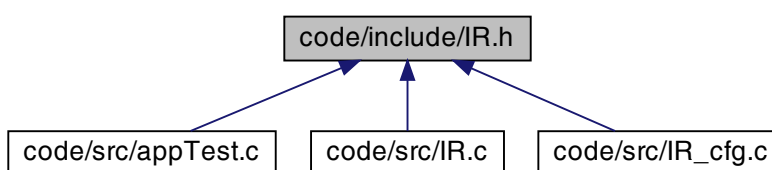
Here is the caller graph for this function:



5.15 code/include/IR.h File Reference

Interfaces header file for [IR.c](#).

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



Enumerations

- enum [IR_SENSOR_t](#) { [IR_0](#) , [IR_1](#) , [IR_2](#) }

Functions

- void [IR_Init](#) (void)
Initialize IR Sensors Configurations.
- [STATE_t IR_GetStatus](#) ([IR_SENSOR_t](#) sensor)

Get status of a specific IR Sensor.

- [u32 IR_ScanAll](#) (void)

Scan all IR Sensors and return the status of all of them.

- [u8 IR_GetCount](#) (void)

Get the number of configured IR sensors.

5.15.1 Detailed Description

Interfaces header file for [IR.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.15.2 Enumeration Type Documentation

5.15.2.1 `IR_SENSOR_t` enum [IR_SENSOR_t](#)

Typedefs

Enumerator

<code>IR↔ _0</code>	
<code>IR↔ _1</code>	
<code>IR↔ _2</code>	

Definition at line 15 of file IR.h.

5.15.3 Function Documentation

5.15.3.1 IR_GetCount() `u8 IR_GetCount (void)`

Get the number of configured IR sensors.

Returns

Number of configured IR sensors

Example:

```
IR_GetNumOfSensors(); // returns 2 if only IR_0 and IR_1 are configured
```

Definition at line 51 of file IR.c.

5.15.3.2 IR_GetStatus() `STATE_t IR_GetStatus (IR_SENSOR_t sensor)`

Get status of a specific IR Sensor.

Parameters

in	<i>sensor</i>	Number of the sensor to be read; IR_0, IR_1, ..., IR7
----	---------------	---

Returns

State of the sensor; HIGH if White, LOW if black.

Example:

```
IR_GetStatus(IR_0); // \ref HIGH if IR_0 is on white track and \ref LOW if black track, member of  
                     \ref STATE_t enum
```

Definition at line 18 of file IR.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.15.3.3 IR_Init() `void IR_Init (`
`void)`

Initialize IR Sensors Configurations.

API FUNCTIONS

Definition at line 14 of file IR.c.

Here is the caller graph for this function:



5.15.3.4 IR_ScanAll() `u32 IR_ScanAll (`
`void)`

Scan all IR Sensors and return the status of all of them.

Returns

State of all sensors; HIGH if White, LOW if black.

Example:

```
IR_ScanAll();  
// when it returns a 0x0F00FFF0(0B0000_1111_0000_0000_1111_1111_1111_0000):  
// This means:  
// * (IR_0 --> IR_15) and (IR_24 --> IR_27) are on white track.  
// * (IR_16 --> IR_23) and (IR_28 --> IR_31) are on black track.
```

Note

This function:

- Only scans the configured sensors. So, if you have configured only IR_0 and IR_1, this function will only return a maximum of 0x00000003 (0B0000_0000_0000_0000_0000_0000_0011)
- Scans in the order of sensors' configurations. So, if you have configured IR_1 before IR_0 in the configuration, the status of IR_1 will be returned on bit 0 and the status of IR_0 will on bit 1.

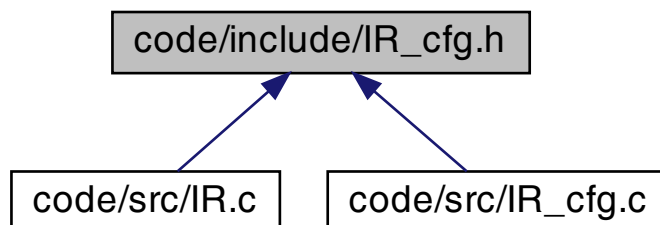
Definition at line 41 of file IR.c.

Here is the call graph for this function:

**5.16 code/include/IR_cfg.h File Reference**

Configuration header file for [IR.c](#).

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

**Data Structures**

- struct [IR_CONFIG_t](#)

Variables

- [IR_CONFIG_t](#) IR_configs []
- const u8 countIRSensorsConfigured

5.16.1 Detailed Description

Configuration header file for [IR.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.16.2 Variable Documentation

5.16.2.1 countIRSensorsConfigured `const u8 countIRSensorsConfigured [extern]`

Definition at line 28 of file [IR_cfg.c](#).

5.16.2.2 IR_configs `IR_CONFIG_t IR_configs[] [extern]`

Note

ACTIVE_LOW means that the pin is:

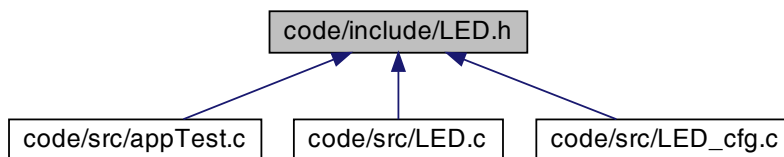
- LOW when the sensor is detecting an object (white)
- HIGH when the sensor is not detecting an object (black) ACTIVE_HIGH means that the pin is:
- HIGH when the sensor is detecting an object (white)
- LOW when the sensor is not detecting an object (black)

Definition at line 22 of file [IR_cfg.c](#).

5.17 code/include/LED.h File Reference

Interfaces header file for [LED.c](#).

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



Enumerations

- enum [LED_t](#) {
 [LED_0](#), [LED_1](#), [LED_2](#), [LED_3](#),
 [LED_4](#), [LED_5](#), [LED_6](#), [LED_7](#) }
- enum [LED_STATE_t](#) { [LED_OFF](#), [LED_ON](#) }

Functions

- void [LED_Init](#) (void)
 Initializes LEDs connected to DIO.
- void [LED_TurnOnOff](#) ([LED_t](#) led, [LED_STATE_t](#) state)
 Turns on/off a specific LED.
- void [LED_Toggle](#) ([LED_t](#) led)
 Toggle state of a specific LED.

5.17.1 Detailed Description

Interfaces header file for [LED.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.17.2 Enumeration Type Documentation

5.17.2.1 [LED_STATE_t](#) enum [LED_STATE_t](#)

Enumerator

LED_OFF	
LED_ON	

Definition at line 28 of file LED.h.

5.17.2.2 LED_t enum [LED_t](#)**TYPEDEFS****Enumerator**

LED↔ _0	
LED↔ _1	
LED↔ _2	
LED↔ _3	
LED↔ _4	
LED↔ _5	
LED↔ _6	
LED↔ _7	

Definition at line 17 of file LED.h.

5.17.3 Function Documentation**5.17.3.1 LED_Init()** void LED_Init (
void)

Initializes LEDs connected to DIO.

API's

Initializes LEDs connected to DIO.

Definition at line 17 of file LED.c.

Here is the caller graph for this function:



5.17.3.2 LED_Toggle() void LED_Toggle (
 const LED_t led)

Toggle state of a specific LED.

Parameters

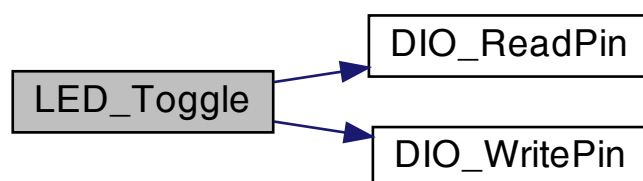
in	led	The LED to be oggles
----	-----	----------------------

Example:

```
LED_Toggle(LED_0); // toggle LED_0: LED_0 will be turned on if it was off and vice versa
```

Definition at line 45 of file LED.c.

Here is the call graph for this function:



5.17.3.3 LED_TurnOnOff() void LED_TurnOnOff (
 const LED_t led,
 const LED_STATE_t state)

Turns on/off a specific LED.

Parameters

in	<i>led</i>	the LED to be turned on/off, LED_0 to LED_7
in	<i>state</i>	the state of the LED, LED_ON or LED_OFF

For example:

- LED_TurnOnOff(LED_0, LED_ON) turns on LED_0
- LED_TurnOnOff(LED_0, LED_OFF) turns off LED_0

Turns on/off a specific LED.

Parameters

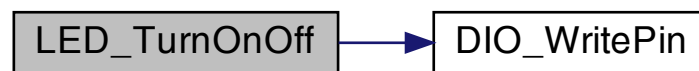
in	<i>led</i>	The LED to be turned on/off
in	<i>state</i>	The state of the LED, either LED_ON or LED_OFF

Example:

```
LED_Set(LED_0, LED_ON); // turn on LED_0
LED_Set(LED_0, LED_OFF); // turn off LED_0
```

Definition at line 29 of file LED.c.

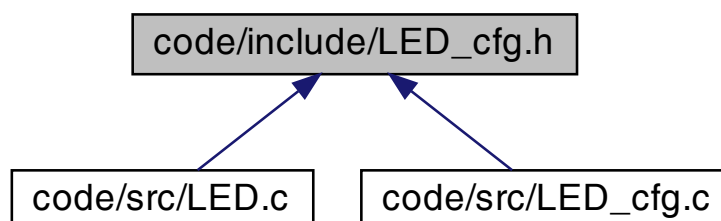
Here is the call graph for this function:



5.18 code/include/LED_cfg.h File Reference

Configuration header file for [LED.c](#).

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



Data Structures

- struct [LED_CONFIGS_t](#)

Variables

- [LED_CONFIGS_t](#) ledConfigs []
- const [u8](#) countLedsConfigured

5.18.1 Detailed Description

Configuration header file for [LED.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.18.2 Variable Documentation

5.18.2.1 countLedsConfigured `const u8 countLedsConfigured [extern]`

Definition at line 19 of file LED_cfg.c.

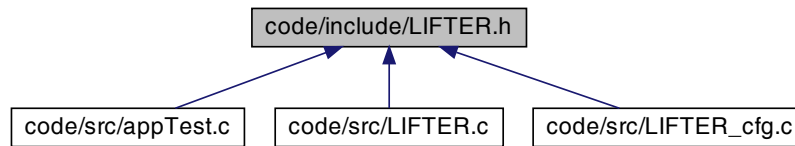
5.18.2.2 ledConfigs `LED_CONFIGS_t ledConfigs[] [extern]`

Definition at line 15 of file LED_cfg.c.

5.19 code/include/LIFTER.h File Reference

Interfaces header file for [LIFTER.c](#).

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



Functions

- void [LIFTER_Init](#) (void)
Initialize the lifter.
- void [LIFTER_MoveUp](#) ()
Move the lifter up.
- void [LIFTER_MoveDown](#) ()
Move the lifter down.
- void [LIFTER_Enable](#) (void)
Enable the lifter motor.
- void [LIFTER_Disable](#) (void)
Disable the lifter motor.
- void [LIFTER_SetSpeed](#) (u8 speed)
Set the speed of the lifter motor.
- void [LIFTER_SetOverallStroke](#) (u8 overallStroke)
Set the stroke of the lifter motor per revolution in mm.
- void [LIFTER_SetPulsesPerRevolution](#) (u16 pulsesPerRevolution)
Set the number of pulses per revolution.
- void [LIFTER_SetRevolutionStroke](#) (u8 revolutionStroke)
Set the distance per revolution.

5.19.1 Detailed Description

Interfaces header file for [LIFTER.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

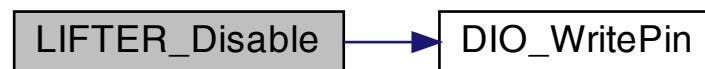
5.19.2 Function Documentation

5.19.2.1 LIFTER_Disable() `void LIFTER_Disable (void)`

Disable the lifter motor.

Definition at line 131 of file LIFTER.c.

Here is the call graph for this function:



5.19.2.2 LIFTER_Enable() `void LIFTER_Enable (void)`

Enable the lifter motor.

Definition at line 124 of file LIFTER.c.

Here is the call graph for this function:



5.19.2.3 LIFTER_Init() `void LIFTER_Init (`
`void)`

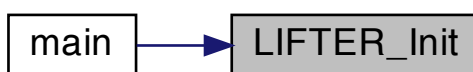
Initialize the lifter.

Initialize the lifter.

API FUNCTIONS IMPLEMENTATION *

Definition at line 103 of file LIFTER.c.

Here is the caller graph for this function:



5.19.2.4 LIFTER_MoveDown() `void LIFTER_MoveDown (`
`void)`

Move the lifter down.

Move the lifter down.

Definition at line 117 of file LIFTER.c.

5.19.2.5 LIFTER_MoveUp() `void LIFTER_MoveUp (`
`void)`

Move the lifter up.

Move the lifter up.

Definition at line 110 of file LIFTER.c.

5.19.2.6 LIFTER_SetOverallStroke() `void LIFTER_SetOverallStroke (`
`const u8 overallStroke)`

Set the stroke of the lifter motor per revolution in mm.

Parameters

in	<i>overallStroke</i>	The stroke of the lifter motor per revolution in mm
----	----------------------	---

Definition at line 147 of file LIFTER.c.

5.19.2.7 LIFTER_SetPulsesPerRevolution() void LIFTER_SetPulsesPerRevolution (
const u16 *pulsesPerRevolution*)

Set the number of pulses per revolution.

Parameters

in	<i>pulsesPerRevolution</i>	The number of pulses per revolution
----	----------------------------	-------------------------------------

Definition at line 155 of file LIFTER.c.

5.19.2.8 LIFTER_SetRevolutionStroke() void LIFTER_SetRevolutionStroke (
const u8 *revolutionStroke*)

Set the distance per revolution.

Parameters

in	<i>revolutionStroke</i>	The distance per revolution in mm
----	-------------------------	-----------------------------------

Definition at line 163 of file LIFTER.c.

5.19.2.9 LIFTER_SetSpeed() void LIFTER_SetSpeed (
const u8 *speed*)

Set the speed of the lifter motor.

Parameters

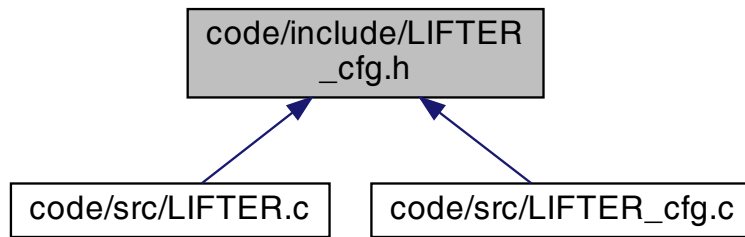
in	<i>speed</i>	The speed of the lifter motor
----	--------------	-------------------------------

Definition at line 139 of file LIFTER.c.

5.20 code/include/LIFTER_cfg.h File Reference

Configuration header file for [LIFTER.c](#).

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



Data Structures

- struct [LIFTER_CONFIGS_t](#)

Macros

- #define [ENABLE_CONNECTION](#) 0
If you are using enable pin, Write 1, else write 0.

Enumerations

- enum [LIFTER_DIR_t](#) { [LIFTER_UP](#) , [LIFTER_DOWN](#) }

Variables

- [LIFTER_CONFIGS_t](#) [LifterConfigs](#)
Lifter configurations.

5.20.1 Detailed Description

Configuration header file for [LIFTER.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.20.2 Macro Definition Documentation

5.20.2.1 ENABLE_CONNECTION `#define ENABLE_CONNECTION 0`

If you are using enable pin, Write 1, else write 0.

Definition at line 15 of file LIFTER_cfg.h.

5.20.3 Enumeration Type Documentation

5.20.3.1 LIFTER_DIR_t `enum LIFTER_DIR_t`

DO NOT CHANGE ANYTHING BELOW THIS LINE

Enumerator

LIFTER_UP	
LIFTER_DOWN	

Definition at line 20 of file LIFTER_cfg.h.

5.20.4 Variable Documentation

5.20.4.1 LifterConfigs `LIFTER_CONFIGS_t LifterConfigs [extern]`

Lifter configurations.

Note

stepSize is the number of millimeters that the lifter will move when calling LIFTER_move function.

For example, if the stepSize is 10, the lifter will move 10 millimeters when calling LIFTER_move function.

pulsePerRevolution is the number of pulses that the lifter will make to make one revolution.

For example, if the pulsePerRevolution is 200, the lifter will make 200 pulses to make one revolution. Options:

- 200 --> Full step
- 400 --> Half step
- 800 --> 1/4 step --> Not allowed with TB6560
- 1600 --> 1/8 step
- 3200 --> 1/16 step
- 6400 --> 1/32 step --> Not allowed with TB6560

distancePerRevolution is the number of millimeters that the lifter will move in one revolution.

For example, if the distancePerRevolution is 100, the lifter will move 100 millimeters in one revolution.

speed is the number of millimeters that the lifter will move in one second. For example, if the speed is 100, the lifter will move 100 millimeters in one second.

speed must be less \leq stepSize

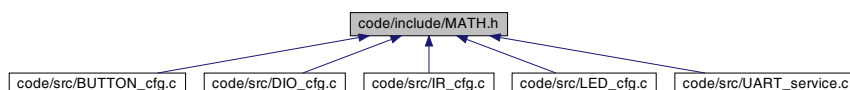
You can change default speed, overallStroke, pulsePerRevolution, and revolutionStroke by calling faunctions LIFTER_SetSpeed, LIFTER_SetOverallStroke, LIFTER_SetPulsesPerRevolution, and LIFTER_SetRevolutionStroke. See prototypes of these functions for more details.

Definition at line 46 of file LIFTER_cfg.c.

5.21 code/include/MATH.h File Reference

Common math functions and constants.

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



Macros

- #define **PI** 3.14159265358979323846
The value of PI constant.
- #define **CLAMP**(x, min, max) ((x) < (min) ? (min) : ((x) > (max) ? (max) : (x)))
*Get the **Clamp value between min and max.***
- #define **DEG_TO_RAD**(x) ((x) * **PI** / 180.0f)
Convert degrees to radians.
- #define **RAD_TO_DEG**(x) ((x) * 180.0f / **PI**)
Convert radians to degrees.
- #define **MIN**(a, b) ((a) < (b) ? (a) : (b))
Get the < b>minimum value between two values.
- #define **MAX**(a, b) ((a) > (b) ? (a) : (b))
Get the < b>maximum value between two values.
- #define **ABS**(x) ((x) < 0 ? -(x) : (x))
Absolute value of a number.
- #define **SIGN**(x) ((x) < 0 ? -1 : 1)
Get the sign of a number.
- #define **IS_POSITIVE**(x) ((x) > 0)
Check if the number is positive (greater than zero).
- #define **IS_NEGATIVE**(x) ((x) < 0)
Check if the number is negative (less than zero).
- #define **FLOOR**(x) ((int)(x) + ((x) > 0 ? 0 : -1))
Get the floor value of a number. It is the largest integer that is less than or equal to x.
- #define **CEIL**(x) ((int)(x) + ((x) > 0 ? 1 : 0))

- Get the Ceil value of a number. It is the smallest integer that is greater than or equal to x.*

 - #define **ROUND**(x) ((int) ((x) + ((x) > 0 ? 0.5f : -0.5f)))
- Get the round value of a number. It is the nearest integer to x.*

 - #define **FRACTION**(x) ((x) - (int)(x))
- Get the fractional part of a floating point number.*

 - #define **LERP**(a, b, t) ((a) + (t) * ((b) - (a))) /* Linear interpolation */
- Gets the Linear Interpolation between two values.*

 - #define **SQRT**(x)
 - #define **POW**(x, y)
 - #define **LOG**(x)
 - #define **LOG2**(x)
 - #define **LOG10**(x)
 - #define **EXP**(x)
 - #define **ACOS**(x) ((-0.69813170079773212 * (x) * (x) - 0.87266462599716477) * (x) + 1.5707963267948966)
- Arc Cosine of an angle in radians.*

 - #define **ASIN**(x) ((0.69813170079773212 * (x) * (x) + 0.87266462599716477) * (x) + 1.5707963267948966)
- Arc Sine of an angle in radians.*

 - #define **ATAN**(x) ((0.55228474983079331 * (x)) + 1.5707963267948966)
- Arc Tangent of an angle in radians.*

 - #define **COS**(x) ((0.87266462599716477 * (x)) + 1.5707963267948966)
- Cosine of an angle in radians.*

 - #define **SIN**(x) ((0.69813170079773212 * (x)) + 1.5707963267948966)
- Sine of an angle in radians.*

 - #define **TAN**(x) ((0.87266462599716477 * (x)) + 1.5707963267948966)
- Tangent of an angle in radians.*

 - #define **SIZE_OF_ARRAY**(array) (sizeof(array) / sizeof(array[0]))
- This is a macro like function to get the size of an array in bytes.*

5.21.1 Detailed Description

Common math functions and constants.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-19

Copyright

Copyright (c) 2022

5.21.2 Macro Definition Documentation

5.21.2.1 ABS #define ABS(
 x) ((x) < 0 ? -(x) : (x))

Absolute value of a number.

Parameters

in	x	The number to take the absolute value of.
----	---	---

Returns

The absolute value of x.

Example:

```
ABS(-1.0f);           // returns 1.0f
ABS(1.0f);            // returns 1.0f
ABS(0.0f);            // returns 0.0f
ABS(-0.0f);           // returns 0.0f
```

Definition at line 118 of file MATH.h.

```
5.21.2.2 ACOS #define ACOS(
    x ) ( (-0.69813170079773212 * (x) * (x) - 0.87266462599716477) * (x) + 1.5707963267948966
)
```

Arc Cosine of an angle in radians.

Parameters

in	x	The value to find the Arc Cosine of.
----	---	--------------------------------------

Note

It has a range of [0, PI].

It is the inverse of the Cosine function.

It has a maximum error of 0.18 radians.

Definition at line 241 of file MATH.h.

```
5.21.2.3 ASIN #define ASIN(
    x ) ( (0.69813170079773212 * (x) * (x) + 0.87266462599716477) * (x) + 1.5707963267948966
)
```

Arc Sine of an angle in radians.

Parameters

in	x	The value to find the Arc Sine of.
----	---	------------------------------------

Note

It has a range of $[-\pi/2, \pi/2]$.

It is the inverse of the Sine function.

It has a maximum error of 0.18 radians.

Definition at line 250 of file MATH.h.

5.21.2.4 ATAN `#define ATAN(
x) ((0.55228474983079331 * (x)) + 1.5707963267948966)`

Arc Tangent of an angle in radians.

Parameters

in	x	The value to find the Arc Tangent of.
----	---	---------------------------------------

Note

It has a range of $[-\pi/2, \pi/2]$.

It is the inverse of the Tangent function.

It has a maximum error of 0.18 radians.

Definition at line 259 of file MATH.h.

5.21.2.5 CEIL `#define CEIL(
x) ((int)(x) + ((x) > 0 ? 1 : 0))`

Get the Ceil value of a number. It is the smallest integer that is greater than or equal to x.

Parameters

in	x	The number to get the Ceil value of.
----	---	--------------------------------------

Returns

The Ceil value of x.

Example:

```
CEIL(5.5f); // returns 6
CEIL(-5.5f); // returns -5
CEIL(1.33f); // returns 2
```

Definition at line 165 of file MATH.h.

5.21.2.6 CLAMP `#define CLAMP(`
 `x,`
 `min,`
 `max) ((x) < (min) ? (min) : ((x) > (max) ? (max) : (x)))`

Get the **Clamp** value between min and max.

Parameters

in	<i>value</i>	The value to clamp.
in	<i>min</i>	The minimum value.
in	<i>max</i>	The maximum value.

Returns

The clamped value.

Example:

```
CLAMP(1.0f, 0.0f, 1.0f);    // returns 1.0f
CLAMP(0.0f, 0.0f, 1.0f);    // returns 0.0f
CLAMP(-1.0f, 0.0f, 1.0f);   // returns 0.0f
CLAMP(2.0f, 0.0f, 1.0f);    // returns 1.0f
```

Definition at line 38 of file MATH.h.

5.21.2.7 COS `#define COS(`
 `x) ((0.87266462599716477 * (x)) + 1.5707963267948966)`

Cosine of an angle in radians.

Parameters

in	<i>x</i>	The value to find the Cosine of.
----	----------	----------------------------------

Note

It has a range of [0, 2*PI].

It is the inverse of the Arc Cosine function.

It has a maximum error of 0.18 radians.

Definition at line 268 of file MATH.h.

5.21.2.8 DEG_TO_RAD `#define DEG_TO_RAD(`
 `x) ((x) * PI / 180.0f)`

Convert degrees to radians.

Parameters

in	<i>degrees</i>	The degrees to convert.
----	----------------	-------------------------

Returns

The radians.

Example:

```
DEG2RAD(180.0f);    // returns PI
DEG2RAD(0.0f);      // returns 0.0f
DEG2RAD(-180.0f);   // returns -PI
DEG2RAD(360.0f);    // returns 2*PI
DEG2RAD(-360.0f);   // returns -2*PI
DEG2RAD(720.0f);    // returns 4*PI
DEG2RAD(-720.0f);   // returns -4*PI
DEG2RAD(255.0f);    // returns 4.45059
```

Definition at line 54 of file MATH.h.

5.21.2.9 EXP `#define EXP(
x)`

Definition at line 218 of file MATH.h.

5.21.2.10 FLOOR `#define FLOOR(
x) ((int)(x) + ((x) > 0 ? 0 : -1))`

Get the floor value of a number. It is the largest integer that is less than or equal to x.

Parameters

in	<i>x</i>	The number to get the floor value of.
----	----------	---------------------------------------

Returns

The floor value of x.

Example:

```
FLOOR(5.6f);        // returns 5
FLOOR(-5.3f);       // returns -6
FLOOR(1.33f);       // returns 1
FLOOR(-1.33f);      // returns -2
```

Definition at line 154 of file MATH.h.

5.21.2.11 FRACTION `#define FRACTION(
x) ((x) - (int)(x))`

Get the fractional part of a floating point number.

Parameters

in	x	The number to get the fractional part of.
----	---	---

Returns

The fractional part of x.

Example:

```
FRACTION(1.6f);           // returns 0.6f
FRACTION(-1.6f);          // returns -0.6f
FRACTION(1.33f);          // returns 0.33f
FRACTION(-1.33f);         // returns -0.33f
FRACTION(1.0f);           // returns 0.0f
FRACTION(-1.0f);          // returns 0.0f
FRACTION(0.0f);           // returns 0.0f
```

Definition at line 192 of file MATH.h.

5.21.2.12 IS_NEGATIVE `#define IS_NEGATIVE(
x) ((x) < 0)`

Check if the number is negative (less than zero).

Parameters

in	x	The number to check.
----	---	----------------------

Returns

TRUE if x is negative, FALSE otherwise.

Definition at line 142 of file MATH.h.

5.21.2.13 IS_POSITIVE `#define IS_POSITIVE(
x) ((x) > 0)`

Check if the number is positive (greater than zero).

Parameters

in	x	The number to check.
----	---	----------------------

Returns

TRUE if x is positive, FALSE otherwise.

Definition at line 135 of file MATH.h.

5.21.2.14 LERP `#define LERP(
 a,
 b,
 t) ((a) + (t) * ((b) - (a))) /* Linear interpolation */`

Gets the Linear Interpolation between two values.

Parameters

in	a	The first value.
in	b	The second value.
in	t	The interpolation value.

Returns

The interpolated value.

Example:

```
LERP(1.0f, 2.0f, 0.5f);    // returns 1.5f
LERP(1.0f, 2.0f, 0.0f);    // returns 1.0f
LERP(1.0f, 2.0f, 1.0f);    // returns 2.0f
LERP(1.0f, 2.0f, 2.0f);    // returns 2.0f
LERP(1.0f, 2.0f, -1.0f);    // returns 1.0f
LERP(1.0f, 2.0f, -2.0f);    // returns 1.0f
LERP(1.0f, 2.0f, 3.0f);    // returns 2.0f
LERP(1.0f, 2.0f, -3.0f);    // returns 1.0f
```

Definition at line 210 of file MATH.h.

5.21.2.15 LOG `#define LOG(
 x)`

Definition at line 215 of file MATH.h.

5.21.2.16 LOG10 `#define LOG10(
 x)`

Definition at line 217 of file MATH.h.

5.21.2.17 LOG2 `#define LOG2(
 x)`

Definition at line 216 of file MATH.h.

5.21.2.18 MAX `#define MAX(
 a,
 b) ((a) > (b) ? (a) : (b))`

Get the < b>maximum value between two values.

Parameters

in	<i>a</i>	The first value.
in	<i>b</i>	The second value.

Returns

The maximum value of the first and second values.

Example:

```
MAX(1.0f, 2.0f);    // returns 2.0f
MAX(2.0f, 1.0f);    // returns 2.0f
MAX(1.0f, 1.0f);    // returns 1.0f
MAX(2.0f, 2.0f);    // returns 2.0f
MAX(0.0f, -1.0f);    // returns 0.0f
MAX(-1.0f, 0.0f);    // returns 0.0f
MAX(-1.0f, -1.0f);   // returns -1.0f
MAX(-2.0f, -1.0f);   // returns -1.0f
MAX(-1.0f, -2.0f);   // returns -1.0f
```

Definition at line 106 of file MATH.h.

5.21.2.19 MIN `#define MIN(
 a,
 b) ((a) < (b) ? (a) : (b))`

Get the < b>minimum value between two values.

Parameters

in	<i>a</i>	The first value.
in	<i>b</i>	The second value.

Returns

The minimum value of the first and second values.

Example:

```

MIN(1.0f, 2.0f);    // returns 1.0f
MIN(2.0f, 1.0f);    // returns 1.0f
MIN(1.0f, 1.0f);    // returns 1.0f
MIN(2.0f, 2.0f);    // returns 2.0f
MIN(0.0f, -1.0f);   // returns -1.0f
MIN(-1.0f, 0.0f);   // returns -1.0f
MIN(-1.0f, -1.0f);  // returns -1.0f
MIN(-2.0f, -1.0f);  // returns -2.0f
MIN(-1.0f, -2.0f);  // returns -2.0f

```

Definition at line 88 of file MATH.h.

5.21.2.20 PI `#define PI 3.14159265358979323846`

The value of PI constant.

Note

This value is used in the trigonometric functions.

Definition at line 24 of file MATH.h.

5.21.2.21 POW `#define POW(`
`x,`
`y)`

Definition at line 214 of file MATH.h.

5.21.2.22 RAD_TO_DEG `#define RAD_TO_DEG(`
`x) ((x) * 180.0f / PI)`

Convert radians to degrees.

Parameters

in	<i>radians</i>	The radians to convert.
----	----------------	-------------------------

Returns

The degrees.

Example:

```

RAD2DEG(PI);           // returns 180.0f
RAD2DEG(0.0f);         // returns 0.0f
RAD2DEG(-PI);          // returns -180.0f
RAD2DEG(2*PI);         // returns 360.0f
RAD2DEG(-2*PI);        // returns -360.0f
RAD2DEG(4*PI);         // returns 720.0f
RAD2DEG(-4*PI);        // returns -720.0f
RAD2DEG(4.45059);      // returns 255.0f

```

Definition at line 70 of file MATH.h.

5.21.2.23 ROUND `#define ROUND(`
`x) ((int) ((x) + ((x) > 0 ? 0.5f : -0.5f)))`

Get the round value of a number. It is the nearest integer to x.

Parameters

in	x	The number to get the round value of.
----	---	---------------------------------------

Returns

The round value of x.

Example:

```

ROUND(1.6f);           // returns 2.0f
ROUND(-1.6f);          // 1.6f -2.0f
ROUND(1.33);           // returns 1.0f
ROUND(-1.33f);         // returns -1.0f

```

Definition at line 177 of file MATH.h.

5.21.2.24 SIGN `#define SIGN(`
`x) ((x) < 0 ? -1 : 1)`

Get the sign of a number.

Parameters

in	x	The number to get the sign of.
----	---	--------------------------------

Returns

-1 if x is negative, 1 if x is positive, 0 if x is zero.

Example:

```
SIGN(-5);           // returns -1
SIGN(5);            // returns 1
```

Definition at line 128 of file MATH.h.

5.21.2.25 SIN `#define SIN(
x) ((0.69813170079773212 * (x)) + 1.5707963267948966)`

Sine of an angle in radians.

Parameters

in	<i>x</i>	The value to find the Sine of.
----	----------	--------------------------------

Note

It has a range of $[-\pi/2, \pi/2]$.

It is the inverse of the Arc Sine function.

It has a maximum error of 0.18 radians.

Definition at line 277 of file MATH.h.

5.21.2.26 SIZE_OF_ARRAY `#define SIZE_OF_ARRAY(
array) (sizeof(array) / sizeof(array[0]))`

This is a macro like function to get the size of an array in bytes.

Parameters

in	<i>array</i>	The name of the array
----	--------------	-----------------------

Returns

The size of the array in bytes

Warning

This macro is not intended to be used to get the size of a dynamic array. The array must be declared at compile time. This is because the sizeof operator is a compile time operator and cannot be used to get the size of a dynamic array.

Example:

```
SIZE_OF_ARRAY(arrayName); // returns the size of the array <arrayName> in bytes
```

Definition at line 314 of file MATH.h.

5.21.2.27 SQRT `#define SQRT(
x)`

Definition at line 213 of file MATH.h.

5.21.2.28 TAN `#define TAN(
x) ((0.87266462599716477 * (x)) + 1.5707963267948966)`

Tangent of an angle in radians.

Parameters

in	x	The value to find the Tangent of.
----	---	-----------------------------------

Note

It has a range of $[-\pi/2, \pi/2]$.

It is the inverse of the Arc Tangent function.

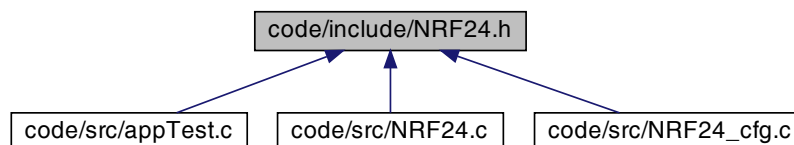
It has a maximum error of 0.18 radians.

Definition at line 286 of file MATH.h.

5.22 code/include/NRF24.h File Reference

Interfaces header file for [NRF24.c](#).

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



Functions

- void [NRF24_Init](#) (void)
- void [NRF24_TxMode](#) (void)
- void [NRF24_SendString](#) (u8 *data, u8 length)
- void [NRF24_RxMode](#) (void)
- u8 [NRF24_Available](#) (void)
- [ERROR_STATUS_t](#) [NRF24_ReceiveString](#) (u8 *data, u8 length)

5.22.1 Detailed Description

Interfaces header file for [NRF24.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.22.2 Function Documentation

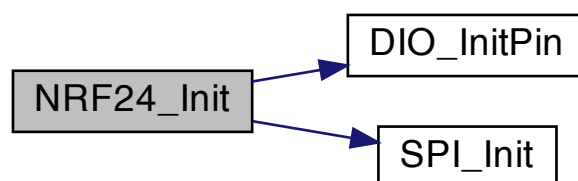
5.22.2.1 NRF24_Available() `u8 NRF24_Available (void)`

Definition at line 307 of file NRF24.c.

5.22.2.2 NRF24_Init() `void NRF24_Init (void)`

Definition at line 155 of file NRF24.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.22.2.3 NRF24_ReceiveString() `ERROR_STATUS_t NRF24_ReceiveString (`
`u8 * data,`
`u8 length)`

Definition at line 328 of file NRF24.c.

5.22.2.4 NRF24_RxMode() `void NRF24_RxMode (`
`void)`

Definition at line 249 of file NRF24.c.

5.22.2.5 NRF24_SendString() `void NRF24_SendString (`
`u8 * data,`
`u8 length)`

Definition at line 214 of file NRF24.c.

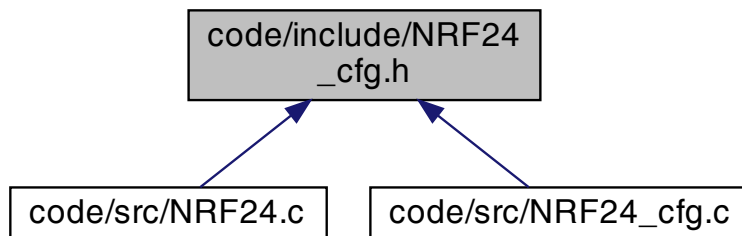
5.22.2.6 NRF24_TxMode() `void NRF24_TxMode (`
`void)`

Definition at line 191 of file NRF24.c.

5.23 code/include/NRF24_cfg.h File Reference

Configuration header file for [NRF24.c](#).

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



Data Structures

- struct [CONNECTIONS_t](#)
- struct [PINS_t](#)
- struct [NRF24_t](#)

Enumerations

- enum [PIPE_t](#) {
 [RX_PIPE0](#) , [RX_PIPE1](#) , [RX_PIPE2](#) , [RX_PIPE3](#) ,
 [RX_PIPE4](#) , [RX_PIPE5](#) }

Variables

- [NRF24_t NRF24_cfg](#)

5.23.1 Detailed Description

Configuration header file for [NRF24.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.23.2 Enumeration Type Documentation

5.23.2.1 PIPE_t enum PIPE_t

Enumerator

RX_PIPE0	
RX_PIPE1	
RX_PIPE2	
RX_PIPE3	
RX_PIPE4	
RX_PIPE5	

Definition at line 23 of file NRF24_cfg.h.

5.23.3 Variable Documentation

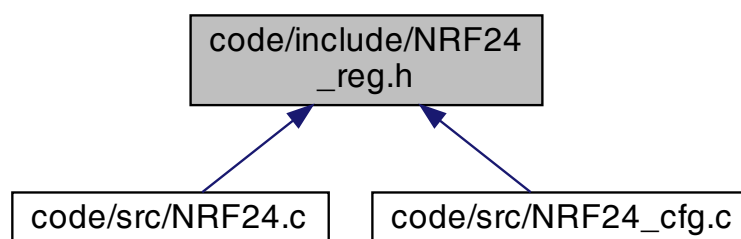
5.23.3.1 NRF24_cfg NRF24_t NRF24_cfg [extern]

Definition at line 19 of file NRF24_cfg.c.

5.24 code/include/NRF24_reg.h File Reference

Registers of NRF24L01 wireless transceiver module.

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



Macros

- #define [CONFIG](#) 0x00
- #define [EN_AA](#) 0x01
- #define [EN_RXADDR](#) 0x02
- #define [SETUP_AW](#) 0x03
- #define [SETUP_RETR](#) 0x04
- #define [RF_CH](#) 0x05
- #define [RF_SETUP](#) 0x06
- #define [STATUS](#) 0x07
- #define [OBSERVE_TX](#) 0x08
- #define [CD](#) 0x09
- #define [RX_ADDR_P0](#) 0x0A
- Pipes 0-5 addressses.*
 - #define [RX_ADDR_P1](#) 0x0B
 - #define [RX_ADDR_P2](#) 0x0C
 - #define [RX_ADDR_P3](#) 0x0D
 - #define [RX_ADDR_P4](#) 0x0E
 - #define [RX_ADDR_P5](#) 0x0F
 - #define [TX_ADDR](#) 0x10
 - Transmit address.*
- #define [RX_PW_P0](#) ((u8)0x11)
- RX payload widths of pipes 0-5: 0 - 32 bytes.*
 - #define [RX_PW_P1](#) ((u8)0x12)
 - #define [RX_PW_P2](#) ((u8)0x13)
 - #define [RX_PW_P3](#) ((u8)0x14)
 - #define [RX_PW_P4](#) ((u8)0x15)
 - #define [RX_PW_P5](#) ((u8)0x16)
 - #define [FIFO_STATUS](#) ((u8)0x17)
 - #define [DYNPD](#) ((u8)0x1C)
 - #define [FEATURE](#) ((u8)0x1D)
 - #define [R_REGISTER](#) ((u8)0x00U) /* Read command */
 - #define [W_REGISTER](#) ((u8)0x20U) /* Write command */
 - #define [R_RX_PAYLOAD](#) ((u8)0x61U) /* Read RX payload */
 - #define [W_TX_PAYLOAD](#) ((u8)0xA0U) /* Write TX payload */
 - #define [FLUSH_TX](#) ((u8)0xE1U) /* Flush TX FIFO */
 - #define [FLUSH_RX](#) ((u8)0xE2U) /* Flush RX FIFO */
 - #define [REUSE_TX_PL](#) ((u8)0xE3U) /* Reuse last transmitted payload */
 - #define [R_RX_PL_WID](#) ((u8)0x60U) /* Read RX payload width for the top RX payload in the FIFO */
 - #define [W_ACK_PAYLOAD](#) ((u8)0xA8U) /* Write Payload to be transmitted together with ACK */
 - #define [W_TX_PAYLOAD_NOACK](#) ((u8)0xB0U) /* Write Payload to be transmitted without ACK */
 - #define [NOP](#) ((u8)0xFF) /* No operation */
 - #define [ACTIVATE](#) ((u8)0x50U) /* Activate features */
 - #define [REGISTER_MASK](#) ((u8)0x1FU) /* Register mask */

Enumerations

- enum {
[PRIM_RX](#) , [PWR_UP](#) , [CRCO](#) , [EN_CRC](#) ,
[MASK_MAX_RT](#) , [MASK_TX_DS](#) , [MASK_RX_DR](#) }
- enum {
[ENAA_P0](#) , [ENAA_P1](#) , [ENAA_P2](#) , [ENAA_P3](#) ,
[ENAA_P4](#) , [ENAA_P5](#) }

- enum {
 ERX_P0 , ERX_P1 , ERX_P2 , ERX_P3 ,
 ERX_P4 , ERX_P5 }
- enum { AW0 , AW1 }
- enum {
 ARC0 , ARC1 , ARC2 , ARC3 ,
 ARD0 , ARD1 , ARD2 , ARD3 }
- enum {
 RF_CH0 , RF_CH1 , RF_CH2 , RF_CH3 ,
 RF_CH4 , RF_CH5 , RF_CH6 }
- enum {
 Obsolete , RF_PWR0 , RF_PWR1 , RF_DR_HIGH ,
 PLL_LOCK , RF_DR_LOW , RESERVED , CONT_WAVE }
- enum {
 TX_FULL0 , RX_P_NO0 , RX_P_NO1 , RX_P_NO2 ,
 MAX_RT , TX_DS , RX_DR }
- enum {
 ARC_CNT0 , ARC_CNT1 , ARC_CNT2 , ARC_CNT3 ,
 PLOS_CNT0 , PLOS_CNT1 , PLOS_CNT2 , PLOS_CNT3 }
- enum { RPD }
- enum {
 RX_EMPTY , RX_FULL1 , TX_EMPTY = 4 , TX_FULL ,
 TX_REUSE }
- enum {
 DYNPD0 , DYNPD1 , DYNPD2 , DYNPD3 ,
 DYNPD4 , DYNPD5 }
- enum { EN_DYN_ACK , EN_ACK_PAY , EN_DPL }
- enum {
 DPL_P0 , DPL_P1 , DPL_P2 , DPL_P3 ,
 DPL_P4 , DPL_P5 }

5.24.1 Detailed Description

Registers of NRF24L01 wireless transceiver module.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2021-07-31

5.24.2 Macro Definition Documentation

5.24.2.1 ACTIVATE `#define ACTIVATE ((u8)0x50U) /* Activate features */`

Definition at line 192 of file NRF24_reg.h.

5.24.2.2 CD `#define CD 0x09`

Definition at line 146 of file NRF24_reg.h.

5.24.2.3 CONFIG `#define CONFIG 0x00`

Definition at line 137 of file NRF24_reg.h.

5.24.2.4 DYNPD `#define DYNPD ((u8)0x1C)`

Definition at line 174 of file NRF24_reg.h.

5.24.2.5 EN_AA `#define EN_AA 0x01`

Definition at line 138 of file NRF24_reg.h.

5.24.2.6 EN_RXADDR `#define EN_RXADDR 0x02`

Definition at line 139 of file NRF24_reg.h.

5.24.2.7 FEATURE `#define FEATURE ((u8)0x1D)`

Definition at line 175 of file NRF24_reg.h.

5.24.2.8 FIFO_STATUS `#define FIFO_STATUS ((u8)0x17)`

Definition at line 173 of file NRF24_reg.h.

5.24.2.9 FLUSH_RX `#define FLUSH_RX ((u8)0xE2U) /* Flush RX FIFO */`

Definition at line 185 of file NRF24_reg.h.

5.24.2.10 FLUSH_TX `#define FLUSH_TX ((u8)0xE1U) /* Flush TX FIFO */`

Definition at line 184 of file NRF24_reg.h.

5.24.2.11 NOP `#define NOP ((u8)0xFF) /* No operation */`

Definition at line 190 of file NRF24_reg.h.

5.24.2.12 OBSERVE_TX `#define OBSERVE_TX 0x08`

Definition at line 145 of file NRF24_reg.h.

5.24.2.13 R_REGISTER `#define R_REGISTER ((u8)0x00U) /* Read command */`

Definition at line 180 of file NRF24_reg.h.

5.24.2.14 R_RX_PAYLOAD `#define R_RX_PAYLOAD ((u8)0x61U) /* Read RX payload */`

Definition at line 182 of file NRF24_reg.h.

5.24.2.15 R_RX_PL_WID `#define R_RX_PL_WID ((u8)0x60U) /* Read RX payload width for the top
RX payload in the FIFO */`

Definition at line 187 of file NRF24_reg.h.

5.24.2.16 REGISTER_MASK `#define REGISTER_MASK ((u8)0x1FU) /* Register mask */`

Definition at line 193 of file NRF24_reg.h.

5.24.2.17 REUSE_TX_PL `#define REUSE_TX_PL ((u8)0xE3U) /* Reuse last transmitted payload */`

Definition at line 186 of file NRF24_reg.h.

5.24.2.18 RF_CH `#define RF_CH 0x05`

Definition at line 142 of file NRF24_reg.h.

5.24.2.19 RF_SETUP `#define RF_SETUP 0x06`

Definition at line 143 of file NRF24_reg.h.

5.24.2.20 RX_ADDR_P0 `#define RX_ADDR_P0 0x0A`

Pipes 0-5 addressses.

Definition at line 151 of file NRF24_reg.h.

5.24.2.21 RX_ADDR_P1 `#define RX_ADDR_P1 0x0B`

Definition at line 152 of file NRF24_reg.h.

5.24.2.22 RX_ADDR_P2 `#define RX_ADDR_P2 0x0C`

Definition at line 153 of file NRF24_reg.h.

5.24.2.23 RX_ADDR_P3 `#define RX_ADDR_P3 0x0D`

Definition at line 154 of file NRF24_reg.h.

5.24.2.24 RX_ADDR_P4 `#define RX_ADDR_P4 0x0E`

Definition at line 155 of file NRF24_reg.h.

5.24.2.25 RX_ADDR_P5 `#define RX_ADDR_P5 0x0F`

Definition at line 156 of file NRF24_reg.h.

5.24.2.26 RX_PW_P0 `#define RX_PW_P0 ((u8)0x11)`

RX payload widths of pipes 0-5: 0 - 32 bytes.

Definition at line 166 of file NRF24_reg.h.

5.24.2.27 RX_PW_P1 `#define RX_PW_P1 ((u8)0x12)`

Definition at line 167 of file NRF24_reg.h.

5.24.2.28 RX_PW_P2 `#define RX_PW_P2 ((u8)0x13)`

Definition at line 168 of file NRF24_reg.h.

5.24.2.29 RX_PW_P3 `#define RX_PW_P3 ((u8)0x14)`

Definition at line 169 of file NRF24_reg.h.

5.24.2.30 RX_PW_P4 `#define RX_PW_P4 ((u8)0x15)`

Definition at line 170 of file NRF24_reg.h.

5.24.2.31 RX_PW_P5 `#define RX_PW_P5 ((u8)0x16)`

Definition at line 171 of file NRF24_reg.h.

5.24.2.32 SETUP_AW `#define SETUP_AW 0x03`

Definition at line 140 of file NRF24_reg.h.

5.24.2.33 SETUP_RETR `#define SETUP_RETR 0x04`

Definition at line 141 of file NRF24_reg.h.

5.24.2.34 STATUS `#define STATUS 0x07`

Definition at line 144 of file NRF24_reg.h.

5.24.2.35 TX_ADDR `#define TX_ADDR 0x10`

Transmit address.

Definition at line 161 of file NRF24_reg.h.

5.24.2.36 W_ACK_PAYLOAD `#define W_ACK_PAYLOAD ((u8)0xA8U) /* Write Payload to be transmitted together with ACK */`

Definition at line 188 of file NRF24_reg.h.

5.24.2.37 W_REGISTER `#define W_REGISTER ((u8)0x20U) /* Write command */`

Definition at line 181 of file NRF24_reg.h.

5.24.2.38 W_TX_PAYLOAD `#define W_TX_PAYLOAD ((u8)0xA0U) /* Write TX payload */`

Definition at line 183 of file NRF24_reg.h.

5.24.2.39 W_TX_PAYLOAD_NOACK `#define W_TX_PAYLOAD_NOACK ((u8)0xB0U) /* Write Payload to be transmitted without ACK */`

Definition at line 189 of file NRF24_reg.h.

5.24.3 Enumeration Type Documentation**5.24.3.1 anonymous enum** `anonymous enum`

Enumerator

RF_CH0	
RF_CH1	
RF_CH2	
RF_CH3	
RF_CH4	
RF_CH5	
RF_CH6	

Definition at line 55 of file NRF24_reg.h.

5.24.3.2 anonymous enum `anonymous enum`**Enumerator**

Obsolete	
RF_PWR0	
RF_PWR1	
RF_DR_HIGH	
PLL_LOCK	
RF_DR_LOW	
RESERVED	
CONT_WAVE	

Definition at line 65 of file NRF24_reg.h.

5.24.3.3 anonymous enum `anonymous enum`**Enumerator**

TX_FULL0	
RX_P_NO0	
RX_P_NO1	
RX_P_NO2	
MAX_RT	
TX_DS	
RX_DR	

Definition at line 76 of file NRF24_reg.h.

5.24.3.4 anonymous enum `anonymous enum`

Enumerator

ARC_CNT0	
ARC_CNT1	
ARC_CNT2	
ARC_CNT3	
PLOS_CNT0	
PLOS_CNT1	
PLOS_CNT2	
PLOS_CNT3	

Definition at line 86 of file NRF24_reg.h.

5.24.3.5 anonymous enum `anonymous enum`

Enumerator

RPD	
-----	--

Definition at line 97 of file NRF24_reg.h.

5.24.3.6 anonymous enum `anonymous enum`

Enumerator

RX_EMPTY	
RX_FULL1	
TX_EMPTY	
TX_FULL	
TX_REUSE	

Definition at line 101 of file NRF24_reg.h.

5.24.3.7 anonymous enum `anonymous enum`

Enumerator

DYNPD0	
DYNPD1	
DYNPD2	
DYNPD3	
DYNPD4	
DYNPD5	

Definition at line 109 of file NRF24_reg.h.

5.24.3.8 anonymous enum `anonymous_enum`

Enumerator

EN_DYN_ACK	
EN_ACK_PAY	
EN_DPL	

Definition at line 118 of file NRF24_reg.h.

5.24.3.9 anonymous enum `anonymous_enum`

Enumerator

DPL_P0	
DPL_P1	
DPL_P2	
DPL_P3	
DPL_P4	
DPL_P5	

Definition at line 125 of file NRF24_reg.h.

5.24.3.10 anonymous enum `anonymous_enum`

Enumerator

PRIM_RX	
PWR_UP	
CRCO	
EN_CRC	
MASK_MAX_RT	
MASK_TX_DS	
MASK_RX_DR	

Definition at line 11 of file NRF24_reg.h.

5.24.3.11 anonymous enum `anonymous_enum`

Enumerator

ENAA_P0	
ENAA_P1	
ENAA_P2	
ENAA_P3	
ENAA_P4	
ENAA_P5	

Definition at line 21 of file NRF24_reg.h.

5.24.3.12 anonymous enum anonymous enum

Enumerator

ERX_P0	
ERX_P1	
ERX_P2	
ERX_P3	
ERX_P4	
ERX_P5	

Definition at line 30 of file NRF24_reg.h.

5.24.3.13 anonymous enum anonymous enum

Enumerator

AW0	
AW1	

Definition at line 39 of file NRF24_reg.h.

5.24.3.14 anonymous enum anonymous enum

Enumerator

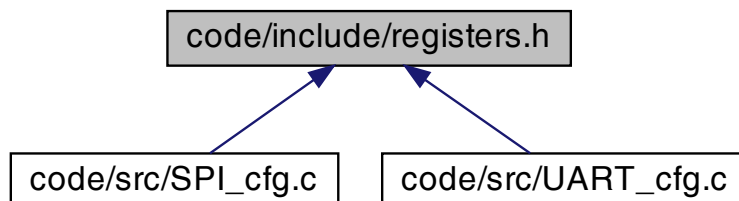
ARC0	
ARC1	
ARC2	
ARC3	
ARD0	
ARD1	
ARD2	
ARD3	

Definition at line 44 of file NRF24_reg.h.

5.25 code/include/registers.h File Reference

Registers of ATmega328p MCU.

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



Macros

- `#define ADC_u8ADMUX_REG (* ((volatile u8 *) 0x27))`
- `#define ADC_u8ADCSRA_REG (* ((volatile u8 *) 0x26))`
- `#define ADC_u8ADCH_REG (* ((volatile u8 *) 0x25))`
- `#define ADC_u8ADCL_REG (* ((volatile u8 *) 0x24))`
- `#define ADC_u8ADCSRB_REG (* ((volatile u8 *) 0x06))`
- `#define EEPROM_u8EECR_REG (* ((volatile u8 *) 0x3F))`
- `#define EEPROM_u8EEDR_REG (* ((volatile u8 *) 0x3E))`
- `#define EEPROM_u8EEARL_REG (* ((volatile u8 *) 0x3D))`
- `#define EEPROM_u8EEARH_REG (* ((volatile u8 *) 0x3C))`
- `#define EEPROM_u8EECR_REG (* ((volatile u8 *) 0x3F))`

5.25.1 Detailed Description

Registers of ATmega328p MCU.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2021-07-31

5.25.2 Macro Definition Documentation

5.25.2.1 ADC_u8ADCH_REG `#define ADC_u8ADCH_REG (* ((volatile u8 *) 0x25))`

Definition at line 16 of file registers.h.

5.25.2.2 ADC_u8ADCL_REG `#define ADC_u8ADCL_REG (* ((volatile u8 *) 0x24))`

Definition at line 17 of file registers.h.

5.25.2.3 ADC_u8ADCSRA_REG `#define ADC_u8ADCSRA_REG (* ((volatile u8 *) 0x26))`

Definition at line 15 of file registers.h.

5.25.2.4 ADC_u8ADCSRB_REG `#define ADC_u8ADCSRB_REG (* ((volatile u8 *) 0x06))`

Definition at line 18 of file registers.h.

5.25.2.5 ADC_u8ADMUX_REG `#define ADC_u8ADMUX_REG (* ((volatile u8 *) 0x27))`

ADC Register

Definition at line 14 of file registers.h.

5.25.2.6 EEPROM_u8EEARH_REG `#define EEPROM_u8EEARH_REG (* ((volatile u8 *) 0x3C))`

Definition at line 26 of file registers.h.

5.25.2.7 EEPROM_u8EEARL_REG `#define EEPROM_u8EEARL_REG (* ((volatile u8 *) 0x3D))`

Definition at line 25 of file registers.h.

5.25.2.8 EEPROM_u8EECR_REG [1/2] `#define EEPROM_u8EECR_REG (* ((volatile u8 *) 0x3F))`

EEPROM Register

Definition at line 27 of file registers.h.

5.25.2.9 EEPROM_u8EECR_REG [2/2] `#define EEPROM_u8EECR_REG (* ((volatile u8 *) 0x3F))`

EEPROM Register

Definition at line 27 of file registers.h.

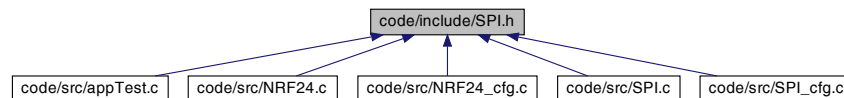
5.25.2.10 EEPROM_u8EEDR_REG `#define EEPROM_u8EEDR_REG (* ((volatile u8 *) 0x3E))`

Definition at line 24 of file registers.h.

5.26 code/include/SPI.h File Reference

Interfaces header file for [SPI.c](#).

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



Functions

- void [SPI_Init](#) (void)
- void [SPI_EnableInterrupt](#) (void(*ptrCallback)(void))
- void [SPI_DisableInterrupt](#) (void)
- [ERROR_STATUS_t SPI_SendByte](#) (const u8 data)
- [ERROR_STATUS_t SPI_SendString](#) (const u8 *str, u8 length)
- [ERROR_STATUS_t SPI_ReceiveByte](#) (u8 *const data)
- [ERROR_STATUS_t SPI_ReceiveString](#) (u8 *const str, u8 length)
- void [SPI_TrancieveByte](#) (const u8 dataToSend, u8 *const dataReceived)

5.26.1 Detailed Description

Interfaces header file for [SPI.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.26.2 Function Documentation

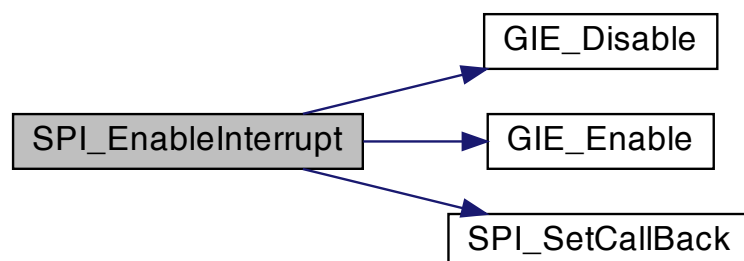
5.26.2.1 SPI_DisableInterrupt() `void SPI_DisableInterrupt (`
`void)`

Definition at line 238 of file SPI.c.

5.26.2.2 SPI_EnableInterrupt() `void SPI_EnableInterrupt (`
`void(*) (void) ptrCallback)`

Definition at line 229 of file SPI.c.

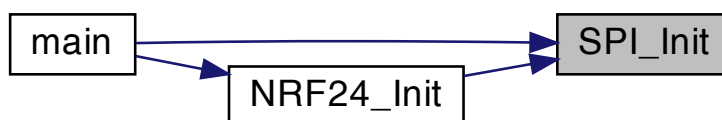
Here is the call graph for this function:



5.26.2.3 SPI_Init() `void SPI_Init (`
`void)`

Definition at line 212 of file SPI.c.

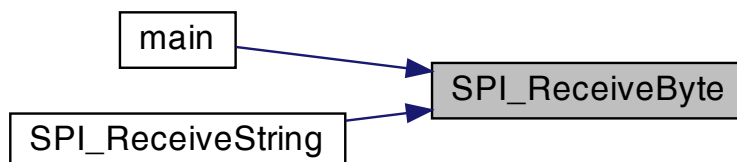
Here is the caller graph for this function:



5.26.2.4 SPI_ReceiveByte() `ERROR_STATUS_t SPI_ReceiveByte (`
`u8 *const data)`

Definition at line 273 of file SPI.c.

Here is the caller graph for this function:



5.26.2.5 SPI_ReceiveString() `ERROR_STATUS_t SPI_ReceiveString (`
`u8 *const str,`
`u8 length)`

Definition at line 293 of file SPI.c.

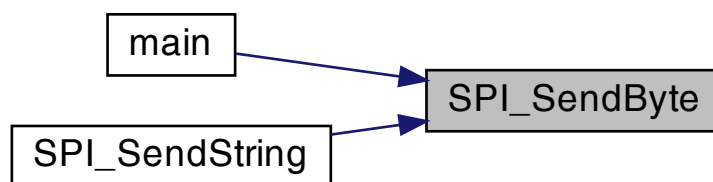
Here is the call graph for this function:



5.26.2.6 SPI_SendByte() `ERROR_STATUS_t SPI_SendByte (`
`const u8 data)`

Definition at line 242 of file SPI.c.

Here is the caller graph for this function:



5.26.2.7 SPI_SendString() `ERROR_STATUS_t SPI_SendString (`
`const u8 * str,`
`u8 length)`

Definition at line 258 of file SPI.c.

Here is the call graph for this function:



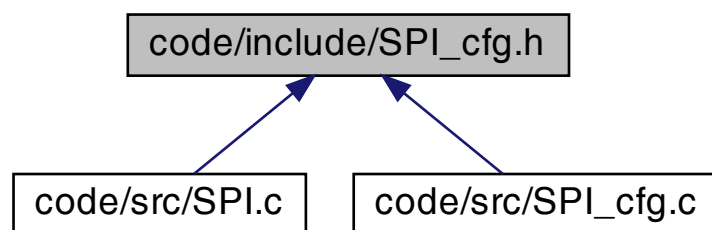
5.26.2.8 SPI_TrancieveByte() `void SPI_TrancieveByte (`
 `const u8 dataToSend,`
 `u8 *const dataReceived)`

Definition at line 285 of file SPI.c.

5.27 code/include/SPI_cfg.h File Reference

Configuration header file for [SPI.c](#).

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



Data Structures

- struct [SPI_PINS_t](#)
- struct [SPI_CONNECTIONS_t](#)
- struct [SPI_CONFIG_t](#)

Enumerations

- enum [SPI_MODE_t](#) { [SPI_MASTER](#) , [SPI_SLAVE](#) }
- enum [SPI_DATA_ORDER_t](#) { [SPI_DATA_ORDER_LSB_FIRST](#) , [SPI_DATA_ORDER_MSB_FIRST](#) }
- enum [SPI_CLOCK_MODE_t](#) { [SPI_MODE0](#) , [SPI_MODE1](#) , [SPI_MODE2](#) , [SPI_MODE3](#) }
- enum [SPI_PRESCALER_t](#) {
 [SPI_PRESCALER_2](#) , [SPI_PRESCALER_4](#) , [SPI_PRESCALER_8](#) , [SPI_PRESCALER_16](#) ,
 [SPI_PRESCALER_32](#) , [SPI_PRESCALER_64](#) , [SPI_PRESCALER_128](#) }
- enum [SPI_DOUBLE_SPEED_t](#) { [SPI_DOUBLE_SPEED_DISABLE](#) , [SPI_DOUBLE_SPEED_ENABLE](#) }

Variables

- [SPI_CONFIG_t](#) [SPI_Config](#)

5.27.1 Detailed Description

Configuration header file for [SPI.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.27.2 Enumeration Type Documentation

5.27.2.1 SPI_CLOCK_MODE_t enum [SPI_CLOCK_MODE_t](#)

Enumerator

SPI_MODE0	
SPI_MODE1	
SPI_MODE2	
SPI_MODE3	

Definition at line 22 of file SPI_cfg.h.

5.27.2.2 SPI_DATA_ORDER_t enum [SPI_DATA_ORDER_t](#)

Enumerator

SPI_DATA_ORDER_LSB_FIRST	
SPI_DATA_ORDER_MSB_FIRST	

Definition at line 17 of file SPI_cfg.h.

5.27.2.3 SPI_DOUBLE_SPEED_t enum [SPI_DOUBLE_SPEED_t](#)

Enumerator

SPI_DOUBLE_SPEED_DISABLE	
SPI_DOUBLE_SPEED_ENABLE	

Definition at line 39 of file SPI_cfg.h.

5.27.2.4 SPI_MODE_t enum [SPI_MODE_t](#)

Enumerator

SPI_MASTER	
SPI_SLAVE	

Definition at line 12 of file SPI_cfg.h.

5.27.2.5 SPI_PRESCALER_t enum [SPI_PRESCALER_t](#)

Enumerator

SPI_PRESCALER_2	
SPI_PRESCALER_4	
SPI_PRESCALER_8	
SPI_PRESCALER_16	
SPI_PRESCALER_32	
SPI_PRESCALER_64	
SPI_PRESCALER_128	

Definition at line 29 of file SPI_cfg.h.

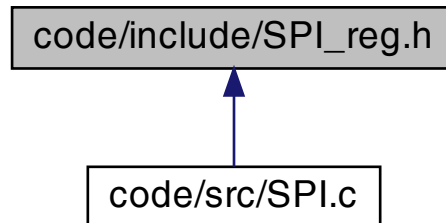
5.27.3 Variable Documentation**5.27.3.1 SPI_Config** [SPI_CONFIG_t](#) SPI_Config [extern]

Definition at line 17 of file SPI_cfg.c.

5.28 code/include/SPI_reg.h File Reference

SPI Registers of ATmega328p MCU.

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



Macros

- `#define SPCR` (* ((volatile `u8` *) 0x2D))
- `#define SPSR` (* ((volatile `u8` *) 0x2E))
- `#define SPDR` (* ((volatile `u8` *) 0x2F))

Enumerations

- `enum {`
 `SPR0` , `SPR1` , `CPHA` , `CPOL` ,
 `MSTR` , `DORD` , `SPE` , `SPIE` }
• `enum { SPI2X` , `WCOL` = 6 , `SPIF` }

5.28.1 Detailed Description

SPI Registers of ATmega328p MCU.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2021-07-31

5.28.2 Macro Definition Documentation

5.28.2.1 SPCR `#define SPCR (* ((volatile u8 *) 0x2D))`

Definition at line 11 of file SPI_reg.h.

5.28.2.2 SPDR `#define SPDR (* ((volatile u8 *) 0x2F))`

Definition at line 13 of file SPI_reg.h.

5.28.2.3 SPSR `#define SPSR (* ((volatile u8 *) 0x2E))`

Definition at line 12 of file SPI_reg.h.

5.28.3 Enumeration Type Documentation

5.28.3.1 anonymous enum `anonymous enum`

Enumerator

SPR0	
SPR1	
CPHA	
CPOL	
MSTR	
DORD	
SPE	
SPIE	

Definition at line 16 of file SPI_reg.h.

5.28.3.2 anonymous enum `anonymous enum`

Enumerator

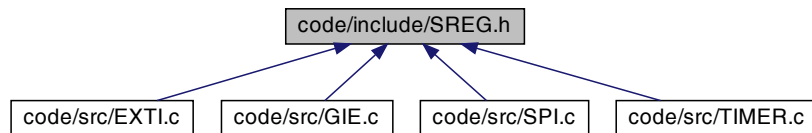
SPI2X	
WCOL	
SPIF	

Definition at line 27 of file SPI_reg.h.

5.29 code/include/SREG.h File Reference

Status Registers of ATmega328p MCU.

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



Macros

- `#define SREG (* ((volatile u8 *) 0x5F))`

Enumerations

- enum {
`C_BIT , Z_BIT , N_BIT , V_BIT ,`
`S_BIT , H_BIT , T_BIT , I_BIT }`

5.29.1 Detailed Description

Status Registers of ATmega328p MCU.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2021-07-31

5.29.2 Macro Definition Documentation

5.29.2.1 SREG `#define SREG (* ((volatile u8 *) 0x5F))`

Definition at line 11 of file SREG.h.

5.29.3 Enumeration Type Documentation

5.29.3.1 anonymous enum `anonymous enum`

Enumerator

C_BIT	
Z_BIT	
N_BIT	
V_BIT	
S_BIT	
H_BIT	
T_BIT	
I_BIT	

Definition at line 13 of file SREG.h.

5.30 code/include/STD_TYPES.h File Reference

Standard data types For AVR Microcontrollers.

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



Macros

- `#define NULL ((void *)0)`
- `#define NULL_BYTE ('\0')`

Typedefs

- typedef signed long long int [s64](#)
- typedef signed long int [s32](#)
- typedef signed short int [s16](#)
- typedef signed char [s8](#)
- typedef unsigned long long int [u64](#)
- typedef unsigned long int [u32](#)
- typedef unsigned short int [u16](#)
- typedef unsigned char [u8](#)
- typedef float [f32](#)
- typedef double [f64](#)
- typedef [u16](#) [size_t](#)

Enumerations

- enum [STATE_t](#) { [LOW](#) , [HIGH](#) , [NORMAL](#) }
- enum [ACTIVATION_STATUS_t](#) { [ACTIVE_LOW](#) , [ACTIVE_HIGH](#) }
- enum [BOOL_t](#) { [FALSE](#) , [TRUE](#) }
- enum [ERROR_STATUS_t](#) { [ERROR_YES](#) , [ERROR_NO](#) , [ERROR_TIMEOUT](#) }

5.30.1 Detailed Description

Standard data types For AVR Microcontrollers.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Date

2022-03-20

Version

1.0.0

5.30.2 Macro Definition Documentation

5.30.2.1 NULL `#define NULL ((void *)0)`

NULL pointer

Definition at line 58 of file STD_TYPES.h.

5.30.2.2 NULL_BYTE `#define NULL_BYTE ('\0')`

Definition at line 61 of file STD_TYPES.h.

5.30.3 Typedef Documentation

5.30.3.1 f32 `typedef float f32`

Definition at line 24 of file STD_TYPES.h.

5.30.3.2 f64 `typedef double f64`

Definition at line 25 of file STD_TYPES.h.

5.30.3.3 s16 typedef signed short int [s16](#)

Definition at line 14 of file STD_TYPES.h.

5.30.3.4 s32 typedef signed long int [s32](#)

Definition at line 13 of file STD_TYPES.h.

5.30.3.5 s64 typedef signed long long int [s64](#)

Definition at line 12 of file STD_TYPES.h.

5.30.3.6 s8 typedef signed char [s8](#)

Definition at line 15 of file STD_TYPES.h.

5.30.3.7 size_t typedef [u16](#) [size_t](#)

< This is a macro defined in the C standard library <stddef.h> for the size_t type size_t is an unsigned integer type of the result of the sizeof operator

Definition at line 29 of file STD_TYPES.h.

5.30.3.8 u16 typedef unsigned short int [u16](#)

Definition at line 20 of file STD_TYPES.h.

5.30.3.9 u32 typedef unsigned long int [u32](#)

Definition at line 19 of file STD_TYPES.h.

5.30.3.10 u64 typedef unsigned long long int [u64](#)

Definition at line 18 of file STD_TYPES.h.

5.30.3.11 u8 typedef unsigned char [u8](#)

Definition at line 21 of file STD_TYPES.h.

5.30.4 Enumeration Type Documentation

5.30.4.1 ACTIVATION_STATUS_t enum [ACTIVATION_STATUS_t](#)

Enumerator

ACTIVE_LOW	Active low means that the pin is pulled low when the pin is set to high
ACTIVE_HIGH	Active high means that the pin is pulled high when the pin is set to low

Definition at line 39 of file STD_TYPES.h.

5.30.4.2 **BOOL_t** enum [BOOL_t](#)

Enumerator

FALSE	
TRUE	

Definition at line 45 of file STD_TYPES.h.

5.30.4.3 **ERROR_STATUS_t** enum [ERROR_STATUS_t](#)

Enumerator

ERROR_YES	Error occurred
ERROR_NO	No error occurred
ERROR_TIMEOUT	Timeout occurred

Definition at line 50 of file STD_TYPES.h.

5.30.4.4 **STATE_t** enum [STATE_t](#)

Enumerator

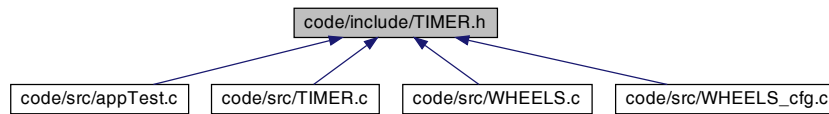
LOW	
HIGH	
NORMAL	

Definition at line 33 of file STD_TYPES.h.

5.31 code/include/TIMER.h File Reference

Interfaces header file for [TIMER.c](#).

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



Enumerations

- enum `TIMER_CLOCK_t` {
`NO_CLOCK` , `F_CPU_CLOCK` , `F_CPU_8` , `F_CPU_32` ,
`F_CPU_64` , `F_CPU_128` , `F_CPU_256` , `F_CPU_1024` ,
`F_CPU_EXT_CLK_FALLING` , `F_CPU_EXT_CLK_RISING` }
- enum `TIMER_MODE_t` {
`TIMER_MODE_NORMAL` , `TIMER_MODE_CTC` , `TIMER_MODE_CTC_OCR` , `TIMER_MODE_CTC_ICR` ,
`TIMER_MODE_FAST_PWM` , `TIMER_MODE_FAST_PWM_8` , `TIMER_MODE_FAST_PWM_9` , `TIMER_MODE_FAST_PWM_10` ,
`TIMER_MODE_FAST_PWM_ICR` , `TIMER_MODE_FAST_PWM_OCR` , `TIMER_MODE_PHASE_CORRECT_PWM` ,
`TIMER_MODE_PHASE_CORRECT_PWM_8` ,
`TIMER_MODE_PHASE_CORRECT_PWM_9` , `TIMER_MODE_PHASE_CORRECT_PWM_10` , `TIMER_MODE_PHASE_CORRECT_PWM_ICR` ,
`TIMER_MODE_PHASE_CORRECT_PWM_OCR` ,
`TIMER_MODE_PHASE_FREQ_CORRECT_ICR` , `TIMER_MODE_PHASE_FREQ_CORRECT_OCR` }
- enum `TIMER_OC_t` { `NO_OC` , `TOGGLE_OC` , `CLEAR_OC` , `SET_OC` }
- enum `TIMER_OCx_t` { `TIMER_OCA` , `TIMER_OCB` , `TIMER_OCC` }
- enum `PWM_t` {
`PWM_0` , `PWM_1` , `PWM_2` , `PWM_3` ,
`PWM_4` , `PWM_5` , `PWM_6` , `PWM_7` }

Functions

- void `TIMER0_Init` (`u8` initValue, `TIMER_CLOCK_t` clock, `TIMER_MODE_t` timerMode, `TIMER_OC_t` compareMode)
- void `TIMER0_SetCompareValue` (`u8` u8CompareValue)
- void `TIMER0_SetTimer` (`u8` u8TimerValue)
- void `TIMER0_EnableOverflowInterrupt` (void(*callBackPtr)(void))
- void `TIMER0_DisableOverflowInterrupt` (void)
- void `TIMER0_EnableCompareMatchInterrupt` (void(*callBackPtr)(void))
- void `TIMER0_DisableCompareMatchInterrupt` (void)
- `u8` `TIMER0_GetTimerValue` (void)
- void `PWM_Init` (`PWM_t` channel, `u32` frequency)
- void `PWM_Write` (`PWM_t` channel, `u8` dutyPercentage)
- void `TIMER1_Init` (`u16` initValue, `TIMER_CLOCK_t` clock, `TIMER_MODE_t` timerMode, `TIMER_OC_t` compareMode, `TIMER_OCx_t` OCx)
- void `TIMER1_SetCompareValue` (`u16` u16CompareValue, `TIMER_OCx_t` OCx)
- void `TIMER1_SetTimer` (`u16` u16TimerValue)
- void `TIMER1_EnableOverflowInterrupt` (void(*callBackPtr)(void))
- void `TIMER1_DisableOverflowInterrupt` (void)
- void `TIMER1_EnableCompareMatchInterrupt` (`TIMER_OCx_t` OCx, void(*callBackPtr)(void))
- void `TIMER1_DisableCompareMatchInterrupt` (`TIMER_OCx_t` OCx)
- void `TIMER1_EnableCaptureInterrupt` (void(*callBackPtr)(void))
- void `TIMER1_DisableCaptureInterrupt` (void)
- `u16` `TIMER1_GetTimerValue` (void)

5.31.1 Detailed Description

Interfaces header file for [TIMER.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.31.2 Enumeration Type Documentation

5.31.2.1 PWM_t enum [PWM_t](#)

Enumerator

PWM↔ _0	
PWM↔ _1	
PWM↔ _2	
PWM↔ _3	
PWM↔ _4	
PWM↔ _5	
PWM↔ _6	
PWM↔ _7	

Definition at line 63 of file [TIMER.h](#).

5.31.2.2 `TIMER_CLOCK_t` enum `TIMER_CLOCK_t`

Enumerator

NO_CLOCK	
F_CPU_CLOCK	
F_CPU_8	
F_CPU_32	
F_CPU_64	
F_CPU_128	
F_CPU_256	
F_CPU_1024	
F_CPU_EXT_CLK_FALLING	
F_CPU_EXT_CLK_RISING	

Definition at line 16 of file TIMER.h.

5.31.2.3 **TIMER_MODE_t** enum [TIMER_MODE_t](#)

Enumerator

TIMER_MODE_NORMAL	
TIMER_MODE_CTC	
TIMER_MODE_CTC_OCR	
TIMER_MODE_CTC_ICR	
TIMER_MODE_FAST_PWM	
TIMER_MODE_FAST_PWM_8	
TIMER_MODE_FAST_PWM_9	
TIMER_MODE_FAST_PWM_10	
TIMER_MODE_FAST_PWM_ICR	
TIMER_MODE_FAST_PWM_OCR	
TIMER_MODE_PHASE_CORRECT_PWM	
TIMER_MODE_PHASE_CORRECT_PWM_8	
TIMER_MODE_PHASE_CORRECT_PWM_9	
TIMER_MODE_PHASE_CORRECT_PWM_10	
TIMER_MODE_PHASE_CORRECT_PWM_ICR	
TIMER_MODE_PHASE_CORRECT_PWM_OCR	
TIMER_MODE_PHASE_FREQ_CORRECT_ICR	
TIMER_MODE_PHASE_FREQ_CORRECT_OCR	

Definition at line 29 of file TIMER.h.

5.31.2.4 **TIMER_OC_t** enum [TIMER_OC_t](#)

Enumerator

NO_OC	
TOGGLE_OC	
CLEAR_OC	
SET_OC	

Definition at line 50 of file TIMER.h.

5.31.2.5 `TIMER_OCx_t` enum `TIMER_OCx_t`

Enumerator

TIMER_OCA	
TIMER_OCB	
TIMER_OCC	

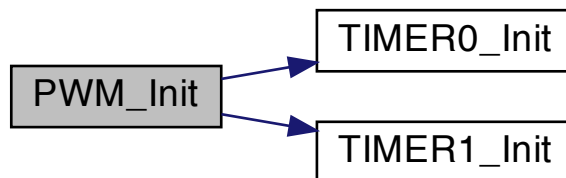
Definition at line 57 of file TIMER.h.

5.31.3 Function Documentation

5.31.3.1 `PWM_Init()` `void PWM_Init (` `PWM_t channel,` `u32 frequency)`

Definition at line 755 of file TIMER.c.

Here is the call graph for this function:



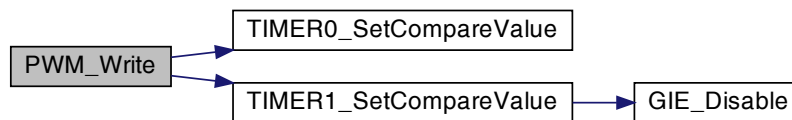
Here is the caller graph for this function:



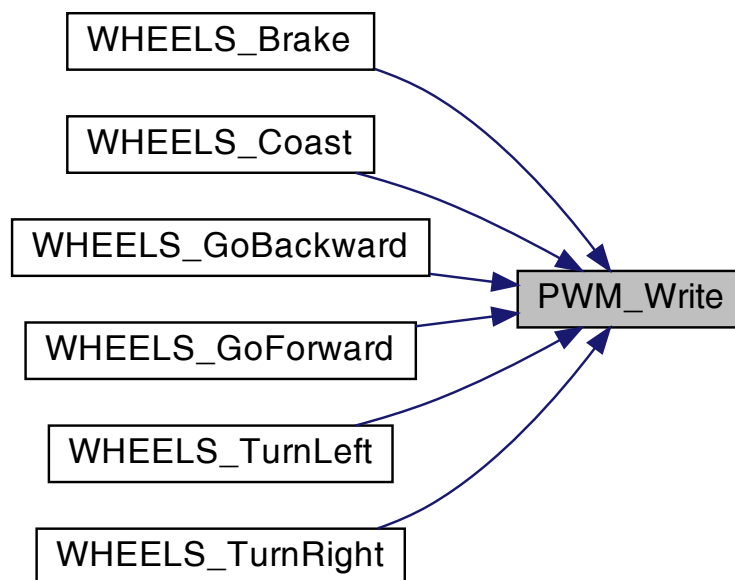
5.31.3.2 PWM_Write() `void PWM_Write (`
 PWM_t *channel*,
 u8 *dutyPercentage*)

Definition at line 793 of file `TIMER.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



5.31.3.3 TIMER0_DisableCompareMatchInterrupt() `void TIMER0_DisableCompareMatchInterrupt (`
 void)

Definition at line 565 of file `TIMER.c`.

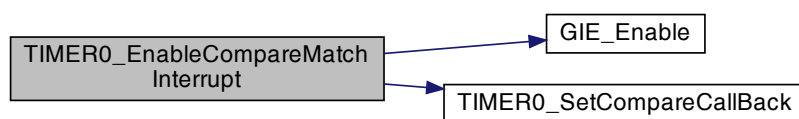
5.31.3.4 TIMERO_DisableOverflowInterrupt() `void TIMERO_DisableOverflowInterrupt (`
`void)`

Definition at line 550 of file TIMER.c.

5.31.3.5 TIMERO_EnableCompareMatchInterrupt() `void TIMERO_EnableCompareMatchInterrupt (`
`void(*) (void) callBackPtr)`

Definition at line 555 of file TIMER.c.

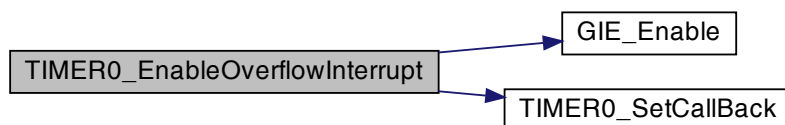
Here is the call graph for this function:



5.31.3.6 TIMERO_EnableOverflowInterrupt() `void TIMERO_EnableOverflowInterrupt (`
`void(*) (void) callBackPtr)`

Definition at line 540 of file TIMER.c.

Here is the call graph for this function:



5.31.3.7 TIMERO_GetTimerValue() `u8 TIMERO_GetTimerValue (`
`void)`

Definition at line 570 of file TIMER.c.

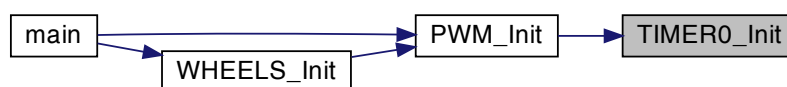
5.31.3.8 TIMER0_Init() `void TIMER0_Init (`
 u8 *initValue*,
 TIMER_CLOCK_t *clock*,
 TIMER_MODE_t *timerMode*,
 TIMER_OC_t *compareMode*)

Prototypes of Timer 0 functions

TIMER0 Implementations

Definition at line 518 of file TIMER.c.

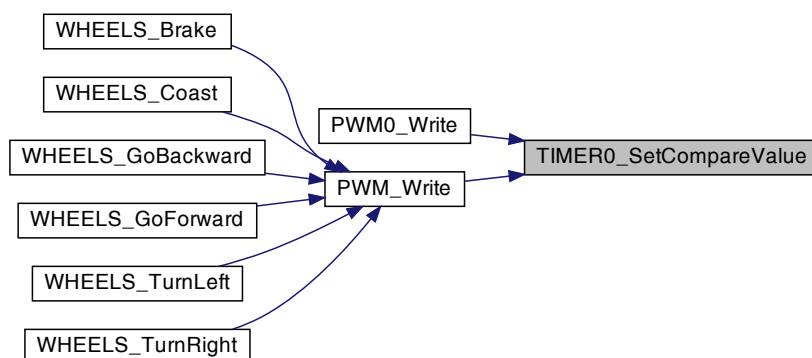
Here is the caller graph for this function:



5.31.3.9 TIMER0_SetCompareValue() `void TIMER0_SetCompareValue (`
 u8 *u8CompareValue*)

Definition at line 532 of file TIMER.c.

Here is the caller graph for this function:



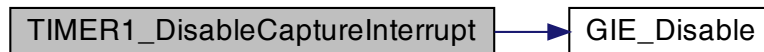
5.31.3.10 TIMER0_SetTimer() `void TIMER0_SetTimer (`
 u8 *u8TimerValue*)

Definition at line 536 of file TIMER.c.

5.31.3.11 TIMER1_DisableCaptureInterrupt() `void TIMER1_DisableCaptureInterrupt (`
`void)`

Definition at line 724 of file TIMER.c.

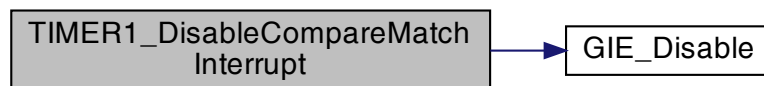
Here is the call graph for this function:



5.31.3.12 TIMER1_DisableCompareMatchInterrupt() `void TIMER1_DisableCompareMatchInterrupt (`
`TIMER_OCx_t OCx)`

Definition at line 692 of file TIMER.c.

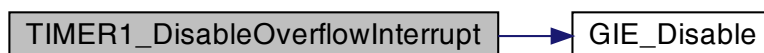
Here is the call graph for this function:



5.31.3.13 TIMER1_DisableOverflowInterrupt() `void TIMER1_DisableOverflowInterrupt (`
`void)`

Definition at line 659 of file TIMER.c.

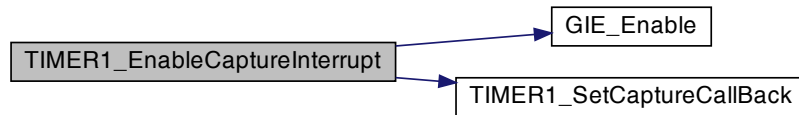
Here is the call graph for this function:



5.31.3.14 TIMER1_EnableCaptureInterrupt() `void TIMER1_EnableCaptureInterrupt (`
`void(*) (void) callBackPtr)`

Definition at line 714 of file TIMER.c.

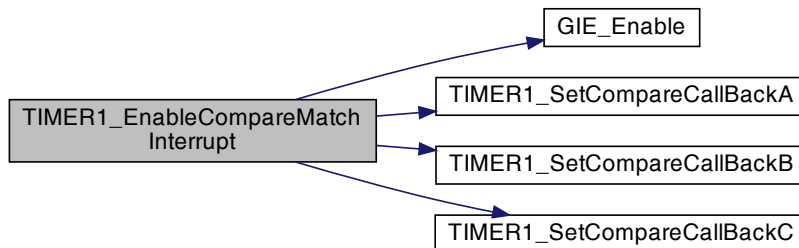
Here is the call graph for this function:



5.31.3.15 TIMER1_EnableCompareMatchInterrupt() `void TIMER1_EnableCompareMatchInterrupt (`
`TIMER_OCx_t OCx,`
`void(*) (void) callBackPtr)`

Definition at line 670 of file TIMER.c.

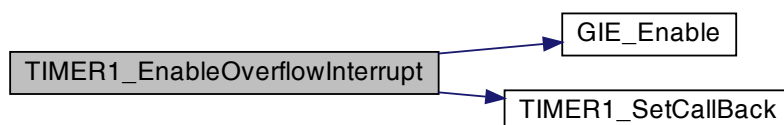
Here is the call graph for this function:



5.31.3.16 TIMER1_EnableOverflowInterrupt() `void TIMER1_EnableOverflowInterrupt (`
`void(*) (void) callBackPtr)`

Definition at line 649 of file TIMER.c.

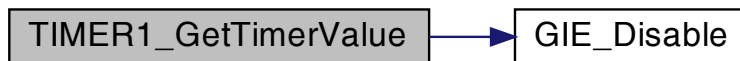
Here is the call graph for this function:



5.31.3.17 TIMER1_GetTimerValue() `u16 TIMER1_GetTimerValue (`
`void)`

Definition at line 735 of file TIMER.c.

Here is the call graph for this function:



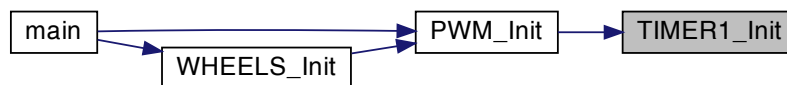
5.31.3.18 TIMER1_Init() `void TIMER1_Init (`
`u16 initValue,`
`TIMER_CLOCK_t clock,`
`TIMER_MODE_t timerMode,`
`TIMER_OC_t compareMode,`
`TIMER_OCx_t OCx)`

Prototypes of Timer 1 functions

TIMER1 Implementations

Definition at line 585 of file TIMER.c.

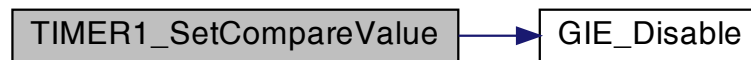
Here is the caller graph for this function:



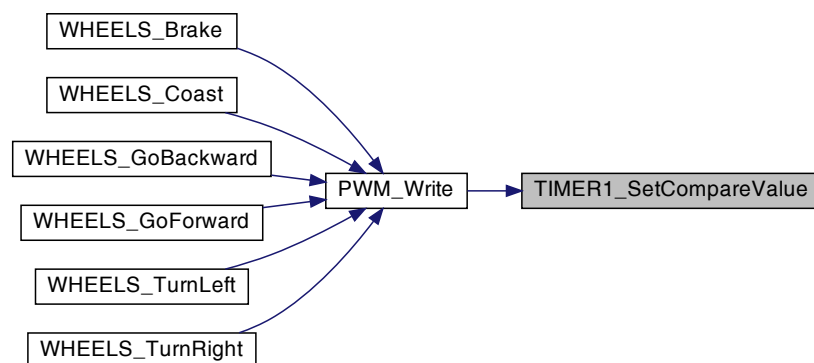
5.31.3.19 TIMER1_SetCompareValue() `void TIMER1_SetCompareValue (`
`u16 u16CompareValue,`
`TIMER_OCx_t OCx)`

Definition at line 613 of file TIMER.c.

Here is the call graph for this function:



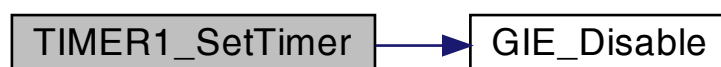
Here is the caller graph for this function:



5.31.3.20 TIMER1_SetTimer() `void TIMER1_SetTimer (`
`u16 u16TimerValue)`

Definition at line 638 of file TIMER.c.

Here is the call graph for this function:



5.32 code/include/TIMER_cfg.h File Reference

Configuration header file for [TIMER.c](#).

5.32.1 Detailed Description

Configuration header file for [TIMER.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

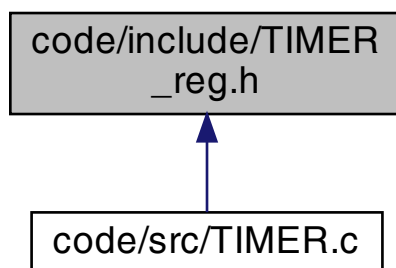
Copyright

Copyright (c) 2022

5.33 code/include/TIMER_reg.h File Reference

Registers of Timers of ATmega328p MCU.

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



Macros

- `#define TCCR0` (* ((volatile u8 *) 0x53)) /* Timer/Counter Control Register A */
- `#define TCNT0` (* ((volatile u8 *) 0x52)) /* Time Counter */
- `#define OCR0` (* ((volatile u8 *) 0x51)) /* Output Compare Register */
- `#define ASSR` (* ((volatile u8 *) 0x50)) /* Asynchronous Status Register */
- `#define TCCR1A` (* ((volatile u8 *) 0x4F))
- `#define TCCR1B` (* ((volatile u8 *) 0x4E))
- `#define TCCR1C` (* ((volatile u8 *) 0x7A))
- `#define TCNT1H` (* ((volatile u8 *) 0x4D))
- `#define TCNT1L` (* ((volatile u8 *) 0x4C))
- `#define ICR1H` (* ((volatile u8 *) 0x47))
- `#define ICR1L` (* ((volatile u8 *) 0x46))
- `#define OCR1AH` (* ((volatile u8 *) 0x4B))
- `#define OCR1AL` (* ((volatile u8 *) 0x4A))
- `#define OCR1BH` (* ((volatile u8 *) 0x49))
- `#define OCR1BL` (* ((volatile u8 *) 0x48))
- `#define OCR1CH` (* ((volatile u8 *) 0x79))
- `#define OCR1CL` (* ((volatile u8 *) 0x78))
- `#define TCCR2` (* ((volatile u8 *) 0x45))
- `#define TCNT2` (* ((volatile u8 *) 0x44))
- `#define OCR2` (* ((volatile u8 *) 0x43))
- `#define TCCR3A` (* ((volatile u8 *) 0x8B))
- `#define TCCR3B` (* ((volatile u8 *) 0x8A))
- `#define TCCR3C` (* ((volatile u8 *) 0x8C))
- `#define TCNT3H` (* ((volatile u8 *) 0x89))
- `#define TCNT3L` (* ((volatile u8 *) 0x88))
- `#define ICR3H` (* ((volatile u8 *) 0x81))
- `#define ICR3L` (* ((volatile u8 *) 0x80))
- `#define OCR3AH` (* ((volatile u8 *) 0x87))
- `#define OCR3AL` (* ((volatile u8 *) 0x86))
- `#define OCR3BH` (* ((volatile u8 *) 0x85))
- `#define OCR3BL` (* ((volatile u8 *) 0x84))
- `#define OCR3CH` (* ((volatile u8 *) 0x83))
- `#define OCR3CL` (* ((volatile u8 *) 0x82))
- `#define TIMER_u8TIMSK_REG` (* ((volatile u8 *) 0x57))
- `#define TIMER_u8TIFR_REG` (* ((volatile u8 *) 0x56))
- `#define TIMER_u8ETIMSK_REG` (* ((volatile u8 *) 0x7D))
- `#define TIMER_u8ETIFR_REG` (* ((volatile u8 *) 0x7C))

Enumerations

- enum {
 CS00 , CS01 , CS02 , WGM01 ,
 COM00 , COM01 , WGM00 , FOC0 }
- enum { TCR0UB , OCR0UB , TCN0UB , AS0 }
- enum {
 WGM10 , WGM11 , COM1C0 , COM1C1 ,
 COM1B0 , COM1B1 , COM1A0 , COM1A1 }
- enum {
 CS10 , CS11 , CS12 , WGM12 ,
 WGM13 , ICES1 = 6 , ICNC1 }
- enum { FOC1C = 5 , FOC1B , FOC1A }

- enum {
 CS20 , CS21 , CS22 , WGM21 ,
 COM20 , COM21 , WGM20 , FOC2 }
- enum {
 WGM30 , WGM31 , COM3C0 , COM3C1 ,
 COM3B0 , COM3B1 , COM3A0 , COM3A1 }
- enum {
 CS30 , CS31 , CS32 , WGM32 ,
 WGM33 , ICES3 = 6 , ICNC3 }
- enum { FOC3C = 5 , FOC3B , FOC3A }
- enum {
 TOIE0 , OCIE0 , TOIE1 , OCIE1B ,
 OCIE1A , TICIE1 , TOIE2 , OCIE2 }
- enum {
 OCIE1C , OCIE3C , TOIE3 , OCIE3B ,
 OCIE3A , TICIE3 }
- enum {
 TOV0 , OCF0 , TOV1 , OCF1B ,
 OCF1A , ICF1 , TOV2 , OCF2 }
- enum {
 OCF1C , OCF3C , TOV3 , OCF3B ,
 OCF3A , ICF3 }

5.33.1 Detailed Description

Registers of Timers of ATmega328p MCU.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2021-07-31

5.33.2 Macro Definition Documentation

5.33.2.1 ASSR `#define ASSR (* ((volatile u8 *) 0x50)) /* Asynchronous Status Register */`

Definition at line 17 of file `TIMER_reg.h`.

5.33.2.2 ICR1H `#define ICR1H (* ((volatile u8 *) 0x47))`

Definition at line 45 of file TIMER_reg.h.

5.33.2.3 ICR1L `#define ICR1L (* ((volatile u8 *) 0x46))`

Definition at line 46 of file TIMER_reg.h.

5.33.2.4 ICR3H `#define ICR3H (* ((volatile u8 *) 0x81))`

Definition at line 107 of file TIMER_reg.h.

5.33.2.5 ICR3L `#define ICR3L (* ((volatile u8 *) 0x80))`

Definition at line 108 of file TIMER_reg.h.

5.33.2.6 OCR0 `#define OCR0 (* ((volatile u8 *) 0x51)) /* Output Compare Register */`

Definition at line 16 of file TIMER_reg.h.

5.33.2.7 OCR1AH `#define OCR1AH (* ((volatile u8 *) 0x4B))`

Definition at line 47 of file TIMER_reg.h.

5.33.2.8 OCR1AL `#define OCR1AL (* ((volatile u8 *) 0x4A))`

Definition at line 48 of file TIMER_reg.h.

5.33.2.9 OCR1BH `#define OCR1BH (* ((volatile u8 *) 0x49))`

Definition at line 49 of file TIMER_reg.h.

5.33.2.10 OCR1BL `#define OCR1BL (* ((volatile u8 *) 0x48))`

Definition at line 50 of file `TIMER_reg.h`.

5.33.2.11 OCR1CH `#define OCR1CH (* ((volatile u8 *) 0x79))`

Definition at line 51 of file `TIMER_reg.h`.

5.33.2.12 OCR1CL `#define OCR1CL (* ((volatile u8 *) 0x78))`

Definition at line 52 of file `TIMER_reg.h`.

5.33.2.13 OCR2 `#define OCR2 (* ((volatile u8 *) 0x43))`

Definition at line 86 of file `TIMER_reg.h`.

5.33.2.14 OCR3AH `#define OCR3AH (* ((volatile u8 *) 0x87))`

Definition at line 109 of file `TIMER_reg.h`.

5.33.2.15 OCR3AL `#define OCR3AL (* ((volatile u8 *) 0x86))`

Definition at line 110 of file `TIMER_reg.h`.

5.33.2.16 OCR3BH `#define OCR3BH (* ((volatile u8 *) 0x85))`

Definition at line 111 of file `TIMER_reg.h`.

5.33.2.17 OCR3BL `#define OCR3BL (* ((volatile u8 *) 0x84))`

Definition at line 112 of file `TIMER_reg.h`.

5.33.2.18 OCR3CH `#define OCR3CH (* ((volatile u8 *) 0x83))`

Definition at line 113 of file TIMER_reg.h.

5.33.2.19 OCR3CL `#define OCR3CL (* ((volatile u8 *) 0x82))`

Definition at line 114 of file TIMER_reg.h.

5.33.2.20 TCCR0 `#define TCCR0 (* ((volatile u8 *) 0x53)) /* Timer/Counter Control Register A */`

Timer/Counter0 Register (8-bit)

Definition at line 14 of file TIMER_reg.h.

5.33.2.21 TCCR1A `#define TCCR1A (* ((volatile u8 *) 0x4F))`

Timer/Counter1 Register (16-bit)

Definition at line 40 of file TIMER_reg.h.

5.33.2.22 TCCR1B `#define TCCR1B (* ((volatile u8 *) 0x4E))`

Definition at line 41 of file TIMER_reg.h.

5.33.2.23 TCCR1C `#define TCCR1C (* ((volatile u8 *) 0x7A))`

Definition at line 42 of file TIMER_reg.h.

5.33.2.24 TCCR2 `#define TCCR2 (* ((volatile u8 *) 0x45))`

Timer/Counter2 Register (8-bit)

Definition at line 84 of file TIMER_reg.h.

5.33.2.25 TCCR3A `#define TCCR3A (* ((volatile u8 *) 0x8B))`

Timer/Counter3 Register (16-bit)

Definition at line 102 of file `TIMER_reg.h`.

5.33.2.26 TCCR3B `#define TCCR3B (* ((volatile u8 *) 0x8A))`

Definition at line 103 of file `TIMER_reg.h`.

5.33.2.27 TCCR3C `#define TCCR3C (* ((volatile u8 *) 0x8C))`

Definition at line 104 of file `TIMER_reg.h`.

5.33.2.28 TCNT0 `#define TCNT0 (* ((volatile u8 *) 0x52)) /* Time Counter */`

Definition at line 15 of file `TIMER_reg.h`.

5.33.2.29 TCNT1H `#define TCNT1H (* ((volatile u8 *) 0x4D))`

Definition at line 43 of file `TIMER_reg.h`.

5.33.2.30 TCNT1L `#define TCNT1L (* ((volatile u8 *) 0x4C))`

Definition at line 44 of file `TIMER_reg.h`.

5.33.2.31 TCNT2 `#define TCNT2 (* ((volatile u8 *) 0x44))`

Definition at line 85 of file `TIMER_reg.h`.

5.33.2.32 TCNT3H `#define TCNT3H (* ((volatile u8 *) 0x89))`

Definition at line 105 of file `TIMER_reg.h`.

5.33.2.33 TCNT3L `#define TCNT3L (* ((volatile u8 *) 0x88))`

Definition at line 106 of file TIMER_reg.h.

5.33.2.34 TIMER_u8ETIFR_REG `#define TIMER_u8ETIFR_REG (* ((volatile u8 *) 0x7C))`

Definition at line 149 of file TIMER_reg.h.

5.33.2.35 TIMER_u8ETIMSK_REG `#define TIMER_u8ETIMSK_REG (* ((volatile u8 *) 0x7D))`

Definition at line 148 of file TIMER_reg.h.

5.33.2.36 TIMER_u8TIFR_REG `#define TIMER_u8TIFR_REG (* ((volatile u8 *) 0x56))`

Definition at line 147 of file TIMER_reg.h.

5.33.2.37 TIMER_u8TIMSK_REG `#define TIMER_u8TIMSK_REG (* ((volatile u8 *) 0x57))`

Timers/Counters Common Registers

Definition at line 146 of file TIMER_reg.h.

5.33.3 Enumeration Type Documentation

5.33.3.1 anonymous enum `anonymous enum`

Enumerator

CS00	
CS01	
CS02	
WGM01	
COM00	
COM01	
WGM00	
FOC0	

Definition at line 19 of file TIMER_reg.h.

5.33.3.2 anonymous enum `anonymous enum`**Enumerator**

TCR0UB	
OCR0UB	
TCN0UB	
AS0	

Definition at line 30 of file `TIMER_reg.h`.

5.33.3.3 anonymous enum `anonymous enum`**Enumerator**

WGM10	
WGM11	
COM1C0	
COM1C1	
COM1B0	
COM1B1	
COM1A0	
COM1A1	

Definition at line 54 of file `TIMER_reg.h`.

5.33.3.4 anonymous enum `anonymous enum`**Enumerator**

CS10	
CS11	
CS12	
WGM12	
WGM13	
ICES1	
ICNC1	

Definition at line 65 of file `TIMER_reg.h`.

5.33.3.5 anonymous enum `anonymous enum`

Enumerator

FOC1C	
FOC1B	
FOC1A	

Definition at line 75 of file TIMER_reg.h.

5.33.3.6 anonymous enum `anonymous enum`**Enumerator**

CS20	
CS21	
CS22	
WGM21	
COM20	
COM21	
WGM20	
FOC2	

Definition at line 88 of file TIMER_reg.h.

5.33.3.7 anonymous enum `anonymous enum`**Enumerator**

WGM30	
WGM31	
COM3C0	
COM3C1	
COM3B0	
COM3B1	
COM3A0	
COM3A1	

Definition at line 116 of file TIMER_reg.h.

5.33.3.8 anonymous enum `anonymous enum`**Enumerator**

CS30	
------	--

Enumerator

CS31	
CS32	
WGM32	
WGM33	
ICES3	
ICNC3	

Definition at line 127 of file TIMER_reg.h.

5.33.3.9 anonymous enum `anonymous_enum`

Enumerator

FOC3C	
FOC3B	
FOC3A	

Definition at line 137 of file TIMER_reg.h.

5.33.3.10 anonymous enum `anonymous_enum`

Enumerator

TOIE0	
OCIE0	
TOIE1	
OCIE1B	
OCIE1A	
TICIE1	
TOIE2	
OCIE2	

Definition at line 151 of file TIMER_reg.h.

5.33.3.11 anonymous enum `anonymous_enum`

Enumerator

OCIE1C	
OCIE3C	
TOIE3	

Enumerator

OCIE3B	
OCIE3A	
TICIE3	

Definition at line 163 of file TIMER_reg.h.

5.33.3.12 anonymous enum `anonymous enum`**Enumerator**

TOV0	
OCF0	
TOV1	
OCF1B	
OCF1A	
ICF1	
TOV2	
OCF2	

Definition at line 172 of file TIMER_reg.h.

5.33.3.13 anonymous enum `anonymous enum`**Enumerator**

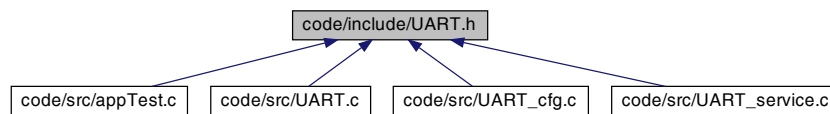
OCF1C	
OCF3C	
TOV3	
OCF3B	
OCF3A	
ICF3	

Definition at line 183 of file TIMER_reg.h.

5.34 code/include/UART.h File Reference

Interfaces header file for [UART.c](#).

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



Functions

- void [UART0_Init](#) (void)
Initialize UART module 0 as configured in [UART_cfg.h](#) and [UART_cfg.c](#).
- void [UART1_Init](#) (void)
Initialize UART module 1 as configured in [UART_cfg.h](#) and [UART_cfg.c](#).
- void [UART0_Disable](#) (void)
Disable UART module 0 if it is enabled.
- void [UART1_Disable](#) (void)
Disable UART module 1 if it is enabled.
- void [UART0_Enable](#) (void)
Enable UART module 0 if it is disabled previously by [UART0_Disable\(\)](#)
- void [UART1_Enable](#) (void)
Enable UART module 1 if it is disabled previously by [UART1_Disable\(\)](#)
- void [UART0_SendByte](#) (const [u8](#) data)
Send a byte using UART module 0 Synchronously.
- void [UART0_SendByte_NoBlock](#) (const [u8](#) data)
Send a byte using UART module 0 Asynchronously.
- void [UART1_SendByte](#) (const [u8](#) data)
Send a byte using UART module 1.
- void [UART1_SendByte_NoBlock](#) (const [u8](#) data)
Send a byte using UART module 1 Asynchronously.
- void [UART0_Send9BitData](#) (const [u16](#) data)
Send 9 bits using UART module 0.
- void [UART0_Send9BitData_NoBlock](#) (const [u16](#) data)
Send 9 bits using UART module 0 Asynchronously.
- void [UART1_Send9BitData](#) (const [u16](#) data)
Send 9 bits using UART module 1.
- void [UART1_Send9BitData_NoBlock](#) (const [u16](#) data)
Send 9 bits using UART module 1 Asynchronously.
- [STATE_t](#) [UART0_Available](#) (void)
Check if there is a byte available in UART module 0.
- [STATE_t](#) [UART1_Available](#) (void)
Check if there is a byte available in UART module 1.
- [ERROR_STATUS_t](#) [UART0_ReceiveByte](#) ([u8](#) *data)
Receive a byte using UART module 0.
- [ERROR_STATUS_t](#) [UART0_ReceiveByte_NoBlock](#) ([u8](#) *data)
Receive a byte using UART module 0 Asynchronously.
- [ERROR_STATUS_t](#) [UART1_ReceiveByte](#) ([u8](#) *data)
Receive a byte using UART module 1.
- [ERROR_STATUS_t](#) [UART1_ReceiveByte_NoBlock](#) ([u8](#) *data)

- Receive a byte using UART module 1 Asynchronously.*

 - [ERROR_STATUS_t UART0_Receive9BitData \(u16 *data\)](#)

Receive a string of 16 bits using UART module 0 (9 bits data)

- [ERROR_STATUS_t UART1_Receive9BitData \(u16 *data\)](#)

Receive a string of 16 bits using UART module 1 (9 bits data)

- void [UART0_Flush](#) (void)

Flush the receive buffer of UART module 0.

- void [UART1_Flush](#) (void)

Flush the receive buffer of UART module 1.

- void [UART0_RX_InterruptEnable](#) (void(*const ptrCallback)(void))

Enable the receive interrupt of UART module 0 with passing the function pointer to the interrupt service routine.

- void [UART1_RX_InterruptEnable](#) (void(*const ptrCallback)(void))

Enable the receive interrupt of UART module 1 with passing the function pointer to the interrupt service routine.

- void [UART0_RX_InterruptDisable](#) (void)

Disable the receive interrupt of UART module 0.

- void [UART1_RX_InterruptDisable](#) (void)

Disable the receive interrupt of UART module 1.

- void [UART1_TX_InterruptEnable](#) (void(*const ptrCallback)(void))

Enable the transmit interrupt of UART module 0 with passing the function pointer to the interrupt service routine.

- void [UART0_TX_InterruptEnable](#) (void(*const ptrCallback)(void))

Enable the transmit interrupt of UART module 1 with passing the function pointer to the interrupt service routine.

- void [UART0_TX_InterruptDisable](#) (void)

Disable the transmit interrupt of UART module 0.

- void [UART1_TX_InterruptDisable](#) (void)

Disable the transmit interrupt of UART module 1.

- void [UART0_UDRE_InterruptEnable](#) (void(*const ptrCallback)(void))

Enable the interrupt of Data Register Empty of UART module 0 with passing the function pointer to the interrupt service routine.

- void [UART1_UDRE_InterruptEnable](#) (void(*const ptrCallback)(void))

Enable the interrupt of Data Register Empty of UART module 1 with passing the function pointer to the interrupt service routine.

- void [UART0_UDRE_InterruptDisable](#) (void)

Disable the interrupt of Data Register Empty of UART module 0.

- void [UART1_UDRE_InterruptDisable](#) (void)

Disable the interrupt of Data Register Empty of UART module 1.

5.34.1 Detailed Description

Interfaces header file for [UART.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-03

Copyright

Copyright (c) 2022

5.34.2 Function Documentation

5.34.2.1 `UART0_Available()` `STATE_t UART0_Available (void)`

Check if there is a byte available in UART module 0.

Returns

HIGH if there is a byte available, LOW otherwise

Definition at line 625 of file UART.c.

Here is the caller graph for this function:



5.34.2.2 `UART0_Disable()` `void UART0_Disable (void)`

Disable UART module 0 if it is enabled.

Definition at line 546 of file UART.c.

5.34.2.3 `UART0_Enable()` `void UART0_Enable (void)`

Enable UART module 0 if it is disabled previously by [UART0_Disable\(\)](#)

Definition at line 535 of file UART.c.

5.34.2.4 UART0_Flush() `void UART0_Flush (`
`void)`

Flush the receive buffer of UART module 0.

For Example: `UART0_Flush();` will flush the receive buffer of UART module 0

Flush the receive buffer of UART module 0.

Definition at line 640 of file UART.c.

5.34.2.5 UART0_Init() `void UART0_Init (`
`void)`

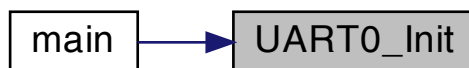
Initialize UART module 0 as configured in [UART_cfg.h](#) and [UART_cfg.c](#).

It configure UART module 0 according to [UART_cfg.h](#) and [UART_cfg.c](#)

and then enable UART module 0

Definition at line 482 of file UART.c.

Here is the caller graph for this function:



5.34.2.6 UART0_Receive9BitData() `ERROR_STATUS_t UART0_Receive9BitData (`
`u16 * data)`

Receive a string of 16 bits using UART module 0 (9 bits data)

For Example: `UART0_Receive9BitString(string);` will receive a string of 16 bits

from UART module 0 and store it in variable `<string>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 599 of file UART.c.

5.34.2.7 UART0_ReceiveByte() `ERROR_STATUS_t UART0_ReceiveByte (u8 * data)`

Receive a byte using UART module 0.

For Example: `UART0_ReceiveByte(&data);` will receive a byte from UART module 0

and store it in variable `<data>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 591 of file `UART.c`.

Here is the caller graph for this function:



5.34.2.8 UART0_ReceiveByte_NoBlock() `ERROR_STATUS_t UART0_ReceiveByte_NoBlock (u8 * data)`

Receive a byte using UART module 0 Asynchronously.

For Example: `UART0_ReceiveByte(&data);` will receive a byte from UART module 0

and store it in variable `<data>` without checking receive complete flag and without blocking the calling thread

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 608 of file `UART.c`.

5.34.2.9 UART0_RX_InterruptDisable() `void UART0_RX_InterruptDisable (void)`

Disable the receive interrupt of UART module 0.

For Example: `UART0_RX_InterruptDisable();` will disable the receive interrupt

of UART module 0

Definition at line 714 of file `UART.c`.

5.34.2.10 UART0_RX_InterruptEnable() `void UART0_RX_InterruptEnable (void(*) (void) ptrCallback)`

Enable the receive interrupt of UART module 0 with passing the function pointer to the interrupt service routine.

Parameters

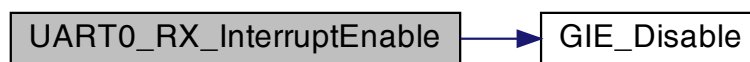
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The interrupt is triggered when a byte is received

Definition at line 648 of file UART.c.

Here is the call graph for this function:



5.34.2.11 UART0_Send9BitData() `void UART0_Send9BitData (`
`const u16 data)`

Send 9 bits using UART module 0.

For Example: `UART0_Send9Bits(0x123);` will send 9 bits 0x123 to UART module 0

Definition at line 565 of file UART.c.

5.34.2.12 UART0_Send9BitData_NoBlock() `void UART0_Send9BitData_NoBlock (`
`const u16 data)`

Send 9 bits using UART module 0 Asynchronously.

For Example: `UART0_Send9Bits_NoBlock(0x123);` will send 9 bits 0x123 to UART module 0

without blocking the calling thread

Definition at line 582 of file UART.c.

5.34.2.13 UART0_SendByte() `void UART0_SendByte (`
`const u8 data)`

Send a byte using UART module 0 Synchronously.

For Example: `UART0_SendByte('a');` will send character 'a' to UART module 0

Definition at line 557 of file UART.c.

Here is the caller graph for this function:



5.34.2.14 UART0_SendByte_NoBlock() `void UART0_SendByte_NoBlock (`
`const u8 data)`

Send a byte using UART module 0 Asynchronously.

For Example: `UART0_SendByte_NoBlock('a');` will send character 'a' to UART module 0
without blocking the calling thread

Definition at line 574 of file UART.c.

5.34.2.15 UART0_TX_InterruptDisable() `void UART0_TX_InterruptDisable (`
`void)`

Disable the transmit interrupt of UART module 0.

For Example: `UART0_TX_InterruptDisable();` will disable the transmit interrupt
of UART module 0

Definition at line 722 of file UART.c.

5.34.2.16 UART0_TX_InterruptEnable() `void UART0_TX_InterruptEnable (`
`void(*) (void) ptrCallback)`

Enable the transmit interrupt of UART module 1 with passing the function pointer to the interrupt service routine.

Parameters

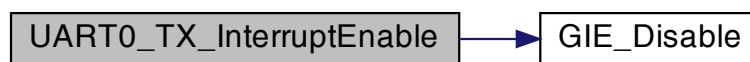
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The transmit interrupt will be triggered after transmission complete.

Definition at line 681 of file UART.c.

Here is the call graph for this function:



5.34.2.17 UART0_UDRE_InterruptDisable() `void UART0_UDRE_InterruptDisable (void)`

Disable the interrupt of Data Register Empty of UART module 0.

For Example: `UART0_UDRE_InterruptDisable();` will disable the interrupt of Data Register Empty of UART module 0

Definition at line 730 of file UART.c.

5.34.2.18 UART0_UDRE_InterruptEnable() `void UART0_UDRE_InterruptEnable (void(*) (void) ptrCallback)`

Enable the interrupt of Data Register Empty of UART module 0 with passing the function pointer to the interrupt service routine.

Parameters

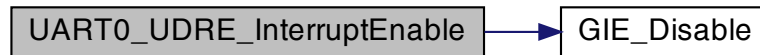
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The interrupt is triggered when the data register is empty

Definition at line 692 of file UART.c.

Here is the call graph for this function:



5.34.2.19 UART1_Available() `STATE_t UART1_Available (void)`

Check if there is a byte available in UART module 1.

Returns

HIGH if there is a byte available, LOW otherwise

Definition at line 629 of file UART.c.

Here is the caller graph for this function:



5.34.2.20 UART1_Disable() `void UART1_Disable (void)`

Disable UART module 1 if it is enabled.

Definition at line 551 of file UART.c.

5.34.2.21 UART1_Enable() `void UART1_Enable (`
`void)`

Enable UART module 1 if it is disabled previously by [UART1_Disable\(\)](#)

It enable UART module 1 if it is disabled previously by [UART1_Disable\(\)](#)

Definition at line 540 of file UART.c.

5.34.2.22 UART1_Flush() `void UART1_Flush (`
`void)`

Flush the receive buffer of UART module 1.

For Example: `UART1_Flush();` will flush the receive buffer of UART module 1

Definition at line 644 of file UART.c.

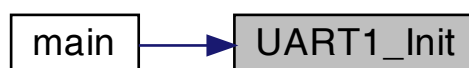
5.34.2.23 UART1_Init() `void UART1_Init (`
`void)`

Initialize UART module 1 as configured in [UART_cfg.h](#) and [UART_cfg.c](#).

It configure UART module 1 accorring to [UART_cfg.h](#) and [UART_cfg.c](#)
and then enable UART module 1

Definition at line 508 of file UART.c.

Here is the caller graph for this function:



5.34.2.24 UART1_Receive9BitData() `ERROR_STATUS_t UART1_Receive9BitData (`
`u16 * data)`

Receive a string of 16 bits using UART module 1 (9 bits data)

For Example: `UART1_Receive9BitString(string);` will receive a string of 16 bits
from UART module 1 and store it in variable `<string>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 603 of file UART.c.

5.34.2.25 UART1_ReceiveByte() `ERROR_STATUS_t UART1_ReceiveByte (`
`u8 * data)`

Receive a byte using UART module 1.

For Example: `UART1_ReceiveByte(&data);` will receive a byte from UART module 1
and store it in variable `<data>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 595 of file UART.c.

Here is the caller graph for this function:



5.34.2.26 UART1_ReceiveByte_NoBlock() `ERROR_STATUS_t UART1_ReceiveByte_NoBlock (u8 * data)`

Receive a byte using UART module 1 Asynchronously.

For Example: `UART1_ReceiveByte(&data);` will receive a byte from UART module 1

and store it in variable `<data>` without checking receive complete flag and without blocking the calling thread

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 612 of file UART.c.

5.34.2.27 UART1_RX_InterruptDisable() `void UART1_RX_InterruptDisable (void)`

Disable the receive interrupt of UART module 1.

For Example: `UART1_RX_InterruptDisable();` will disable the receive interrupt

of UART module 1

Definition at line 718 of file UART.c.

5.34.2.28 UART1_RX_InterruptEnable() `void UART1_RX_InterruptEnable (void(*) (void) ptrCallback)`

Enable the receive interrupt of UART module 1 with passing the function pointer to the interrupt service routine.

Parameters

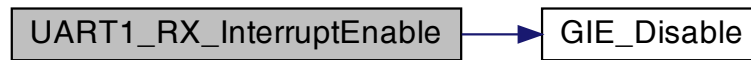
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The interrupt is triggered when a byte is received

Definition at line 659 of file UART.c.

Here is the call graph for this function:



5.34.2.29 UART1_Send9BitData() `void UART1_Send9BitData (`
`const u16 data)`

Send 9 bits using UART module 1.

Parameters

in	<i>data</i>	9 bits data to be sent. Should be between 0 and 0x1FF
----	-------------	---

Note

Size of data should be 2 Bytes to be able to send 9 bits data

For Example: `UART1_Send9Bits(0x123);` will send 9 bits 0x123 to UART module 1

Definition at line 569 of file UART.c.

5.34.2.30 UART1_Send9BitData_NoBlock() `void UART1_Send9BitData_NoBlock (`
`const u16 data)`

Send 9 bits using UART module 1 Asynchronously.

Parameters

in	<i>data</i>	9 bits data to be sent. Should be between 0 and 0x1FF
----	-------------	---

Note

Size of data should be 2 Bytes to be able to send 9 bits data

For Example: `UART1_Send9Bits_NoBlock(0x123);` will send 9 bits 0x123 to UART module 1

without blocking the calling thread

Definition at line 586 of file UART.c.

5.34.2.31 UART1_SendByte() `void UART1_SendByte (`
`const u8 data)`

Send a byte using UART module 1.

For Example: `UART1_SendByte('y');` will send character 'y' to UART module 1

Definition at line 561 of file UART.c.

Here is the caller graph for this function:



5.34.2.32 UART1_SendByte_NoBlock() `void UART1_SendByte_NoBlock (`
`const u8 data)`

Send a byte using UART module 1 Asynchronously.

For Example: `UART1_SendByte_NoBlock('a');` will send character 'a' to UART module 1

without blocking the calling thread

Definition at line 578 of file UART.c.

5.34.2.33 UART1_TX_InterruptDisable() `void UART1_TX_InterruptDisable (`
`void)`

Disable the transmit interrupt of UART module 1.

For Example: `UART1_TX_InterruptDisable();` will disable the transmit interrupt

of UART module 1

Definition at line 726 of file UART.c.

5.34.2.34 UART1_TX_InterruptEnable() `void UART1_TX_InterruptEnable (`
`void(*) (void) ptrCallback)`

Enable the transmit interrupt of UART module 0 with passing the function pointer to the interrupt service routine.

Parameters

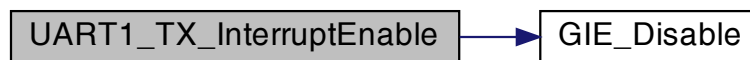
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The transmit interrupt will be triggered after transmission complete.

Definition at line 670 of file UART.c.

Here is the call graph for this function:



5.34.2.35 UART1_UDRE_InterruptDisable() `void UART1_UDRE_InterruptDisable (void)`

Disable the interrupt of Data Register Empty of UART module 1.

For Example: `UART1_UDRE_InterruptDisable();` will disable the interrupt of Data Register Empty of UART module 1

Definition at line 734 of file UART.c.

5.34.2.36 UART1_UDRE_InterruptEnable() `void UART1_UDRE_InterruptEnable (void(*) (void) ptrCallback)`

Enable the interrupt of Data Register Empty of UART module 1 with passing the function pointer to the interrupt service routine.

Parameters

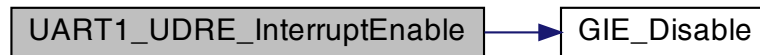
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The interrupt is triggered when the data register is empty

Definition at line 703 of file UART.c.

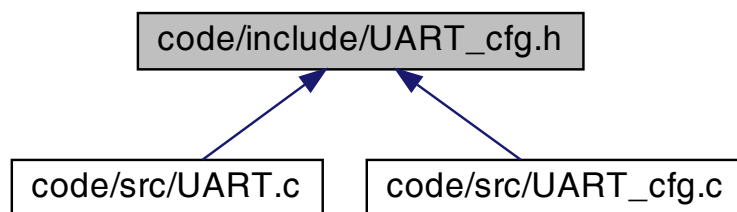
Here is the call graph for this function:



5.35 code/include/UART_cfg.h File Reference

Configuration header file for [UART.c](#).

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



Data Structures

- struct [UART_CFG_t](#)

Macros

- `#define` [UART_TIMEOUT_CYCLE_COUNT](#) (16000)

Enumerations

- enum [UART_DATA_BITS_t](#) { [UART_DATA_5_BITS](#) , [UART_DATA_6_BITS](#) , [UART_DATA_7_BITS](#) , [UART_DATA_8_BITS](#) , [UART_DATA_9_BITS](#) }
- enum [UART_STOP_BITS_t](#) { [UART_STOP_1_BIT](#) , [UART_STOP_2_BIT](#) }
- enum [UART_PARITY_t](#) { [UART_PARITY_DISABLE](#) , [UART_PARITY_ODD](#) , [UART_PARITY_EVEN](#) }
- enum [UART_MODE_t](#) { [UART_MODE_ASYNCHRONOUS_NORMAL](#) , [UART_MODE_ASYNCHRONOUS_DOUBLE_SPEED](#) , [UART_MODE_SYNCHRONOUS_MASTER](#) , [UART_MODE_SYNCHRONOUS_SLAVE](#) }
- enum [UART_MODE_TYPE_t](#) { [UART_MODE_TX](#) , [UART_MODE_RX](#) , [UART_MODE_TX_RX](#) }
- enum [UART_CLOCK_POLARITY_t](#) { [UART_RISING_EDGE_CLOCK](#) , [UART_FALLING_EDGE_CLOCK](#) }

Variables

- [UART_CFG_t UART0_Configs](#)
- [UART_CFG_t UART1_Configs](#)

5.35.1 Detailed Description

Configuration header file for [UART.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.35.2 Macro Definition Documentation

5.35.2.1 **UART_TIMEOUT_CYCLE_COUNT** `#define UART_TIMEOUT_CYCLE_COUNT (16000)`

Definition at line 12 of file [UART_cfg.h](#).

5.35.3 Enumeration Type Documentation

5.35.3.1 **UART_CLOCK_POLARITY_t** `enum UART_CLOCK_POLARITY_t`

Enumerator

UART_RISING_EDGE_CLOCK	
UART_FALLING_EDGE_CLOCK	

Definition at line 49 of file [UART_cfg.h](#).

5.35.3.2 UART_DATA_BITS_t enum [UART_DATA_BITS_t](#)

Enumerator

UART_DATA_5_BITS	
UART_DATA_6_BITS	
UART_DATA_7_BITS	
UART_DATA_8_BITS	
UART_DATA_9_BITS	

Definition at line 17 of file UART_cfg.h.

5.35.3.3 UART_MODE_t enum [UART_MODE_t](#)

Enumerator

UART_MODE_ASYNCHRONOUS_NORMAL	
UART_MODE_ASYNCHRONOUS_DOUBLE_SPEED	
UART_MODE_SYNCHRONOUS_MASTER	
UART_MODE_SYNCHRONOUS_SLAVE	

Definition at line 36 of file UART_cfg.h.

5.35.3.4 UART_MODE_TYPE_t enum [UART_MODE_TYPE_t](#)

Enumerator

UART_MODE_TX	
UART_MODE_RX	
UART_MODE_TX_RX	

Definition at line 43 of file UART_cfg.h.

5.35.3.5 UART_PARITY_t enum [UART_PARITY_t](#)

Enumerator

UART_PARITY_DISABLE	
UART_PARITY_ODD	
UART_PARITY_EVEN	

Definition at line 30 of file UART_cfg.h.

5.35.3.6 UART_STOP_BITS_t enum [UART_STOP_BITS_t](#)

Enumerator

UART_STOP_1_BIT	
UART_STOP_2_BIT	

Definition at line 25 of file UART_cfg.h.

5.35.4 Variable Documentation

5.35.4.1 UART0_Configs [UART_CFG_t](#) UART0_Configs [extern]

Note

Baud rate options:

- 2400UL --> 2400 bits per second
- 4800UL --> 4800 bits per second
- 9600UL --> 9600 bits per second
- 14400UL --> 14400 bits per second
- 19200UL --> 19200 bits per second
- 28800UL --> 28800 bits per second
- 38400UL --> 38400 bits per second
- 57600UL --> 57600 bits per second
- 76800UL --> 76800 bits per second
- 115200UL --> 115200 bits per second
- 230400UL --> 230400 bits per second
- 250000UL --> 250000 bits per second
- 500000UL --> 500000 bits per second
- 1000000UL --> 1000000 bits per second

Definition at line 34 of file UART_cfg.c.

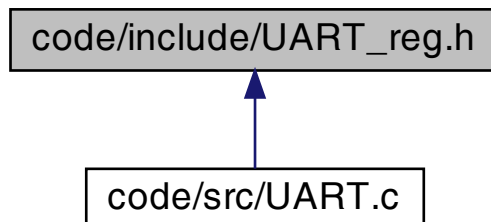
5.35.4.2 UART1_Configs [UART_CFG_t](#) UART1_Configs [extern]

Definition at line 43 of file UART_cfg.c.

5.36 code/include/UART_reg.h File Reference

Registers of UART of ATmega328p MCU.

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



Macros

- #define `UDR0` (* ((volatile `u8` *) 0x2C))
- #define `UCSR0A` (* ((volatile `u8` *) 0x2B))
- #define `UCSR0B` (* ((volatile `u8` *) 0x2A))
- #define `UCSR0C` (* ((volatile `u8` *) 0x95))
- #define `UBRR0L` (* ((volatile `u8` *) 0x29))
- #define `UBRR0H` (* ((volatile `u8` *) 0x90))
- #define `UDR1` (* ((volatile `u8` *) 0x9C))
- #define `UCSR1A` (* ((volatile `u8` *) 0x9B))
- #define `UCSR1B` (* ((volatile `u8` *) 0x9A))
- #define `UCSR1C` (* ((volatile `u8` *) 0x9D))
- #define `UBRR1L` (* ((volatile `u8` *) 0x99))
- #define `UBRR1H` (* ((volatile `u8` *) 0x98))

Enumerations

- enum {
`MPCM` , `U2X` , `UPE` , `DOR` ,
`FE` , `UDRE` , `TXC` , `RXC` }
- enum {
`TXB8` , `RXB8` , `UCSZ2` , `TXEN` ,
`RXEN` , `UDRIE` , `TXCIE` , `RXCIE` }
- enum {
`UCPOL` , `UCSZ0` , `UCSZ1` , `USBS` ,
`UPM0` , `UPM1` , `UMSEL` }

5.36.1 Detailed Description

Registers of UART of ATmega328p MCU.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2021-07-31

5.36.2 Macro Definition Documentation

5.36.2.1 UBRR0H `#define UBRR0H (* ((volatile u8 *) 0x90))`

Definition at line 19 of file UART_reg.h.

5.36.2.2 UBRR0L `#define UBRR0L (* ((volatile u8 *) 0x29))`

Definition at line 18 of file UART_reg.h.

5.36.2.3 UBRR1H `#define UBRR1H (* ((volatile u8 *) 0x98))`

Definition at line 29 of file UART_reg.h.

5.36.2.4 UBRR1L `#define UBRR1L (* ((volatile u8 *) 0x99))`

Definition at line 28 of file UART_reg.h.

5.36.2.5 UCSR0A `#define UCSR0A (* ((volatile u8 *) 0x2B))`

Definition at line 15 of file UART_reg.h.

5.36.2.6 UCSR0B `#define UCSR0B (* ((volatile u8 *) 0x2A))`

Definition at line 16 of file UART_reg.h.

5.36.2.7 UCSR0C `#define UCSR0C (* ((volatile u8 *) 0x95))`

Definition at line 17 of file UART_reg.h.

5.36.2.8 UCSR1A `#define UCSR1A (* ((volatile u8 *) 0x9B))`

Definition at line 25 of file UART_reg.h.

5.36.2.9 UCSR1B `#define UCSR1B (* ((volatile u8 *) 0x9A))`

Definition at line 26 of file UART_reg.h.

5.36.2.10 UCSR1C `#define UCSR1C (* ((volatile u8 *) 0x9D))`

Definition at line 27 of file UART_reg.h.

5.36.2.11 UDR0 `#define UDR0 (* ((volatile u8 *) 0x2C))`

UART0 Registers

Definition at line 14 of file UART_reg.h.

5.36.2.12 UDR1 `#define UDR1 (* ((volatile u8 *) 0x9C))`

UART1 Registers

Definition at line 24 of file UART_reg.h.

5.36.3 Enumeration Type Documentation

5.36.3.1 anonymous enum `anonymous enum`

Registers' Bits

Enumerator

MPCM	
U2X	
UPE	
DOR	
FE	
UDRE	
TXC	
RXC	

Definition at line 34 of file UART_reg.h.

5.36.3.2 anonymous enum `anonymous_enum`

Enumerator

TXB8	
RXB8	
UCSZ2	
TXEN	
RXEN	
UDRIE	
TXCIE	
RXCIE	

Definition at line 45 of file UART_reg.h.

5.36.3.3 anonymous enum `anonymous_enum`

Enumerator

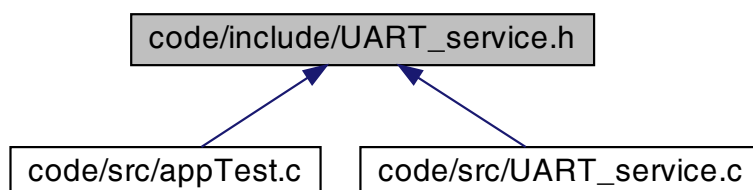
UCPOL	
UCSZ0	
UCSZ1	
USBS	
UPM0	
UPM1	
UMSEL	

Definition at line 56 of file UART_reg.h.

5.37 code/include/UART_service.h File Reference

UART services interfaces file.

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



Functions

- void [UART0_SendString](#) (const [u8](#) *const string)
Send a string using UART module 0.
- void [UART0_SendString_Checksum](#) (const [u8](#) *const string)
- void [UART1_SendString](#) (const [u8](#) *const string)
Send a string using UART module 1.
- void [UART0_Send9BitString](#) (const [u16](#) *const string)
Send a sequence of elements, each element has 9 bits, using UART module 0.
- void [UART1_Send9BitString](#) (const [u16](#) *const string)
Send a sequence of elements, each element has 9 bits, using UART module 1.
- void [UART0_SendInteger](#) ([s32](#) integer)
Write Integer to UART0 as a string.
- void [UART1_SendInteger](#) ([s32](#) integer)
Write Integer to UART1 as a string.
- void [UART0_SendFloat](#) (float number, const [u8](#) precision)
Write Float to UART0 as a string.
- void [UART1_SendFloat](#) (float number, const [u8](#) precision)
Write Float to UART1 as a string.
- [ERROR_STATUS_t](#) [UART0_ReceiveString](#) ([u8](#) *const string)
Receive a string of characters (unsigned 8 bits data) using UART module 0.
- void [UART0_SendString_Asynchronous](#) (const [u8](#) *const string)
- [ERROR_STATUS_t](#) [UART1_ReceiveString](#) ([u8](#) *const string)
Receive a string of characters (unsigned 8 bits data) using UART module 1.
- void [UART1_SendString_Asynchronous](#) (const [u8](#) *const string)
- [ERROR_STATUS_t](#) [UART0_ReceiveString_Checksum](#) ([u8](#) *const string)
- [ERROR_STATUS_t](#) [UART0_Receive9BitString](#) ([u16](#) *const string)
Receive a string of elements, each element has 9 bits, using UART module 0.
- [ERROR_STATUS_t](#) [UART1_Receive9BitString](#) ([u16](#) *const string)
Receive a string of elements, each element has 9 bits, using UART module 1.

5.37.1 Detailed Description

UART services interfaces file.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-03

Copyright

Copyright (c) 2022

5.37.2 Function Documentation

5.37.2.1 UART0_Receive9BitString() `ERROR_STATUS_t UART0_Receive9BitString (ul6 *const string)`

Receive a string of elements, each element has 9 bits, using UART module 0.

Parameters

in	<i>string</i>	Pointer to the first element of the sequence (pointer to 16 bits)
----	---------------	---

Note

Size of each element should be 2 Bytes to be able to receive 9 bits data

For Example: `UART0_Receive9BitDataSequence(string);` will receive elements of 9 bits

each from UART module 0.

Warning

The sequence will be ended with a NULL pointer(0): The last element of the sequence will be a NULL pointer. So, consider the length of the sequence as `strlen(string) + 1`.

Definition at line 394 of file `UART_service.c`.

5.37.2.2 UART0_ReceiveString() `ERROR_STATUS_t UART0_ReceiveString (`
`u8 *const string)`

Receive a string of characters (unsigned 8 bits data) using UART module 0.

For Example: `UART0_ReceiveString(string);` will receive a string of characters

from UART module 0 and store it in variable `<string>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Warning

The string will be null terminated. So, the last character will be `'\0'`. So, the length of the string should be `strlen(string) + 1`.

Definition at line 386 of file `UART_service.c`.

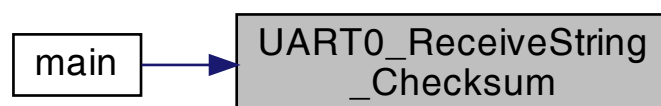
Here is the caller graph for this function:



5.37.2.3 UART0_ReceiveString_Checksum() `ERROR_STATUS_t UART0_ReceiveString_Checksum (`
`u8 *const string)`

Definition at line 403 of file `UART_service.c`.

Here is the caller graph for this function:



5.37.2.4 UART0_Send9BitString() `void UART0_Send9BitString (`
`const u16 *const string)`

Send a sequence of elements, each element has 9 bits, using UART module 0.

Parameters

in	<i>string</i>	Pointer to the first element of the sequence
----	---------------	--

Note

Size of each element should be 2 Bytes to be able to send 9 bits data

For Example: `UART0_Send9BitDataSequence(string);` will send elements of 9 bits

each to UART module 0.

Note

The sequence should be ended with a NULL pointer(0): The last element of the sequence should be a NULL pointer.

But if you are sending an array, you must explicitly null terminate it by adding `'\0'` at the end of the array --> `string[lenght-1] = '\0'`

Definition at line 331 of file `UART_service.c`.

5.37.2.5 UART0_SendFloat() `void UART0_SendFloat (`
`float number,`
`const u8 precision)`

Write Float to UART0 as a string.

Parameters

in	<i>number</i>	The number to be written
----	---------------	--------------------------

For Example:

- `UART0_WriteFloat(123.456, 3)` will write the string "123.456" to UART0
- `UART0_WriteFloat(123.456, 2)` will write the string "123.45" to UART0
- `UART0_WriteFloat(123.456, 5)` will write the string "123.45600" to UART0
- `UART0_WriteFloat(-123.456, 3)` will write the string "-123.456" to UART0
- `UART0_WriteFloat(-123.456, 2)` will write the string "-123.45" to UART0
- `UART0_WriteFloat(-123.456, 5)` will write the string "-123.45600" to UART0

Definition at line 377 of file `UART_service.c`.

5.37.2.6 UART0_SendInteger() `void UART0_SendInteger (`
`s32 integer)`

Write Integer to UART0 as a string.

Parameters

in	<i>integer</i>	The integer to be written
----	----------------	---------------------------

For Example:

- UART0_WriteInt(123) will write the string "123" to UART0
- UART0_WriteInt(-123) will write the string "-123" to UART0

Definition at line 368 of file UART_service.c.

Here is the caller graph for this function:



5.37.2.7 UART0_SendString() `void UART0_SendString (`
`const u8 *const string)`

Send a string using UART module 0.

For Example: `UART0_SendString("Hello World");` will send string "Hello World" to UART module 0

Definition at line 323 of file UART_service.c.

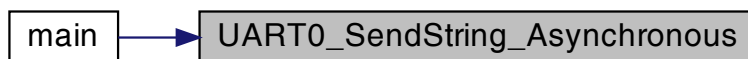
Here is the caller graph for this function:



5.37.2.8 UART0_SendString_Asynchronous() `void UART0_SendString_Asynchronous (`
`const u8 *const string)`

Definition at line 340 of file UART_service.c.

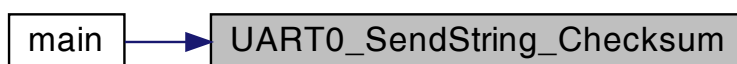
Here is the caller graph for this function:



5.37.2.9 UART0_SendString_Checksum() `void UART0_SendString_Checksum (`
`const u8 *const string)`

Definition at line 359 of file UART_service.c.

Here is the caller graph for this function:



5.37.2.10 UART1_Receive9BitString() `ERROR_STATUS_t UART1_Receive9BitString (`
`u16 *const string)`

Receive a string of elements, each element has 9 bits, using UART module 1.

Parameters

in	string	Pointer to the first element of the sequence (pointer to 16 bits)
----	--------	---

Note

Size of each element should be 2 Bytes to be able to receive 9 bits data

For Example: `UART1_Receive9BitDataSequence(string);` will receive elements of 9 bits each from UART module 1.

Warning

The sequence will be ended with a NULL pointer(0): The last element of the sequence will be a NULL pointer. So, consider the length of the sequence as `strlen(string) + 1`.

Definition at line 398 of file `UART_service.c`.

5.37.2.11 UART1_ReceiveString() `ERROR_STATUS_t UART1_ReceiveString (u8 *const string)`

Receive a string of characters (unsigned 8 bits data) using UART module 1.

For Example: `UART1_ReceiveString(string);` will receive a string of characters from UART module 1 and store it in variable `<string>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Warning

The string will be null terminated. So, the last character will be `'\0'`. So, the length of the string should be `strlen(string) + 1`.

Definition at line 390 of file `UART_service.c`.

5.37.2.12 UART1_Send9BitString() `void UART1_Send9BitString (const u16 *const string)`

Send a sequence of elements, each element has 9 bits, using UART module 1.

Parameters

in	<i>string</i>	Pointer to the first element of the sequence
----	---------------	--

Note

Size of each element should be 2 Bytes to be able to send 9 bits data

For Example: `UART1_Send9BitDataSequence(string);` will send elements of 9 bits

each to UART module 1.

Note

The sequence should be ended with a NULL pointer(0): The last element of the sequence should be a NULL pointer.

But if you are sending an array, you must explicitly null terminate it by adding '\0' at the end of the array --> `string[lenght-1] = '\0'`

Definition at line 335 of file `UART_service.c`.

5.37.2.13 UART1_SendFloat() `void UART1_SendFloat (`
 `float number,`
 `const u8 precision)`

Write Float to UART1 as a string.

Parameters

in	<i>number</i>	The number to be written
----	---------------	--------------------------

For Example:

- `UART1_WriteFloat(123.456, 3)` will write the string "123.456" to UART1
- `UART1_WriteFloat(123.456, 2)` will write the string "123.45" to UART1
- `UART1_WriteFloat(123.456, 5)` will write the string "123.45600" to UART1
- `UART1_WriteFloat(-123.456, 3)` will write the string "-123.456" to UART1
- `UART1_WriteFloat(-123.456, 2)` will write the string "-123.45" to UART1
- `UART1_WriteFloat(-123.456, 5)` will write the string "-123.45600" to UART1

Definition at line 381 of file `UART_service.c`.

Here is the caller graph for this function:



5.37.2.14 UART1_SendInteger() `void UART1_SendInteger (`
 `s32 integer)`

Write Integer to UART1 as a string.

Parameters

in	<i>integer</i>	The integer to be written
----	----------------	---------------------------

For Example:

- UART1_WriteInt(123) will write the string "123" to UART1
- UART1_WriteInt(-123) will write the string "-123" to UART1

Definition at line 372 of file UART_service.c.

Here is the caller graph for this function:



5.37.2.15 UART1_SendString() `void UART1_SendString (`
`const u8 *const string)`

Send a string using UART module 1.

For Example: `UART1_SendString("Hello World");` will send string "Hello World"

to UART module 1.

Note

String must be null terminated: "Hello World\0". So, the last character must be '\0'.

Sending "Hello World" will send "Hello World\0" because it is null terminated implicitly. But if you are sending an array of characters, you must explicitly null terminate it by adding '\0' at the end of the array. --> `string[lenght-1] = '\0'`

Definition at line 327 of file UART_service.c.

Here is the caller graph for this function:



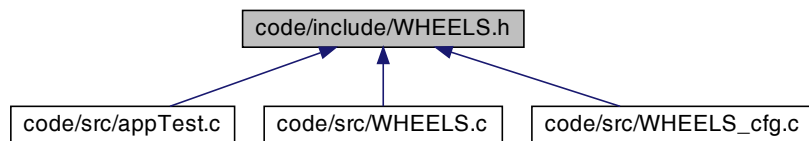
5.37.2.16 UART1_SendString_Asynchronous() `void UART1_SendString_Asynchronous (const u8 *const string)`

Definition at line 344 of file UART_service.c.

5.38 code/include/WHEELS.h File Reference

Interfaces header file for [WHEELS.c](#).

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



Enumerations

- enum [WHEELS_POSITION_t](#) { [WHEELS_ON_FRONT](#) , [WHEELS_ON_BACK](#) }
- enum [WHEELS_TURN_t](#) { [SMOOTH_TURN](#) , [SHARP_TURN](#) }

Functions

- void [WHEELS_Init](#) (void)
Initialize the wheels module.
- void [WHEELS_GoForward](#) (void)
Turn the wheels to go forward.
- void [WHEELS_GoBackward](#) (void)
Turn the wheels to go backward.
- void [WHEELS_TurnLeft](#) ([WHEELS_TURN_t](#) smoothOrSharp)
Turn the wheels to turn left.
- void [WHEELS_TurnRight](#) ([WHEELS_TURN_t](#) smoothOrSharp)
Turn the wheels to turn right.
- void [WHEELS_Brake](#) (void)
Brake the wheels.
- void [WHEELS_Coast](#) (void)
Disables wheels and allows for free rolling.
- void [WHEELS_SetSpeed](#) (u8 Speed)
Set the speed of the wheels.
- void [WHEELS_SetWheelsPosition](#) ([WHEELS_POSITION_t](#) wheelsPositions)
Set the position of the wheels: on the front or on the back.
- f32 [WHEELS_GetCurrentConsumption](#) ()
Get the current consumption of the wheels.
- [WHEELS_POSITION_t](#) [WHEELS_GetWheelsPosition](#) (void)
Get the current position of the wheels.

5.38.1 Detailed Description

Interfaces header file for [WHEELS.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.38.2 Enumeration Type Documentation

5.38.2.1 `WHEELS_POSITION_t` enum [WHEELS_POSITION_t](#)

Enumerator

WHEELS_ON_FRONT	
WHEELS_ON_BACK	

Definition at line 12 of file WHEELS.h.

5.38.2.2 `WHEELS_TURN_t` enum [WHEELS_TURN_t](#)

Enumerator

SMOOTH_TURN	
SHARP_TURN	

Definition at line 17 of file WHEELS.h.

5.38.3 Function Documentation

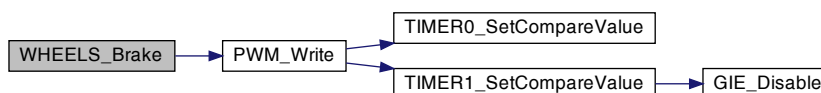
5.38.3.1 WHEELS_Brake()

```
void WHEELS_Brake (
    void )
```

Brake the wheels.

Definition at line 153 of file WHEELS.c.

Here is the call graph for this function:



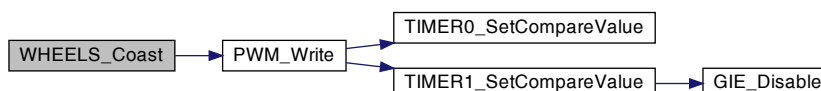
5.38.3.2 WHEELS_Coast()

```
void WHEELS_Coast (
    void )
```

Disables wheels and allows for free rolling.

Definition at line 164 of file WHEELS.c.

Here is the call graph for this function:



5.38.3.3 WHEELS_GetCurrentConsumption()

```
f32 WHEELS_GetCurrentConsumption ( )
```

Get the current consumption of the wheels.

Returns

f32 The current consumption of the wheels in A.

5.38.3.4 WHEELS_GetWheelsPosition() `WHEELS_POSITION_t WHEELS_GetWheelsPosition (void)`

Get the current position of the wheels.

Returns

f32 The current consumption of the wheels in A.

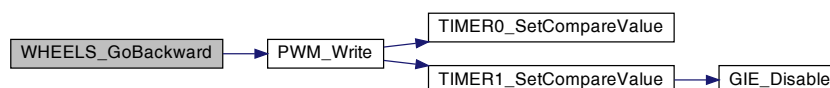
Definition at line 202 of file WHEELS.c.

5.38.3.5 WHEELS_GoBackward() `void WHEELS_GoBackward (void)`

Turn the wheels to go backward.

Definition at line 91 of file WHEELS.c.

Here is the call graph for this function:

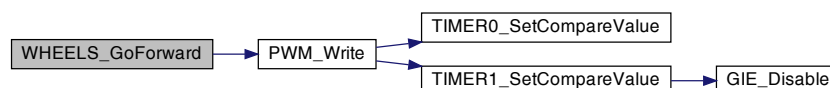


5.38.3.6 WHEELS_GoForward() `void WHEELS_GoForward (void)`

Turn the wheels to go forward.

Definition at line 79 of file WHEELS.c.

Here is the call graph for this function:

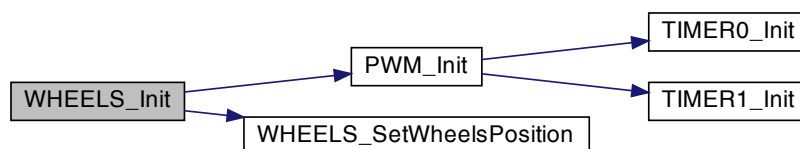


5.38.3.7 WHEELS_Init() `void WHEELS_Init (`
`void)`

Initialize the wheels module.

Definition at line 68 of file WHEELS.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.8 WHEELS_SetSpeed() `void WHEELS_SetSpeed (`
`u8 Speed)`

Set the speed of the wheels.

Parameters

in	<i>Speed</i>	The speed of the wheels.
----	--------------	--------------------------

Definition at line 176 of file WHEELS.c.

5.38.3.9 WHEELS_SetWheelsPosition() `void WHEELS_SetWheelsPosition (`
`WHEELS_POSITION_t wheelsPositions)`

Set the position of the wheels: on the front or on the back.

Parameters

in	<i>wheelsPositions</i>	The poition of the wheels: WHEELS_ON_FRONT or WHEELS_ON_BACK.
----	------------------------	---

Definition at line 181 of file WHEELS.c.

Here is the caller graph for this function:



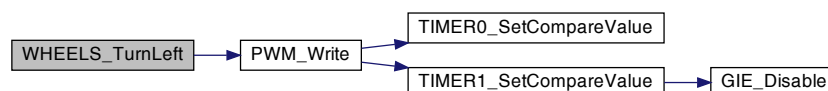
5.38.3.10 WHEELS_TurnLeft()

```
void WHEELS_TurnLeft (
    WHEELS_TURN_t smoothOrSharp )
```

Turn the wheels to turn left.

Definition at line 103 of file WHEELS.c.

Here is the call graph for this function:



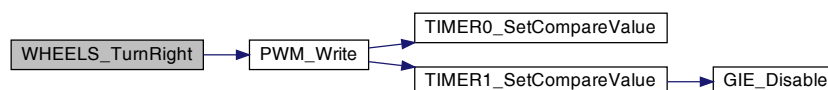
5.38.3.11 WHEELS_TurnRight()

```
void WHEELS_TurnRight (
    WHEELS_TURN_t smoothOrSharp )
```

Turn the wheels to turn right.

Definition at line 128 of file WHEELS.c.

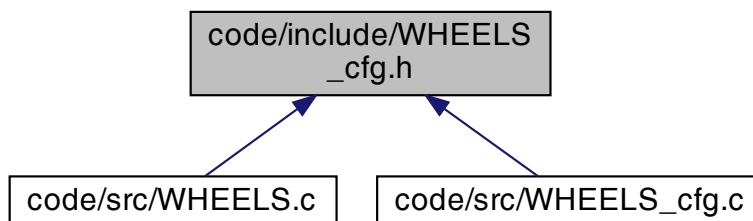
Here is the call graph for this function:



5.39 code/include/WHEELS_cfg.h File Reference

Configuration header file for [WHEELS.c](#).

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



Data Structures

- struct [WHEELS_CONFIG_t](#)

Enumerations

- enum [ROTATE_DIR_t](#) {
[ROTATE_STOP](#) , [ROTATE_FWD](#) , [ROTATE_BACK](#) , [ROTATE_RIGHT_SMOOTH](#) ,
[ROTATE_RIGHT_SHARP](#) , [ROTATE_LEFT_SMOOTH](#) , [ROTATE_LEFT_SHARP](#) }

Variables

- [WHEELS_CONFIG_t](#) [WHEELS_Config](#)

5.39.1 Detailed Description

Configuration header file for [WHEELS.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.39.2 Enumeration Type Documentation

5.39.2.1 ROTATE_DIR_t enum [ROTATE_DIR_t](#)

Enumerator

ROTATE_STOP	
ROTATE_FWD	
ROTATE_BACK	
ROTATE_RIGHT_SMOOTH	
ROTATE_RIGHT_SHARP	
ROTATE_LEFT_SMOOTH	
ROTATE_LEFT_SHARP	

Definition at line 33 of file WHEELS_cfg.h.

5.39.3 Variable Documentation

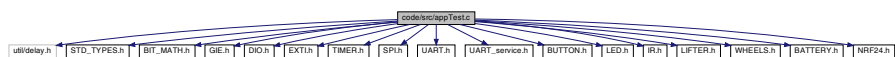
5.39.3.1 WHEELS_Config `WHEELS_CONFIG_t` WHEELS_Config [extern]

Definition at line 16 of file WHEELS_cfg.c.

5.40 code/src/appTest.c File Reference

```
#include <util/delay.h>
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "GIE.h"
#include "DIO.h"
#include "EXTI.h"
#include "TIMER.h"
#include "SPI.h"
#include "UART.h"
#include "UART_service.h"
#include "BUTTON.h"
#include "LED.h"
#include "IR.h"
#include "LIFTER.h"
#include "WHEELS.h"
#include "BATTERY.h"
#include "NRF24.h"
```

Include dependency graph for appTest.c:



Functions

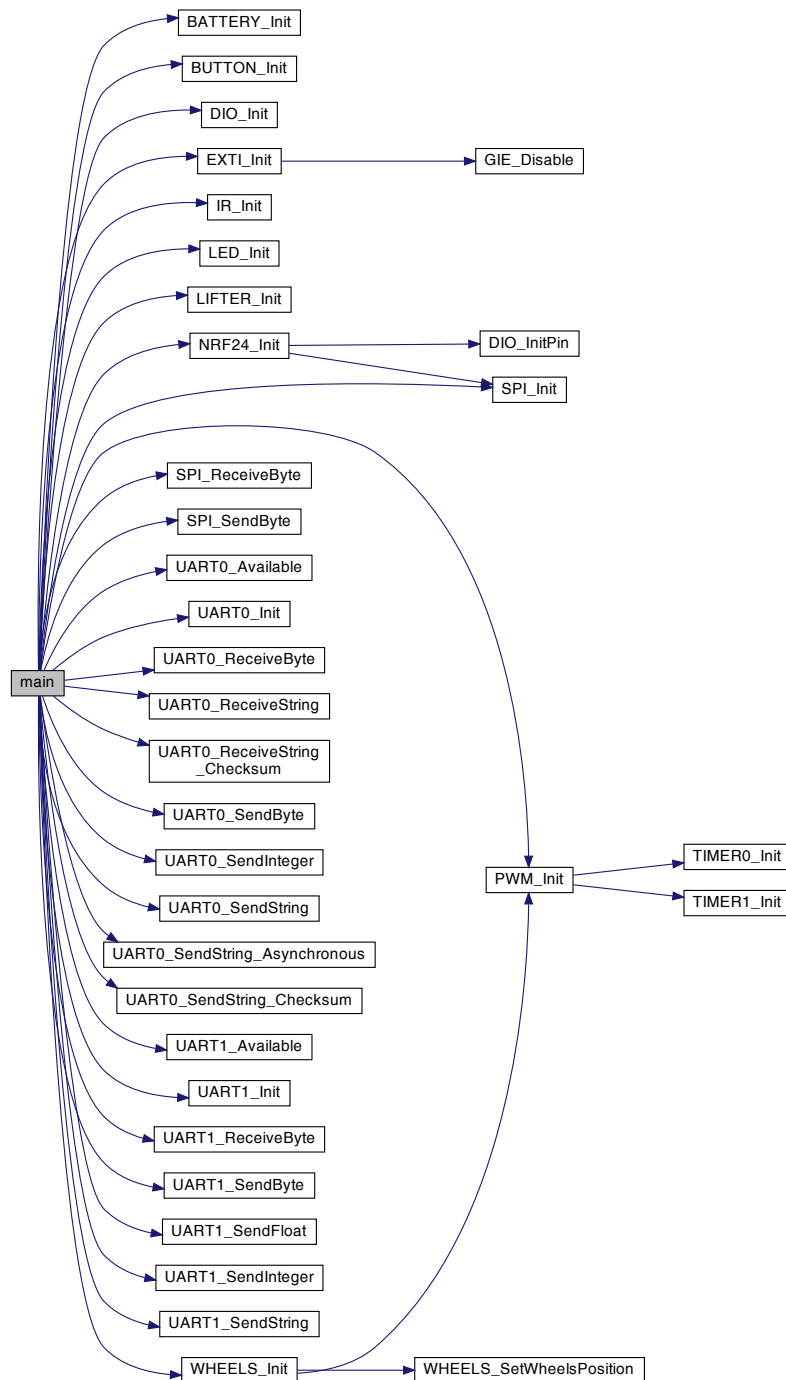
- int `main` (void)

5.40.1 Function Documentation

5.40.1.1 main() `int main (`
`void)`

Definition at line 27 of file appTest.c.

Here is the call graph for this function:



5.41 code/src/BATTERY.c File Reference

Battery Management Module.

5.41.1 Detailed Description

Battery Management Module.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-02-08

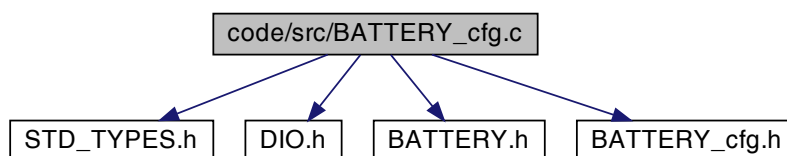
Copyright

Copyright (c) 2022

5.42 code/src/BATTERY_cfg.c File Reference

Configuration source file for [BATTERY.c](#).

```
#include "STD_TYPES.h"
#include "DIO.h"
#include "BATTERY.h"
#include "BATTERY_cfg.h"
Include dependency graph for BATTERY_cfg.c:
```



Variables

- [BatteryConfigs_t BatteryConfigs](#)

Configurations for the Battery Management Module.

5.42.1 Detailed Description

Configuration source file for [BATTERY.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.42.2 Variable Documentation

5.42.2.1 BatteryConfigs [BatteryConfigs_t](#) BatteryConfigs

Initial value:

```
= {  
    .pin      = PIN_0,  
    .port     = PORT_C,  
    .minVoltage = 11.2f,  
    .maxVoltage = 12.8f  
}
```

Configurations for the Battery Management Module.

This is the configuration for the Battery Management Module. The configuration is done by the user. The user can change the configuration according to his needs.

Note

The configuration is done in the [BATTERY_cfg.h](#) and [BATTERY_cfg.c](#) files.

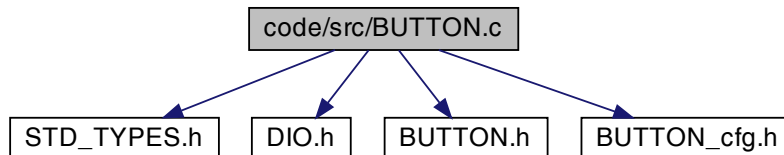
Definition at line 21 of file [BATTERY_cfg.c](#).

5.43 code/src/BUTTON.c File Reference

Button Management Module.

```
#include "STD_TYPES.h"
#include "DIO.h"
#include "BUTTON.h"
#include "BUTTON_cfg.h"
```

Include dependency graph for BUTTON.c:



Functions

- void [BUTTON_Init](#) (void)
Initializes Buttons connected to DIO.
- [STATE_t BUTTON_GetStatus](#) ([BUTTON_t](#) button)
Check whether a specific button is pressed or not.

5.43.1 Detailed Description

Button Management Module.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.43.2 Function Documentation

5.43.2.1 [BUTTON_GetStatus](#)([STATE_t](#) [BUTTON_GetStatus](#) ([BUTTON_t](#) button)

Check whether a specific button is pressed or not.

Parameters

in	<i>button</i>	Button number: options from BUTTON_t enum in BUTTON.h file
----	---------------	--

Returns

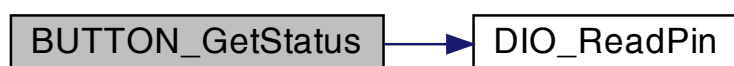
State of the button: [HIGH](#) if pressed and [LOW](#) if not, member of [STATE_t](#) enum

For example:

```
BUTTON_GetStatus(BUTTON_0); //returns \ref HIGH if button 0 is pressed and \ref LOW otherwise
```

Definition at line 18 of file BUTTON.c.

Here is the call graph for this function:



5.43.2.2 `BUTTON_Init()` `void BUTTON_Init (`
`void)`

Initializes Buttons connected to DIO.

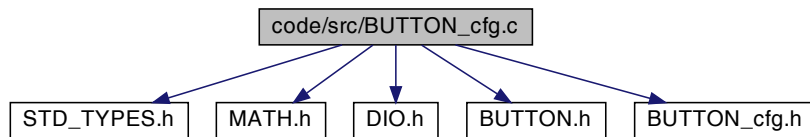
Definition at line 14 of file BUTTON.c.

Here is the caller graph for this function:



5.44 code/src/BUTTON_cfg.c File Reference

```
#include "STD_TYPES.h"
#include "MATH.h"
#include "DIO.h"
#include "BUTTON.h"
#include "BUTTON_cfg.h"
Include dependency graph for BUTTON_cfg.c:
```



Variables

- `BUTTON_CONFIGS_t` `buttonsConfigs []`
- `const u8` `countButtonsConfigured = SIZE_OF_ARRAY(buttonsConfigs)`

5.44.1 Variable Documentation

5.44.1.1 `buttonsConfigs` `BUTTON_CONFIGS_t` `buttonsConfigs []`

Initial value:

```
= {
    {BUTTON_0, PIN_6, PORT_D, ACTIVE_LOW, DEBOUNCE_OFF},
    {BUTTON_1, PIN_7, PORT_D, ACTIVE_LOW, DEBOUNCE_OFF},
}
```

Note

ACTIVE_LOW means that the pin is:

- LOW when the sensor is pressed
- HIGH when the sensor is not pressed
- ACTIVE_HIGH means that the pin is:
 - HIGH when the sensor is pressed
 - LOW when the sensor is not pressed

Definition at line 23 of file `BUTTON_cfg.c`.

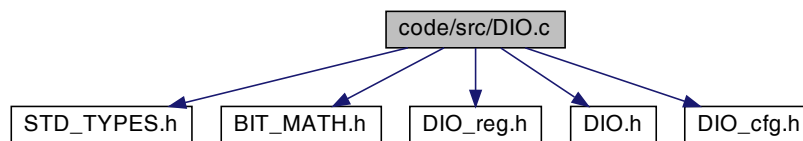
5.44.1.2 `countButtonsConfigured` `const u8` `countButtonsConfigured = SIZE_OF_ARRAY(buttonsConfigs)`

Definition at line 28 of file `BUTTON_cfg.c`.

5.45 code/src/DIO.c File Reference

Digital Input Output (DIO) driver for Atmega128 microcontroller.

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "DIO_reg.h"
#include "DIO.h"
#include "DIO_cfg.h"
Include dependency graph for DIO.c:
```



Macros

- `#define INIT_PIN(pin, port, direction, pullup, direction_reg)`
Used inside `DIO_Init()` to initialize a pin.
- `#define WRITE_PIN(pin, port, state)`
used inside the `writePin()` function to write a pin Calling direction: `DIO_WritePin()` -> `writePin()` -> `WRITE_PIN()`

Functions

- `void DIO_Init (void)`
Initialize DIO configurations based on user configurations in `DIO_cfg.h` and `DIO_cfg.c`.
- `void DIO_InitPin (const PIN_t pin, const PORT_t port, const DIR_t direction, const PULLUP_t pullup)`
Initialize a pin as input or output.
- `void DIO_WritePin (const PIN_t pin, const PORT_t port, const STATE_t pinState)`
write a value on the output pins, options are defined in `STD_TYPES.h` in the enum `STATE_t`
- `void DIO_TogglePin (const PIN_t pin, const PORT_t port)`
- `void DIO_WritePort (const PORT_t port, const u8 value)`
write a value on a specific port (value of 8-bits ranges from 0 to 255)
- `STATE_t DIO_ReadPin (const PIN_t pin, const PORT_t port)`
Read the state of a pin.
- `u8 DIO_ReadPort (const PORT_t port)`
Read the state of the port (8 bits --> 0-255)

5.45.1 Detailed Description

Digital Input Output (DIO) driver for Atmega128 microcontroller.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.45.2 Macro Definition Documentation

5.45.2.1 INIT_PIN #define INIT_PIN(
 pin,
 port,
 direction,
 pullup,
 direction_reg)

Value:

```
if(INPUT == direction) {
    CLR_BIT(direction_reg, pin);

    if(PULLUP_TRUE == pullup) {
        SET_BIT(port, pin);
    }
    else {
        CLR_BIT(port, pin);
    }
}
else if(OUTPUT == direction) {
    SET_BIT(direction_reg, pin);
    CLR_BIT(port, pin);
}
else { /* Invalid Direction */
    ; /* Null operation */
}
```

Used inside [DIO_Init\(\)](#) to initialize a pin.

Parameters

<i>pin</i>	pin to initialize, options are defined in DIO.h file in the enum PIN_t
<i>port</i>	port to initialize, options are defined in DIO.h file in the enum PORT_t
<i>direction</i>	pin direction, options are defined in DIO.h file in the enum DIRECTION_t

Definition at line 25 of file DIO.c.

5.45.2.2 WRITE_PIN `#define WRITE_PIN(
 pin,
 port,
 state)`

Value:

```
if (LOW == state) {
    CLR_BIT(port, pin);
} else if (HIGH == state) {
    SET_BIT(port, pin);
} else {
    /* Invalid State */
    ; /* Null operation */
}
```

used inside the writePin() function to write a pin Calling direction: [DIO_WritePin\(\)](#) -> writePin() -> [WRITE_PIN\(\)](#)

Parameters

<i>pin</i>	the pin number, options are defined in the DIO.h file in the enum PIN_t
<i>port</i>	the port number, options are defined in the DIO.h file in the enum PORT_t
<i>pinState</i>	the state of the pin, options are defined in the DIO.h file in the enum STATE_t

Definition at line 51 of file DIO.c.

5.45.3 Function Documentation

5.45.3.1 DIO_Init() `void DIO_Init (
 void)`

Initialize DIO configurations based on user configurations in [DIO_cfg.h](#) and [DIO_cfg.c](#).

Initialize DIO pins to a specific direction (input or output), pullup or not according to the configuration in the [DIO_cfg.h](#) file.

Definition at line 67 of file DIO.c.

Here is the caller graph for this function:



5.45.3.2 DIO_InitPin() `void DIO_InitPin (`
 `const PIN_t pin,`
 `const PORT_t port,`
 `const DIR_t direction,`
 `const PULLUP_t pullup)`

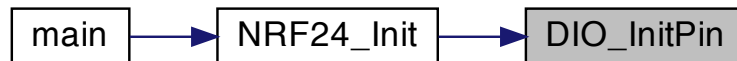
Initialize a pin as input or output.

If:

- DIO pin is not configured in the [DIO_cfg.h](#) file
- Or the pin is configurations need to be modified Then, this function will be called to modify the pin configuration.

Definition at line 110 of file DIO.c.

Here is the caller graph for this function:



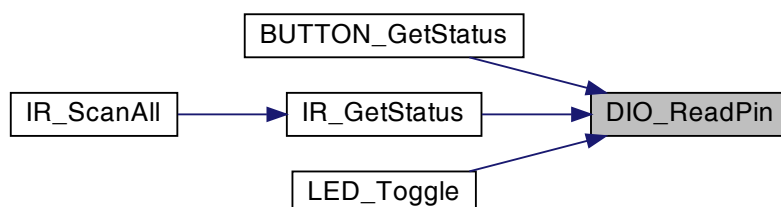
5.45.3.3 DIO_ReadPin() `STATE_t DIO_ReadPin (`
 `const PIN_t pin,`
 `const PORT_t port)`

Read the state of a pin.

If the pin is configured in the [DIO_cfg.h](#) file, then this function can be called to read the pin (1 or 0).

Definition at line 273 of file DIO.c.

Here is the caller graph for this function:



5.45.3.4 DIO_ReadPort() `u8 DIO_ReadPort (`
`const PORT_t port)`

Read the state of the port (8 bits --> 0-255)

Read the value of the port (all pins).

Definition at line 321 of file DIO.c.

5.45.3.5 DIO_TogglePin() `void DIO_TogglePin (`
`const PIN_t pin,`
`const PORT_t port)`

This function can be called to toggle the state of the pin. If the pins state is HIGH, then it will be set to LOW, and vice versa.

Definition at line 192 of file DIO.c.

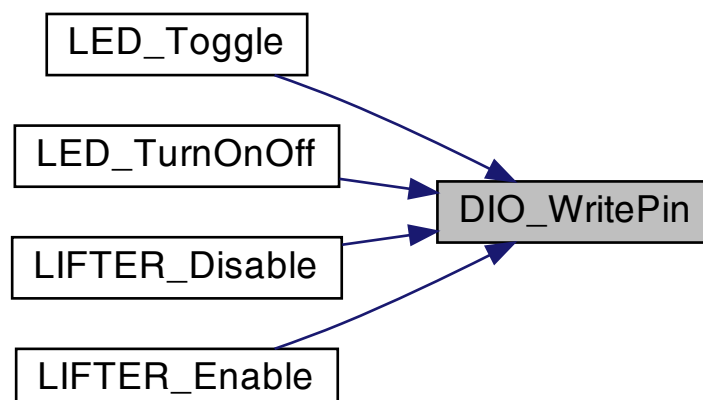
5.45.3.6 DIO_WritePin() `void DIO_WritePin (`
`const PIN_t pin,`
`const PORT_t port,`
`const STATE_t pinState)`

write a value on the output pins, options are defined in [STD_TYPES.h](#) in the enum STATE_t

If the pin is configured in the [DIO_cfg.h](#) file, then this function can be called to write to the pin (1 or 0).

Definition at line 149 of file DIO.c.

Here is the caller graph for this function:



5.45.3.7 DIO_WritePort() `void DIO_WritePort (`
 `const PORT_t port,`
 `const u8 value)`

write a value on a specific port (value of 8-bits ranges from 0 to 255)

Write a value to the port (all pins).

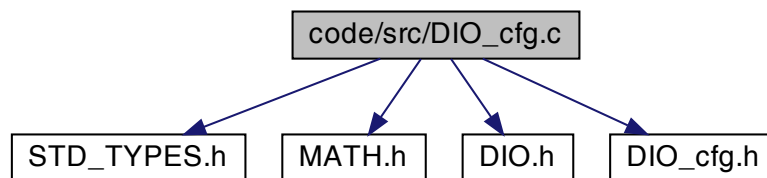
Definition at line 234 of file DIO.c.

5.46 code/src/DIO_cfg.c File Reference

Configuration source file for [DIO.c](#).

```
#include "STD_TYPES.h"  
#include "MATH.h"  
#include "DIO.h"  
#include "DIO_cfg.h"
```

Include dependency graph for DIO_cfg.c:



Variables

- [PinConfig_t](#) pinConfigs []
- `const u8 countPinsConfigured = SIZE_OF_ARRAY(pinConfigs)`

5.46.1 Detailed Description

Configuration source file for [DIO.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.46.2 Variable Documentation

5.46.2.1 countPinsConfigured `const u8 countPinsConfigured = SIZE_OF_ARRAY(pinConfigs)`

Definition at line 53 of file DIO_cfg.c.

5.46.2.2 pinConfigs `PinConfig_t pinConfigs[]`

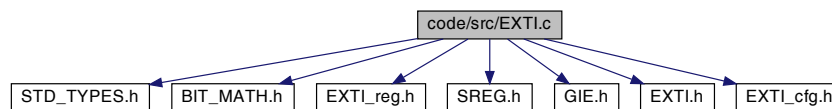
Definition at line 14 of file DIO_cfg.c.

5.47 code/src/EXTI.c File Reference

External Interrupts (EXTI) driver for Atmega128 microcontroller.

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "EXTI_reg.h"
#include "SREG.h"
#include "GIE.h"
#include "EXTI.h"
#include "EXTI_cfg.h"
```

Include dependency graph for EXTI.c:



Functions

- void [__vector_1](#) (void)
- void [__vector_2](#) (void)
- void [__vector_3](#) (void)
- void [__vector_4](#) (void)
- void [__vector_5](#) (void)
- void [__vector_6](#) (void)
- void [__vector_7](#) (void)
- void [__vector_8](#) (void)
- void [EXTI_Init](#) (const [EXTI_t](#) extiNumber, const [EXTI_SENSITIVITY_t](#) sensitivity, void(*const callback↔Ptr)(void))
 - Initialize an EXTI pin with a given sensitivity and callback function.*
- void [EXTI_EnableExternalInterrupt](#) (const [EXTI_t](#) extiNumber)
 - Enable an EXTI pin (EXTI0 - EXTI7)*
- void [EXTI_DisableExternalInterrupt](#) (const [EXTI_t](#) extiNumber)
 - Disable an EXTI pin (EXTI0 - EXTI7)*

5.47.1 Detailed Description

External Interrupts (EXTI) driver for Atmega128 microcontroller.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-02-05

There are two types of interrupts:

1. Interrupts that are generated by the peripheral (HW interrupt).
2. Interrupts that are generated by the application (SW interrupt). The application can generate interrupts by calling the function `EXTI_GenerateSWInterrupt()`. The peripheral can generate interrupts by calling the function `EXTI_GenerateHWInterrupt()`. The application can clear the interrupt by calling the function `EXTI_ClearSWInterrupt()`.

There are 4 modes of interrupt generation:

1. Interrupt on falling edge.
2. Interrupt on rising edge.
3. Interrupt on both edges (On Change).
4. Interrupt on active level. When edge triggered, the interrupt is triggered only once when a falling edge from high logic level to low logic level happens. Only once even if the signal stays low, or goes high again. But when level triggered, the interrupt is triggered repeatedly while the logic level is low, continuously, until the logic level goes high again. So you really don't want to re-enable interrupts in a ISR which is triggered on low level. And if the level is low, not much mainline code gets chance to run, because the ISR is run repeatedly. So yes, basically low level and falling edge types of interrupts are both triggered when signal goes from high to low. Difference is that other gets triggered only once at the falling edge, the other triggers constantly while the level is low.

Note

INT0 to INT3 do not have (On Change mode)

INT4 to INT7 need IO clock if configured as edge triggered.

INT0 to INT3 use Asynchronous clock. So, they can wake up the MCU from sleep mode at any mode.

Todo Debug configuration of INT4 to INT7 as edge triggered not working.

Copyright

Copyright (c) 2022

5.47.2 Function Documentation

5.47.2.1 `__vector_1()` `void __vector_1 (`
`void)`

ISR of INT0

Definition at line 70 of file EXTI.c.

5.47.2.2 `__vector_2()` `void __vector_2 (`
`void)`

ISR of INT1

Definition at line 91 of file EXTI.c.

5.47.2.3 `__vector_3()` `void __vector_3 (`
`void)`

ISR of INT2

Definition at line 113 of file EXTI.c.

5.47.2.4 `__vector_4()` `void __vector_4 (`
`void)`

ISR of INT3

Definition at line 134 of file EXTI.c.

5.47.2.5 `__vector_5()` `void __vector_5 (`
`void)`

ISR of INT4

Definition at line 155 of file EXTI.c.

5.47.2.6 `__vector_6()` `void __vector_6 (`
`void)`

ISR of INT5

Definition at line 176 of file EXTI.c.

5.47.2.7 `__vector_7()` `void __vector_7 (`
`void)`

ISR of INT6

Definition at line 197 of file EXTI.c.

5.47.2.8 `__vector_8()` `void __vector_8 (`
`void)`

ISR of INT7

Definition at line 218 of file EXTI.c.

5.47.2.9 `EXTI_DisableExternalInterrupt()` `void EXTI_DisableExternalInterrupt (`
`const EXTI_t extiNumber)`

Disable an EXTI pin (EXTI0 - EXTI7)

Parameters

<code>in</code>	<code>extiNumber</code>	The EXTI pin to disable (EXTI0 - EXTI7)
-----------------	-------------------------	---

Definition at line 501 of file EXTI.c.

5.47.2.10 `EXTI_EnableExternalInterrupt()` `void EXTI_EnableExternalInterrupt (`
`const EXTI_t extiNumber)`

Enable an EXTI pin (EXTI0 - EXTI7)

Enable external interrupt pin

Definition at line 453 of file EXTI.c.

Here is the call graph for this function:



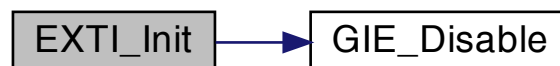
5.47.2.11 EXTI_Init() void EXTI_Init (
 const EXTI_t extiNumber,
 const EXTI_SENSITIVITY_t sensitivity,
 void(*) (void) callbackPtr)

Initialize an EXTI pin with a given sensitivity and callback function.

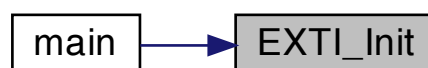
Initialize an external interrupt pin as input pin and set the

Definition at line 434 of file EXTI.c.

Here is the call graph for this function:



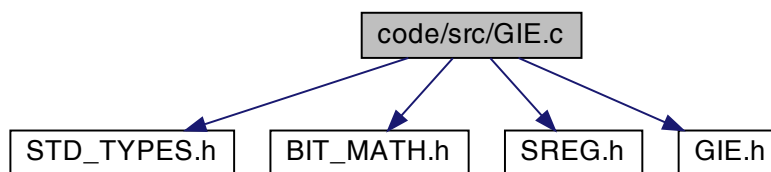
Here is the caller graph for this function:



5.48 code/src/GIE.c File Reference

Global Interrupt Enable (GIE)

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "SREG.h"
#include "GIE.h"
Include dependency graph for GIE.c:
```



Functions

- void [GIE_Disable](#) (void)
Global Interrupt Disable (GID)
- void [GIE_Enable](#) (void)
Global Interrupt Enable (GIE)

5.48.1 Detailed Description

Global Interrupt Enable (GIE)

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-02-11

Copyright

Copyright (c) 2022

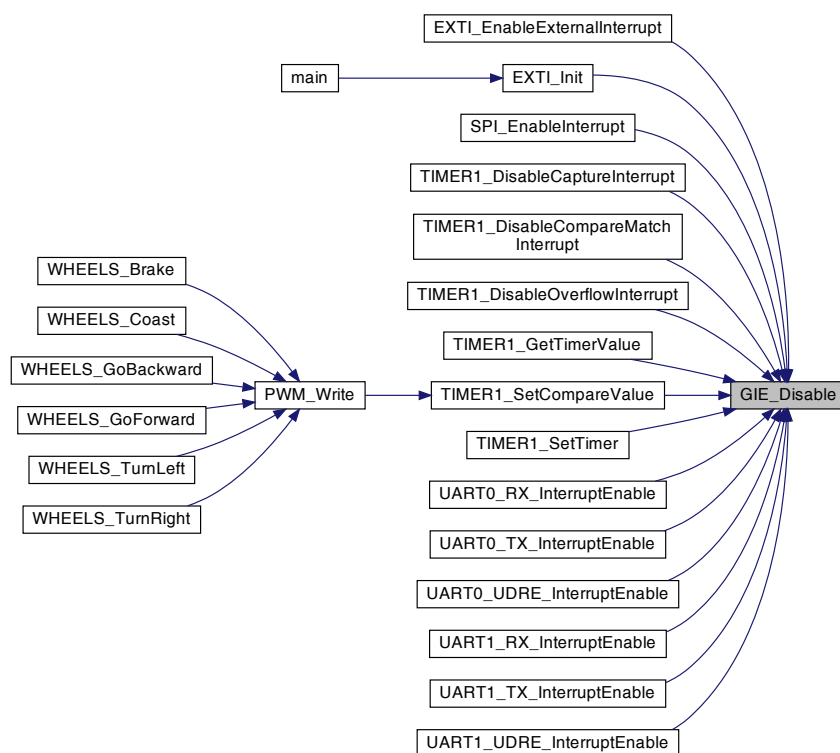
5.48.2 Function Documentation

5.48.2.1 GIE_Disable() `void GIE_Disable (void)`

Global Interrupt Disable (GID)

Definition at line 14 of file GIE.c.

Here is the caller graph for this function:

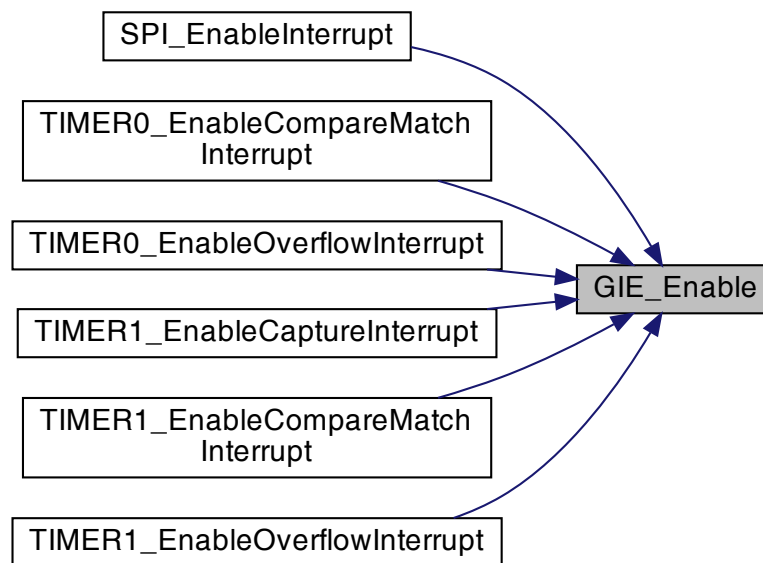


5.48.2.2 GIE_Enable() `void GIE_Enable (void)`

Global Interrupt Enable (GIE)

Definition at line 18 of file GIE.c.

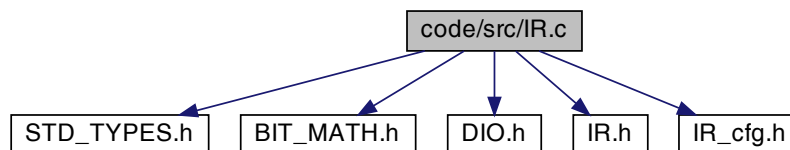
Here is the caller graph for this function:



5.49 code/src/IR.c File Reference

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "DIO.h"
#include "IR.h"
#include "IR_cfg.h"
```

Include dependency graph for IR.c:



Functions

- void `IR_Init` (void)
Initialize IR Sensors Configurations.
- `STATE_t IR_GetStatus` (`IR_SENSOR_t` sensor)
Get status of a specific IR Sensor.
- `u32 IR_ScanAll` (void)
Scan all IR Sensors and return the status of all of them.
- `u8 IR_GetCount` (void)
Get the number of configured IR sensors.

5.49.1 Function Documentation

5.49.1.1 IR_GetCount() `u8 IR_GetCount (void)`

Get the number of configured IR sensors.

Returns

Number of configured IR sensors

Example:

```
IR_GetNumOfSensors(); // returns 2 if only IR_0 and IR_1 are configured
```

Definition at line 51 of file IR.c.

5.49.1.2 IR_GetStatus() `STATE_t IR_GetStatus (IR_SENSOR_t sensor)`

Get status of a specific IR Sensor.

Parameters

in	<i>sensor</i>	Number of the sensor to be read; IR_0, IR_1, ..., IR7
----	---------------	---

Returns

State of the sensor; HIGH if White, LOW if black.

Example:

```
IR_GetStatus(IR_0); // \ref HIGH if IR_0 is on white track and \ref LOW if black track, member of  
                     \ref STATE_t enum
```

Definition at line 18 of file IR.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.49.1.3 IR_Init() `void IR_Init (`
`void)`

Initialize IR Sensors Configurations.

API FUNCTIONS

Definition at line 14 of file IR.c.

Here is the caller graph for this function:



5.49.1.4 IR_ScanAll() `u32 IR_ScanAll (`
`void)`

Scan all IR Sensors and return the status of all of them.

Returns

State of all sensors; HIGH if White, LOW if black.

Example:

```
IR_ScanAll();  
// when it returns a 0x0F00FFF0(0B0000_1111_0000_0000_1111_1111_1111_0000):  
// This means:  
// * (IR_0 --> IR_15) and (IR_24 --> IR_27) are on white track.  
// * (IR_16 --> IR_23) and (IR_28 --> IR_31) are on black track.
```

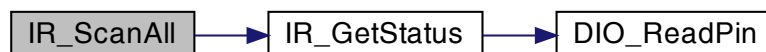
Note

This function:

- Only scans the configured sensors. So, if you have configured only IR_0 and IR_1, this function will only return a maximum of 0x00000003 (0B0000_0000_0000_0000_0000_0000_0011)
- Scans in the order of sensors' configurations. So, if you have configured IR_1 before IR_0 in the configuration, the status of IR_1 will be returned on bit 0 and the status of IR_0 will on bit 1.

Definition at line 41 of file IR.c.

Here is the call graph for this function:

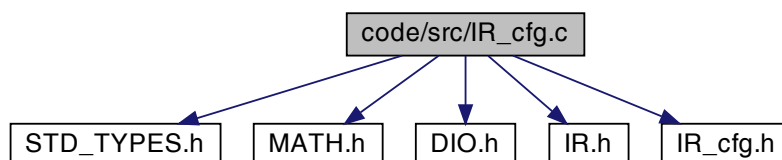
**5.50 code/src/IR_cfg.c File Reference**

```

#include "STD_TYPES.h"
#include "MATH.h"
#include "DIO.h"
#include "IR.h"
#include "IR_cfg.h"

```

Include dependency graph for IR_cfg.c:

**Variables**

- `IR_CONFIG_t IR_configs []`
- `const u8 countIRSensorsConfigured = SIZE_OF_ARRAY(IR_configs)`

5.50.1 Variable Documentation

5.50.1.1 countIRSensorsConfigured `const u8 countIRSensorsConfigured = SIZE_OF_ARRAY(IR_configs)`

Definition at line 28 of file IR_cfg.c.

5.50.1.2 IR_configs `IR_CONFIG_t IR_configs[]`

Initial value:

```
= {
    {IR_0, PIN_2, PORT_E, ACTIVE_LOW},
    {IR_1, PIN_3, PORT_E, ACTIVE_LOW},
    {IR_2, PIN_4, PORT_E, ACTIVE_LOW}
}
```

Note

ACTIVE_LOW means that the pin is:

- LOW when the sensor is detecting an object (white)
- HIGH when the sensor is not detecting an object (black)
- ACTIVE_HIGH means that the pin is:
- HIGH when the sensor is detecting an object (white)
- LOW when the sensor is not detecting an object (black)

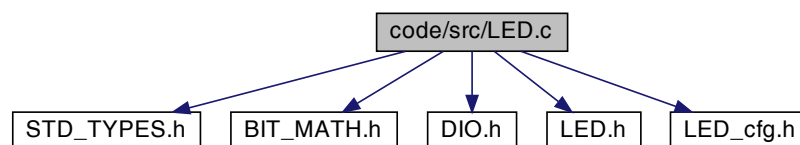
Definition at line 22 of file IR_cfg.c.

5.51 code/src/LED.c File Reference

LED Driver.

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "DIO.h"
#include "LED.h"
#include "LED_cfg.h"
```

Include dependency graph for LED.c:



Functions

- void [LED_Init](#) (void)
Initialize LEDs Configurations.
- void [LED_TurnOnOff](#) (const [LED_t](#) led, const [LED_STATE_t](#) state)
Turn on/off a specific LED.
- void [LED_Toggle](#) (const [LED_t](#) led)
Toggle state of a specific LED.

5.51.1 Detailed Description

LED Driver.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-02-01

5.51.2 Function Documentation

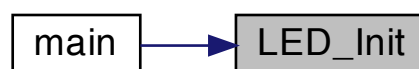
5.51.2.1 LED_Init() `void LED_Init (`
 `void)`

Initialize LEDs Configurations.

Initializes LEDs connected to DIO.

Definition at line 17 of file LED.c.

Here is the caller graph for this function:



5.51.2.2 LED_Toggle() `void LED_Toggle (`
 `const LED_t led)`

Toggle state of a specific LED.

Parameters

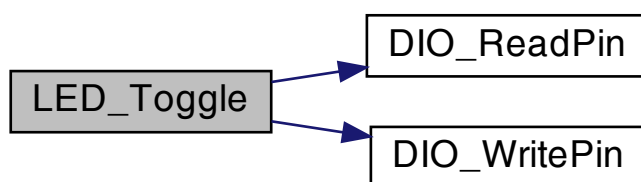
in	<i>led</i>	The LED to be oggles
----	------------	----------------------

Example:

```
LED_Toggle(LED_0); // toggle LED_0: LED_0 will be turned on if it was off and vice versa
```

Definition at line 45 of file LED.c.

Here is the call graph for this function:



5.51.2.3 LED_TurnOnOff() void LED_TurnOnOff (
const LED_t led,
const LED_STATE_t state)

Turn on/off a specific LED.

Turns on/off a specific LED.

Parameters

in	<i>led</i>	The LED to be turned on/off
in	<i>state</i>	The state of the LED, either LED_ON or LED_OFF

Example:

```
LED_Set(LED_0, LED_ON); // turn on LED_0  
LED_Set(LED_0, LED_OFF); // turn off LED_0
```

Definition at line 29 of file LED.c.

Here is the call graph for this function:

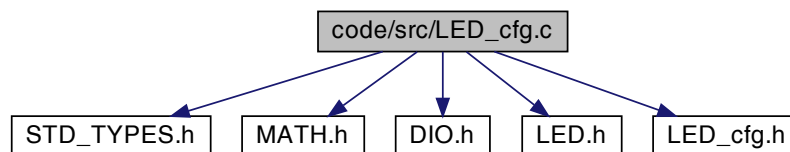


5.52 code/src/LED_cfg.c File Reference

Configuration source file for [LED.c](#).

```
#include "STD_TYPES.h"
#include "MATH.h"
#include "DIO.h"
#include "LED.h"
#include "LED_cfg.h"
```

Include dependency graph for LED_cfg.c:



Variables

- [LED_CONFIGS_t](#) ledConfigs []
- const [u8](#) countLedsConfigured = [SIZE_OF_ARRAY](#)(ledConfigs)

5.52.1 Detailed Description

Configuration source file for [LED.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.52.2 Variable Documentation

5.52.2.1 countLedsConfigured `const u8 countLedsConfigured = SIZE_OF_ARRAY(ledConfigs)`

Definition at line 19 of file LED_cfg.c.

5.52.2.2 ledConfigs `LED_CONFIGS_t ledConfigs[]`

Initial value:

```
= {
    {LED_0, PIN_6, PORT_F},
}
```

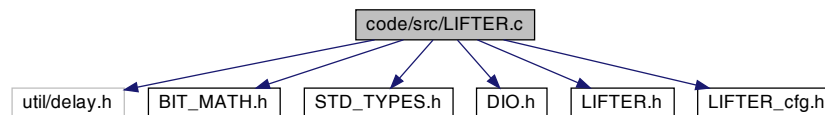
Definition at line 15 of file LED_cfg.c.

5.53 code/src/LIFTER.c File Reference

Lifter module that control the movement of the lifter actuator.

```
#include <util/delay.h>
#include "BIT_MATH.h"
#include "STD_TYPES.h"
#include "DIO.h"
#include "LIFTER.h"
#include "LIFTER_cfg.h"
```

Include dependency graph for LIFTER.c:



Functions

- void `LIFTER_Init` (void)
Initialize the LIFTER module.
- void `LIFTER_MoveUp` (void)
Move the lifter up until it reaches the top.
- void `LIFTER_MoveDown` (void)
Move the lifter down until it reaches the bottom.
- void `LIFTER_Enable` (void)
Enable the lifter motor.
- void `LIFTER_Disable` (void)
Disable the lifter motor.
- void `LIFTER_SetSpeed` (const u8 speed)
Set the speed of the lifter motor.
- void `LIFTER_SetOverallStroke` (const u8 overallStroke)
Set the stroke of the lifter motor per revolution in mm.
- void `LIFTER_SetPulsesPerRevolution` (const u16 pulsesPerRevolution)
Set the number of pulses per revolution.
- void `LIFTER_SetRevolutionStroke` (const u8 revolutionStroke)
Set the distance per revolution.

5.53.1 Detailed Description

Lifter module that control the movement of the lifter actuator.

Author

Mahmoud Karam (ma.karam272@gmail.com)

This module uses TB6560 motor driver to control the LIFTER. TB6560 is configured to use a stepper motor. And connected in common-anode mode: PUL-, DIR-, and EN- are connected to GND.

Version

1.0.0

Date

2022-02-09

Copyright

Copyright (c) 2022

5.53.2 Function Documentation

5.53.2.1 LIFTER_Disable() `void LIFTER_Disable (`
`void)`

Disable the lifter motor.

Definition at line 131 of file LIFTER.c.

Here is the call graph for this function:



5.53.2.2 LIFTER_Enable() `void LIFTER_Enable (`
`void)`

Enable the lifter motor.

Definition at line 124 of file LIFTER.c.

Here is the call graph for this function:



5.53.2.3 LIFTER_Init() `void LIFTER_Init (`
`void)`

Initialize the LIFTER module.

Initialize the lifter.

API FUNCTIONS IMPLEMENTATION *

Definition at line 103 of file LIFTER.c.

Here is the caller graph for this function:



5.53.2.4 LIFTER_MoveDown() `void LIFTER_MoveDown (`
`void)`

Move the lifter down until it reaches the bottom.

Move the lifter down.

Definition at line 117 of file LIFTER.c.

5.53.2.5 LIFTER_MoveUp() `void LIFTER_MoveUp (`
`void)`

Move the lifter up until it reaches the top.

Move the lifter up.

Definition at line 110 of file LIFTER.c.

5.53.2.6 LIFTER_SetOverallStroke() `void LIFTER_SetOverallStroke (`
`const u8 overallStroke)`

Set the stroke of the lifter motor per revolution in mm.

Parameters

in	<i>overallStroke</i>	The stroke of the lifter motor per revolution in mm
----	----------------------	---

Definition at line 147 of file LIFTER.c.

5.53.2.7 LIFTER_SetPulsesPerRevolution() `void LIFTER_SetPulsesPerRevolution (`
`const u16 pulsesPerRevolution)`

Set the number of pulses per revolution.

Parameters

in	<i>pulsesPerRevolution</i>	The number of pulses per revolution
----	----------------------------	-------------------------------------

Definition at line 155 of file LIFTER.c.

5.53.2.8 LIFTER_SetRevolutionStroke() `void LIFTER_SetRevolutionStroke (`
`const u8 revolutionStroke)`

Set the distance per revolution.

Parameters

in	<i>revolutionStroke</i>	The distance per revolution in mm
----	-------------------------	-----------------------------------

Definition at line 163 of file LIFTER.c.

5.53.2.9 LIFTER_SetSpeed() `void LIFTER_SetSpeed (`
`const u8 speed)`

Set the speed of the lifter motor.

Parameters

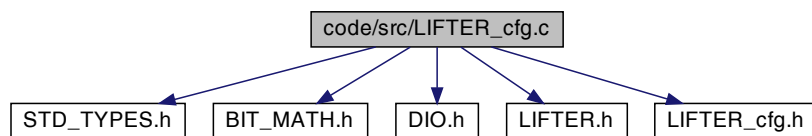
in	<i>speed</i>	The speed of the lifter motor
----	--------------	-------------------------------

Definition at line 139 of file LIFTER.c.

5.54 code/src/LIFTER_cfg.c File Reference

Configuration source file for [LIFTER.c](#).

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "DIO.h"
#include "LIFTER.h"
#include "LIFTER_cfg.h"
Include dependency graph for LIFTER_cfg.c:
```



Variables

- [LIFTER_CONFIGS_t LifterConfigs](#)
Lifter configurations.

5.54.1 Detailed Description

Configuration source file for [LIFTER.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.54.2 Variable Documentation

5.54.2.1 LifterConfigs `LIFTER_CONFIGS_t` LifterConfigs

Initial value:

```
= {
    .directionPin      = PIN_4,
    .directionPinPort  = PORT_D,
    .pulsePin         = PIN_5,
    .pulsePinPort     = PORT_D,
    .enablePin        = PIN_6,
    .enablePinPort    = PORT_D,
    .overallStroke     = (u8)20,
    .pulsesPerRevolution = (u16)1600,
    .revolutionStroke  = (u8)2,
    .speed            = (u8)4,
}
```

Lifter configurations.

Note

stepSize is the number of millimeters that the lifter will move when calling LIFTER_move function.

For example, if the stepSize is 10, the lifter will move 10 millimeters when calling LIFTER_move function.

pulsePerRevolution is the number of pulses that the lifter will make to make one revolution.

For example, if the pulsePerRevolution is 200, the lifter will make 200 pulses to make one revolution. Options:

- 200 --> Full step
- 400 --> Half step
- 800 --> 1/4 step --> Not allowed with TB6560
- 1600 --> 1/8 step
- 3200 --> 1/16 step
- 6400 --> 1/32 step --> Not allowed with TB6560

distancePerRevolution is the number of millimeters that the lifter will move in one revolution.

For example, if the distancePerRevolution is 100, the lifter will move 100 millimeters in one revolution.

speed is the number of millimeters that the lifter will move in one second. For example, if the speed is 100, the lifter will move 100 millimeters in one second.

speed must be less \leq stepSize

You can change default speed, overallStroke, pulsePerRevolution, and revolutionStroke by calling functions LIFTER_SetSpeed, LIFTER_SetOverallStroke, LIFTER_SetPulsesPerRevolution, and LIFTER_SetRevolutionStroke. See prototypes of these functions for more details.

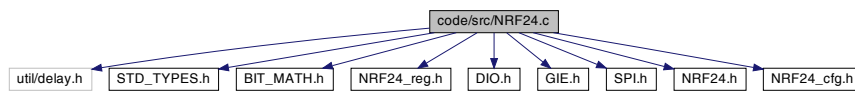
Definition at line 46 of file LIFTER_cfg.c.

5.55 code/src/NRF24.c File Reference

NRF24 wireless transceiver module driver.

```
#include "util/delay.h"
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "NRF24_reg.h"
#include "DIO.h"
#include "GIE.h"
#include "SPI.h"
#include "NRF24.h"
#include "NRF24_cfg.h"
```

Include dependency graph for NRF24.c:



Functions

- void [NRF24_Init](#) (void)
- void [NRF24_TxMode](#) (void)
- void [NRF24_SendString](#) (u8 *data, u8 len)
- void [NRF24_RxMode](#) (void)
- u8 [NRF24_Available](#) (void)
- [ERROR_STATUS_t](#) [NRF24_ReceiveString](#) (u8 *data, u8 length)

5.55.1 Detailed Description

NRF24 wireless transceiver module driver.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.55.2 Function Documentation

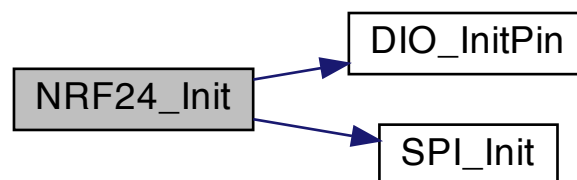
5.55.2.1 NRF24_Available() `u8 NRF24_Available (void)`

Definition at line 307 of file NRF24.c.

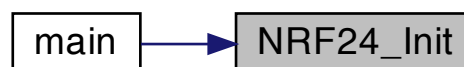
5.55.2.2 NRF24_Init() `void NRF24_Init (void)`

Definition at line 155 of file NRF24.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.55.2.3 NRF24_ReceiveString() `ERROR_STATUS_t NRF24_ReceiveString (u8 * data, u8 length)`

Definition at line 328 of file NRF24.c.

5.55.2.4 NRF24_RxMode() `void NRF24_RxMode (`
`void)`

Definition at line 249 of file NRF24.c.

5.55.2.5 NRF24_SendString() `void NRF24_SendString (`
`u8 * data,`
`u8 len)`

Definition at line 214 of file NRF24.c.

5.55.2.6 NRF24_TxMode() `void NRF24_TxMode (`
`void)`

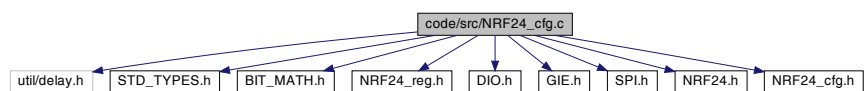
Definition at line 191 of file NRF24.c.

5.56 code/src/NRF24_cfg.c File Reference

Configuration source file for [NRF24.c](#).

```
#include "util/delay.h"
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "NRF24_reg.h"
#include "DIO.h"
#include "GIE.h"
#include "SPI.h"
#include "NRF24.h"
#include "NRF24_cfg.h"
```

Include dependency graph for NRF24_cfg.c:



Variables

- [NRF24_t NRF24_cfg](#)

5.56.1 Detailed Description

Configuration source file for [NRF24.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.56.2 Variable Documentation

5.56.2.1 NRF24_cfg [NRF24_t](#) NRF24_cfg

Initial value:

```
= {
    .pins = {
        .ce = { .pin = PIN_7, .port = PORT_E },
        .csn = { .pin = PIN_0, .port = PORT_B }
    },
    .channel = 76,
    .payload_len = 20,
    .addressWidth = 5,
    .txAddress = { 0xE7, 0xE7, 0xE7, 0xE7, 0xE7 },
    .rx0Address = { 0xE7, 0xD3, 0xF0, 0x35, 0x77 },
    .rx1Address = { 0xC2, 0xC2, 0xC2, 0xC2, 0xC2 },
    .rx2Address = { 0xC3, 0xC2, 0xC2, 0xC2, 0xC2 },
    .rx3Address = { 0xC4, 0xC2, 0xC2, 0xC2, 0xC2 },
    .rx4Address = { 0xC5, 0xC2, 0xC2, 0xC2, 0xC2 },
    .rx5Address = { 0xC6, 0xC2, 0xC2, 0xC2, 0xC2 },
    .rxPipe = RX_PIPE1,
}
```

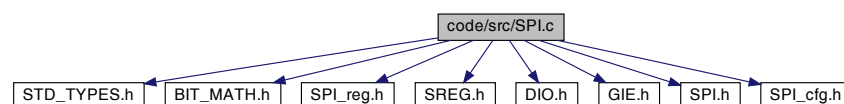
Definition at line 19 of file NRF24_cfg.c.

5.57 code/src/SPI.c File Reference

SPI driver for ATMEGA128 microcontroller.

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "SPI_reg.h"
#include "SREG.h"
#include "DIO.h"
#include "GIE.h"
#include "SPI.h"
#include "SPI_cfg.h"
```

Include dependency graph for SPI.c:



Functions

- void [SPI_SetCallback](#) (void(*ptrCallback)(void))
- void [__vector_17](#) (void)
- void [SPI_Init](#) (void)
- void [SPI_EnableInterrupt](#) (void(*ptrCallback)(void))
- void [SPI_DisableInterrupt](#) (void)
- [ERROR_STATUS_t SPI_SendByte](#) (const [u8](#) data)
- [ERROR_STATUS_t SPI_SendString](#) (const [u8](#) *str, [u8](#) length)
- [ERROR_STATUS_t SPI_ReceiveByte](#) ([u8](#) *const data)
- void [SPI_TransceiveByte](#) (const [u8](#) dataToSend, [u8](#) *const dataReceived)
- [ERROR_STATUS_t SPI_ReceiveString](#) ([u8](#) *const str, [u8](#) length)

Variables

- void(* [SPI_StcCallback](#))(void)

5.57.1 Detailed Description

SPI driver for ATMEGA128 microcontroller.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-03 @features: - SPI Master mode

- SPI Slave mode
- SPI Interrupt mode
- SPI DMA mode
- SPI Double Speed mode

Copyright

Copyright (c) 2022

5.57.2 Function Documentation

5.57.2.1 `__vector_17()` `void __vector_17 (`
`void)`

Definition at line 37 of file SPI.c.

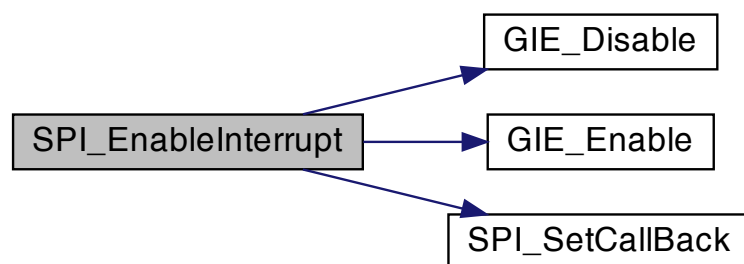
5.57.2.2 `SPI_DisableInterrupt()` `void SPI_DisableInterrupt (`
`void)`

Definition at line 238 of file SPI.c.

5.57.2.3 `SPI_EnableInterrupt()` `void SPI_EnableInterrupt (`
`void(*) (void) ptrCallback)`

Definition at line 229 of file SPI.c.

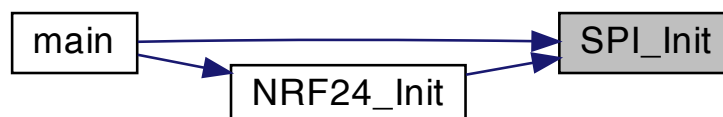
Here is the call graph for this function:



5.57.2.4 `SPI_Init()` `void SPI_Init (`
`void)`

Definition at line 212 of file SPI.c.

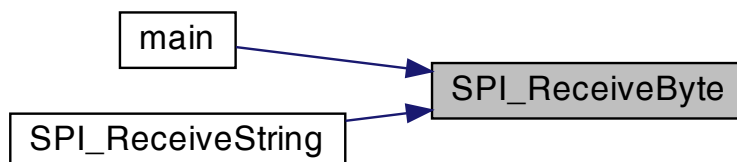
Here is the caller graph for this function:



5.57.2.5 SPI_ReceiveByte() `ERROR_STATUS_t SPI_ReceiveByte (`
`u8 *const data)`

Definition at line 273 of file SPI.c.

Here is the caller graph for this function:



5.57.2.6 SPI_ReceiveString() `ERROR_STATUS_t SPI_ReceiveString (`
`u8 *const str,`
`u8 length)`

Definition at line 293 of file SPI.c.

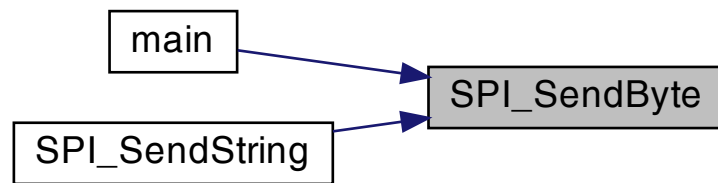
Here is the call graph for this function:



5.57.2.7 SPI_SendByte() `ERROR_STATUS_t SPI_SendByte (`
`const u8 data)`

Definition at line 242 of file SPI.c.

Here is the caller graph for this function:



5.57.2.8 SPI_SendString() `ERROR_STATUS_t SPI_SendString (`
 `const u8 * str,`
 `u8 length)`

Definition at line 258 of file SPI.c.

Here is the call graph for this function:



5.57.2.9 SPI_SetCallBack() `void SPI_SetCallBack (`
 `void(*) (void) ptrCallback)`

Definition at line 29 of file SPI.c.

Here is the caller graph for this function:



5.57.2.10 SPI_TrancieveByte() `void SPI_TrancieveByte (`
`const u8 dataToSend,`
`u8 *const dataReceived)`

Definition at line 285 of file SPI.c.

5.57.3 Variable Documentation

5.57.3.1 SPI_StcCallBack `void(* SPI_StcCallBack) (void) (`
`void)`

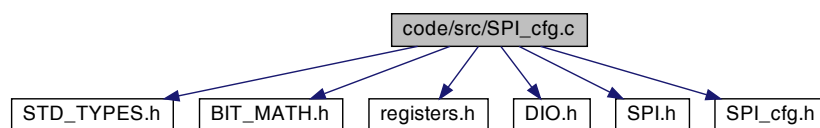
Definition at line 27 of file SPI.c.

5.58 code/src/SPI_cfg.c File Reference

Configuration source file for [SPI.c](#).

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "registers.h"
#include "DIO.h"
#include "SPI.h"
#include "SPI_cfg.h"
```

Include dependency graph for SPI_cfg.c:



Variables

- [SPI_CONFIG_t SPI_Config](#)

5.58.1 Detailed Description

Configuration source file for [SPI.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-20

Copyright

Copyright (c) 2022

5.58.2 Variable Documentation

5.58.2.1 SPI_Config `SPI_CONFIG_t` SPI_Config

Initial value:

```
= {
    .connections = {
        .SS      = { .pin = PIN_0, .port = PORT_B},
        .SCK     = { .pin = PIN_1, .port = PORT_B},
        .MOSI    = { .pin = PIN_2, .port = PORT_B},
        .MISO    = { .pin = PIN_3, .port = PORT_B}
    },
    .mode        = SPI_MASTER,
    .clockDivider = SPI_PRESCALER_8,
    .dataOrder   = SPI_DATA_ORDER_MSB_FIRST,
    .doubleSpeed = SPI_DOUBLE_SPEED_DISABLE,
    .clockMode   = SPI_MODE0,
}
```

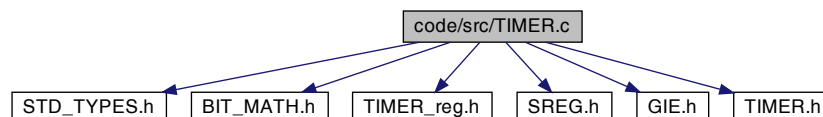
Definition at line 17 of file SPI_cfg.c.

5.59 code/src/TIMER.c File Reference

Timer driver for ATMEGA128 microcontroller.

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "TIMER_reg.h"
#include "SREG.h"
#include "GIE.h"
#include "TIMER.h"
```

Include dependency graph for TIMER.c:



Functions

- void `TIMER0_SetCallBack` (void(*ptr)(void))
- void `TIMER0_SetCompareCallBack` (void(*ptr)(void))
- void `TIMER1_SetCallBack` (void(*ptr)(void))
- void `TIMER1_SetCompareCallBackA` (void(*ptr)(void))
- void `TIMER1_SetCompareCallBackB` (void(*ptr)(void))
- void `TIMER1_SetCompareCallBackC` (void(*ptr)(void))
- void `TIMER1_SetCaptureCallBack` (void(*ptr)(void))
- void `TIMER2_SetCallBack` (void(*ptr)(void))
- void `TIMER2_SetCompareCallBack` (void(*ptr)(void))
- void `TIMER3_SetCallBack` (void(*ptr)(void))
- void `TIMER3_SetCompareCallBackA` (void(*ptr)(void))

- void `TIMER3_SetCompareCallBackB` (void(*ptr)(void))
- void `TIMER3_SetCompareCallBackC` (void(*ptr)(void))
- void `TIMER3_SetCaptureCallBack` (void(*ptr)(void))
- void `TIMER0_Init` (u8 initValue, `TIMER_CLOCK_t` clock, `TIMER_MODE_t` timerMode, `TIMER_OC_t` compareMode)
- void `TIMER0_SetCompareValue` (u8 u8CompareValue)
- void `TIMER0_SetTimer` (u8 u8TimerValue)
- void `TIMER0_EnableOverflowInterrupt` (void(*callBackPtr)(void))
- void `TIMER0_DisableOverflowInterrupt` (void)
- void `TIMER0_EnableCompareMatchInterrupt` (void(*callBackPtr)(void))
- void `TIMER0_DisableCompareMatchInterrupt` (void)
- u8 `TIMER0_GetTimerValue` (void)
- void `PWM0_Write` (u8 dutyPercentage)
- void `TIMER1_Init` (u16 initValue, `TIMER_CLOCK_t` clock, `TIMER_MODE_t` timerMode, `TIMER_OC_t` compareMode, `TIMER_OCx_t` OCx)
- void `TIMER1_SetCompareValue` (u16 u16CompareValue, `TIMER_OCx_t` OCx)
- void `TIMER1_SetTimer` (u16 u16TimerValue)
- void `TIMER1_EnableOverflowInterrupt` (void(*callBackPtr)(void))
- void `TIMER1_DisableOverflowInterrupt` (void)
- void `TIMER1_EnableCompareMatchInterrupt` (`TIMER_OCx_t` OCx, void(*callBackPtr)(void))
- void `TIMER1_DisableCompareMatchInterrupt` (`TIMER_OCx_t` OCx)
- void `TIMER1_EnableCaptureInterrupt` (void(*callBackPtr)(void))
- void `TIMER1_DisableCaptureInterrupt` (void)
- u16 `TIMER1_GetTimerValue` (void)
- void `PWM_Init` (`PWM_t` channel, u32 frequency)
- void `PWM_Write` (`PWM_t` channel, u8 dutyPercentage)
- void `__vector_16` (void)
- void `__vector_15` (void)
- void `__vector_14` (void)
- void `__vector_13` (void)
- void `__vector_12` (void)
- void `__vector_24` (void)
- void `__vector_11` (void)
- void `__vector_10` (void)
- void `__vector_9` (void)
- void `__vector_29` (void)
- void `__vector_28` (void)
- void `__vector_27` (void)
- void `__vector_26` (void)
- void `__vector_25` (void)

5.59.1 Detailed Description

Timer driver for ATMEGA128 microcontroller.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-02-11

Copyright

Copyright (c) 2022

5.59.2 Function Documentation

5.59.2.1 `__vector_10()` `void __vector_10 (`
`void)`

Definition at line 914 of file TIMER.c.

5.59.2.2 `__vector_11()` `void __vector_11 (`
`void)`

Definition at line 901 of file TIMER.c.

5.59.2.3 `__vector_12()` `void __vector_12 (`
`void)`

Definition at line 875 of file TIMER.c.

5.59.2.4 `__vector_13()` `void __vector_13 (`
`void)`

Definition at line 862 of file TIMER.c.

5.59.2.5 `__vector_14()` `void __vector_14 (`
`void)`

Definition at line 849 of file TIMER.c.

5.59.2.6 `__vector_15()` `void __vector_15 (`
`void)`

Definition at line 836 of file TIMER.c.

5.59.2.7 `__vector_16()` `void __vector_16 (`
`void)`

Definition at line 823 of file TIMER.c.

5.59.2.8 `__vector_24()` `void __vector_24 (`
`void)`

Definition at line 888 of file TIMER.c.

5.59.2.9 `__vector_25()` `void __vector_25 (`
`void)`

Definition at line 992 of file TIMER.c.

5.59.2.10 `__vector_26()` `void __vector_26 (`
`void)`

Definition at line 979 of file TIMER.c.

5.59.2.11 `__vector_27()` `void __vector_27 (`
`void)`

Definition at line 966 of file TIMER.c.

5.59.2.12 `__vector_28()` `void __vector_28 (`
`void)`

Definition at line 953 of file TIMER.c.

5.59.2.13 `__vector_29()` `void __vector_29 (`
`void)`

Definition at line 940 of file TIMER.c.

5.59.2.14 `__vector_9()` `void __vector_9 (`
`void)`

Definition at line 927 of file TIMER.c.

5.59.2.15 PWM0_Write() `void PWM0_Write (`
 u8 *dutyPercentage* `)`

Definition at line 574 of file TIMER.c.

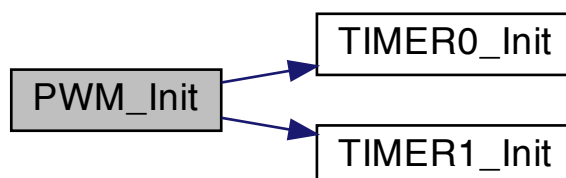
Here is the call graph for this function:



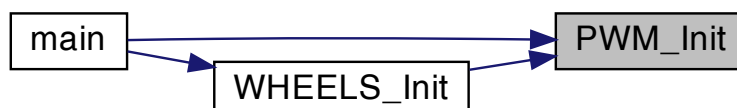
5.59.2.16 PWM_Init() `void PWM_Init (`
 PWM_t *channel*,
 u32 *frequency* `)`

Definition at line 755 of file TIMER.c.

Here is the call graph for this function:



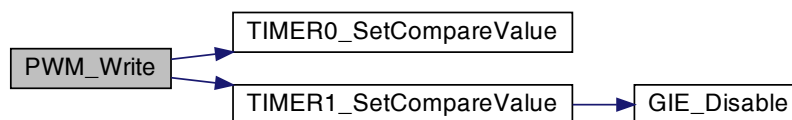
Here is the caller graph for this function:



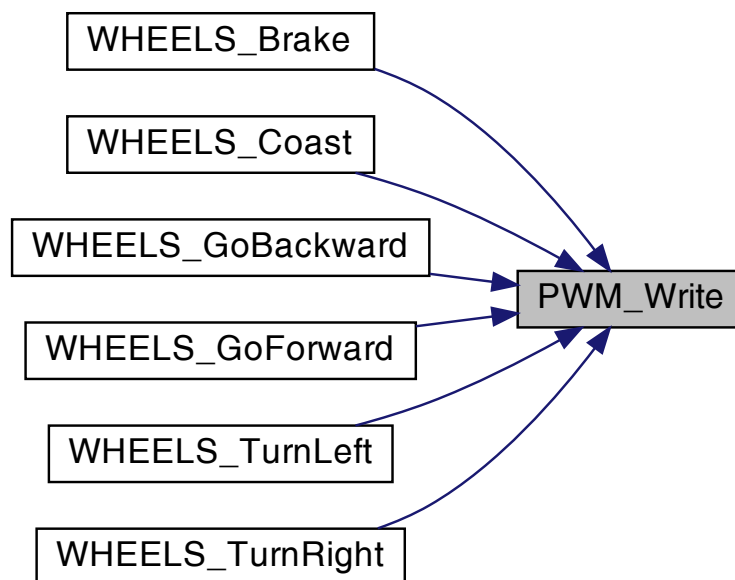
5.59.2.17 PWM_Write() `void PWM_Write (`
 PWM_t channel,
 u8 dutyPercentage)

Definition at line 793 of file TIMER.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.59.2.18 TIMER0_DisableCompareMatchInterrupt() `void TIMER0_DisableCompareMatchInterrupt (`
 void)

Definition at line 565 of file TIMER.c.

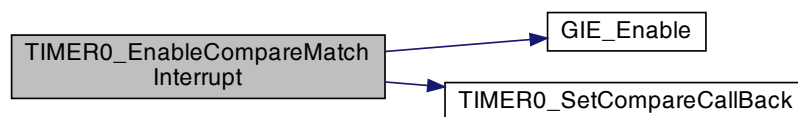
5.59.2.19 TIMER0_DisableOverflowInterrupt() `void TIMER0_DisableOverflowInterrupt (void)`

Definition at line 550 of file TIMER.c.

5.59.2.20 TIMER0_EnableCompareMatchInterrupt() `void TIMER0_EnableCompareMatchInterrupt (void(*) (void) callBackPtr)`

Definition at line 555 of file TIMER.c.

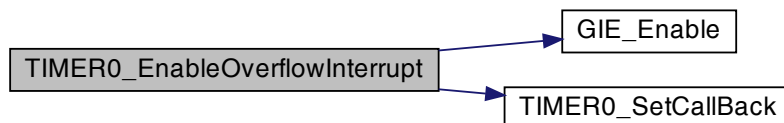
Here is the call graph for this function:



5.59.2.21 TIMER0_EnableOverflowInterrupt() `void TIMER0_EnableOverflowInterrupt (void(*) (void) callBackPtr)`

Definition at line 540 of file TIMER.c.

Here is the call graph for this function:



5.59.2.22 TIMER0_GetTimerValue() `u8 TIMER0_GetTimerValue (void)`

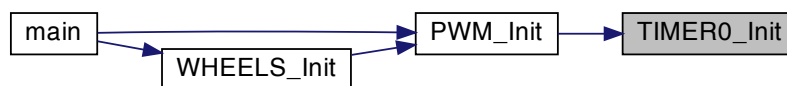
Definition at line 570 of file TIMER.c.

5.59.2.23 **TIMER0_Init()** `void TIMER0_Init (`
 u8 *initValue*,
 TIMER_CLOCK_t *clock*,
 TIMER_MODE_t *timerMode*,
 TIMER_OC_t *compareMode*)

TIMER0 Implementations

Definition at line 518 of file TIMER.c.

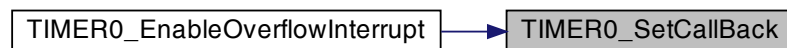
Here is the caller graph for this function:



5.59.2.24 **TIMER0_SetCallBack()** `void TIMER0_SetCallBack (`
 void(*) (void) *ptr*)

Definition at line 37 of file TIMER.c.

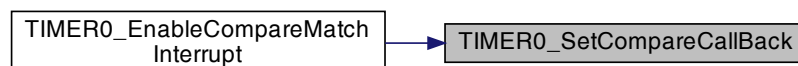
Here is the caller graph for this function:



5.59.2.25 **TIMER0_SetCompareCallBack()** `void TIMER0_SetCompareCallBack (`
 void(*) (void) *ptr*)

Definition at line 42 of file TIMER.c.

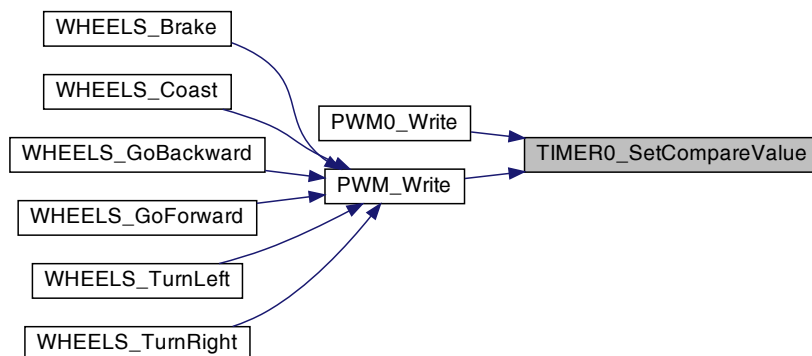
Here is the caller graph for this function:



5.59.2.26 TIMER0_SetCompareValue() `void TIMER0_SetCompareValue (`
`u8 u8CompareValue)`

Definition at line 532 of file TIMER.c.

Here is the caller graph for this function:



5.59.2.27 TIMER0_SetTimer() `void TIMER0_SetTimer (`
`u8 u8TimerValue)`

Definition at line 536 of file TIMER.c.

5.59.2.28 TIMER1_DisableCaptureInterrupt() `void TIMER1_DisableCaptureInterrupt (`
`void)`

Definition at line 724 of file TIMER.c.

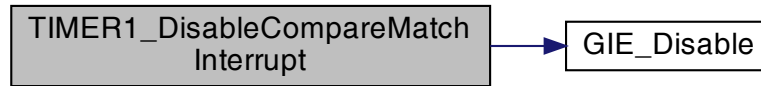
Here is the call graph for this function:



5.59.2.29 TIMER1_DisableCompareMatchInterrupt() `void TIMER1_DisableCompareMatchInterrupt (
 TIMER_OCx_t OCx)`

Definition at line 692 of file TIMER.c.

Here is the call graph for this function:



5.59.2.30 TIMER1_DisableOverflowInterrupt() `void TIMER1_DisableOverflowInterrupt (
 void)`

Definition at line 659 of file TIMER.c.

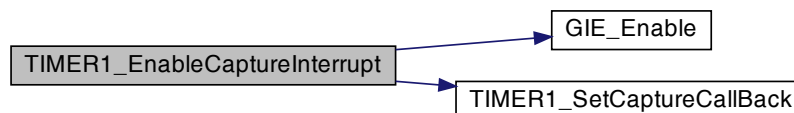
Here is the call graph for this function:



5.59.2.31 TIMER1_EnableCaptureInterrupt() `void TIMER1_EnableCaptureInterrupt (
 void(*) (void) callBackPtr)`

Definition at line 714 of file TIMER.c.

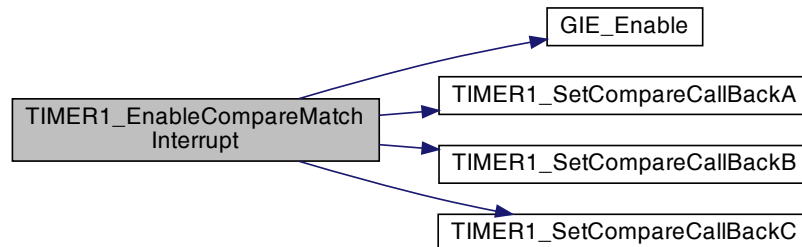
Here is the call graph for this function:



5.59.2.32 TIMER1_EnableCompareMatchInterrupt() `void TIMER1_EnableCompareMatchInterrupt (`
 `TIMER_OCx_t OCx,`
 `void(*) (void) callBackPtr)`

Definition at line 670 of file TIMER.c.

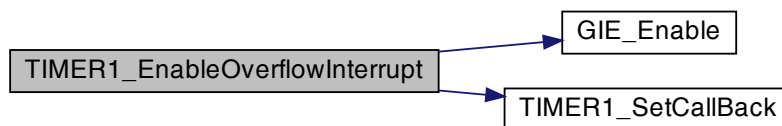
Here is the call graph for this function:



5.59.2.33 TIMER1_EnableOverflowInterrupt() `void TIMER1_EnableOverflowInterrupt (`
 `void(*) (void) callBackPtr)`

Definition at line 649 of file TIMER.c.

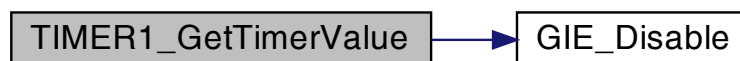
Here is the call graph for this function:



5.59.2.34 TIMER1_GetTimerValue() `u16 TIMER1_GetTimerValue (`
 `void)`

Definition at line 735 of file TIMER.c.

Here is the call graph for this function:

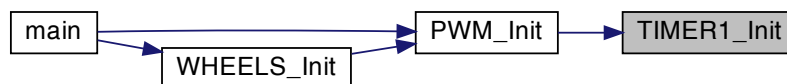


5.59.2.35 **TIMER1_Init()** `void TIMER1_Init (`
 `u16 initValue,`
 `TIMER_CLOCK_t clock,`
 `TIMER_MODE_t timerMode,`
 `TIMER_OC_t compareMode,`
 `TIMER_OCx_t OCx)`

TIMER1 Implementations

Definition at line 585 of file TIMER.c.

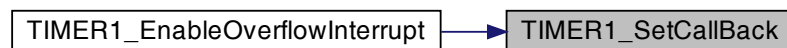
Here is the caller graph for this function:



5.59.2.36 **TIMER1_SetCallBack()** `void TIMER1_SetCallBack (`
 `void(*) (void) ptr)`

Definition at line 47 of file TIMER.c.

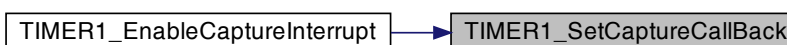
Here is the caller graph for this function:



5.59.2.37 **TIMER1_SetCaptureCallBack()** `void TIMER1_SetCaptureCallBack (`
 `void(*) (void) ptr)`

Definition at line 67 of file TIMER.c.

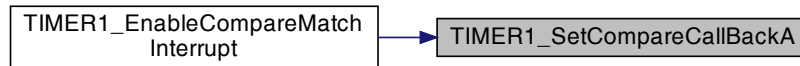
Here is the caller graph for this function:



5.59.2.38 TIMER1_SetCompareCallbackA() `void TIMER1_SetCompareCallBackA (`
`void(*) (void) ptr)`

Definition at line 52 of file TIMER.c.

Here is the caller graph for this function:



5.59.2.39 TIMER1_SetCompareCallbackB() `void TIMER1_SetCompareCallBackB (`
`void(*) (void) ptr)`

Definition at line 57 of file TIMER.c.

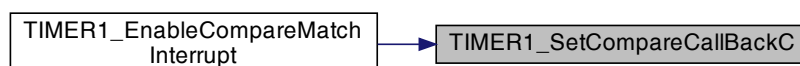
Here is the caller graph for this function:



5.59.2.40 TIMER1_SetCompareCallbackC() `void TIMER1_SetCompareCallBackC (`
`void(*) (void) ptr)`

Definition at line 62 of file TIMER.c.

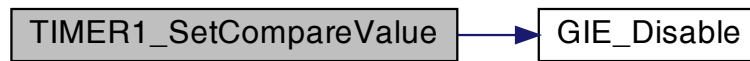
Here is the caller graph for this function:



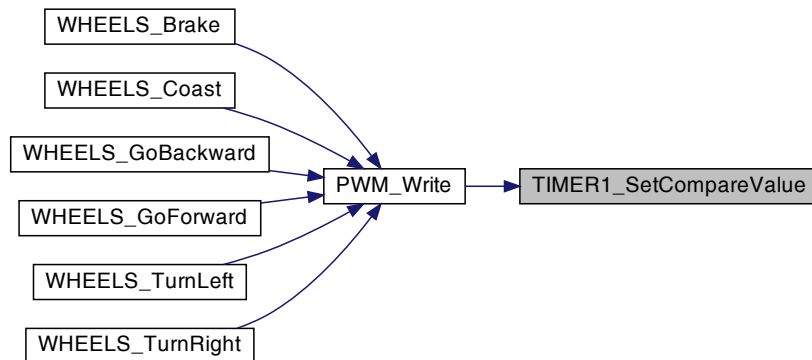
5.59.2.41 TIMER1_SetCompareValue() `void TIMER1_SetCompareValue (`
 u16 *u16CompareValue*,
 TIMER_OCx_t *OCx*)

Definition at line 613 of file TIMER.c.

Here is the call graph for this function:



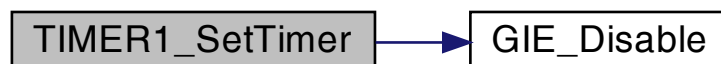
Here is the caller graph for this function:



5.59.2.42 TIMER1_SetTimer() `void TIMER1_SetTimer (`
 u16 *u16TimerValue*)

Definition at line 638 of file TIMER.c.

Here is the call graph for this function:



5.59.2.43 TIMER2_SetCallBack() void TIMER2_SetCallBack (
void(*) (void) ptr)

Definition at line 72 of file TIMER.c.

5.59.2.44 TIMER2_SetCompareCallBack() void TIMER2_SetCompareCallBack (
void(*) (void) ptr)

Definition at line 77 of file TIMER.c.

5.59.2.45 TIMER3_SetCallBack() void TIMER3_SetCallBack (
void(*) (void) ptr)

Definition at line 82 of file TIMER.c.

5.59.2.46 TIMER3_SetCaptureCallBack() void TIMER3_SetCaptureCallBack (
void(*) (void) ptr)

Definition at line 102 of file TIMER.c.

5.59.2.47 TIMER3_SetCompareCallBackA() void TIMER3_SetCompareCallBackA (
void(*) (void) ptr)

Definition at line 87 of file TIMER.c.

5.59.2.48 TIMER3_SetCompareCallBackB() void TIMER3_SetCompareCallBackB (
void(*) (void) ptr)

Definition at line 92 of file TIMER.c.

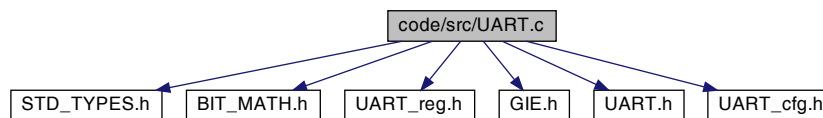
5.59.2.49 TIMER3_SetCompareCallBackC() void TIMER3_SetCompareCallBackC (
void(*) (void) ptr)

Definition at line 97 of file TIMER.c.

5.60 code/src/UART.c File Reference

UART driver for ATMEGA128 microcontroller.

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "UART_reg.h"
#include "GIE.h"
#include "UART.h"
#include "UART_cfg.h"
Include dependency graph for UART.c:
```



Typedefs

- typedef void(*volatile [UART_Callback_typedef_fn](#)) (void)

Functions

- void [__vector_18](#) (void)
- void [__vector_20](#) (void)
- void [__vector_19](#) (void)
- void [__vector_30](#) (void)
- void [__vector_32](#) (void)
- void [__vector_31](#) (void)
- void [UART0_Init](#) (void)
 - Initialize UART module 0 as configured in [UART_cfg.h](#) and [UART_cfg.c](#).*
- void [UART1_Init](#) (void)
 - Initialize UART module 1 as configured in [UART_cfg.h](#) and [UART_cfg.c](#).*
- void [UART0_Enable](#) (void)
 - Enable UART module 0 if it is disabled previously by [UART0_Disable\(\)](#)*
- void [UART1_Enable](#) (void)
 - Enable UART module 1 if it is disabled previously by [UART1_Disable\(\)](#)*
- void [UART0_Disable](#) (void)
 - Disable UART module 0 if it is enabled.*
- void [UART1_Disable](#) (void)
 - Disable UART module 1 if it is enabled.*
- void [UART0_SendByte](#) (const [u8](#) data)
 - Send a byte using UART module 0 Synchronously.*
- void [UART1_SendByte](#) (const [u8](#) data)
 - Send a byte using UART module 1.*
- void [UART0_Send9BitData](#) (const [u16](#) data)
 - Send 9 bits using UART module 0.*
- void [UART1_Send9BitData](#) (const [u16](#) data)

- Send 9 bits using UART module 1.*

 - void `UART0_SendByte_NoBlock` (const u8 data)

Send a byte using UART module 0 Asynchronously.
- void `UART1_SendByte_NoBlock` (const u8 data)

Send a byte using UART module 1 Asynchronously.
- void `UART0_Send9BitData_NoBlock` (const u16 data)

Send 9 bits using UART module 0 Asynchronously.
- void `UART1_Send9BitData_NoBlock` (const u16 data)

Send 9 bits using UART module 1 Asynchronously.
- `ERROR_STATUS_t` `UART0_ReceiveByte` (u8 *const data)

Receive a byte using UART module 0.
- `ERROR_STATUS_t` `UART1_ReceiveByte` (u8 *const data)

Receive a byte using UART module 1.
- `ERROR_STATUS_t` `UART0_Receive9BitData` (u16 *const data)

Receive a string of 16 bits using UART module 0 (9 bits data)
- `ERROR_STATUS_t` `UART1_Receive9BitData` (u16 *const data)

Receive a string of 16 bits using UART module 1 (9 bits data)
- `ERROR_STATUS_t` `UART0_ReceiveByte_NoBlock` (u8 *const data)

Receive a byte using UART module 0 Asynchronously.
- `ERROR_STATUS_t` `UART1_ReceiveByte_NoBlock` (u8 *const data)

Receive a byte using UART module 1 Asynchronously.
- `ERROR_STATUS_t` `UART0_Receive9BitData_NoBlock` (u16 *const data)
- `ERROR_STATUS_t` `UART1_Receive9BitData_NoBlock` (u16 *const data)
- `STATE_t` `UART0_Available` (void)

Check if there is a byte available in UART module 0.
- `STATE_t` `UART1_Available` (void)

Check if there is a byte available in UART module 1.
- void `UART0_Flush` (void)

The receiver buffer FIFO will be flushed when the receiver is disabled, i.e. the buffer will be emptied of its contents. Unread data will be lost. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDR I/O location until the RXC flag is cleared.
- void `UART1_Flush` (void)

Flush the receive buffer of UART module 1.
- void `UART0_RX_InterruptEnable` (void(*const ptrCallback)(void))

Enable the receive interrupt of UART module 0 with passing the function pointer to the interrupt service routine.
- void `UART1_RX_InterruptEnable` (void(*const ptrCallback)(void))

Enable the receive interrupt of UART module 1 with passing the function pointer to the interrupt service routine.
- void `UART0_TX_InterruptEnable` (void(*const ptrCallback)(void))

Enable the transmit interrupt of UART module 0 with passing the function pointer to the interrupt service routine.
- void `UART1_TX_InterruptEnable` (void(*const ptrCallback)(void))

Enable the transmit interrupt of UART module 1 with passing the function pointer to the interrupt service routine.
- void `UART0_UDRE_InterruptEnable` (void(*const ptrCallback)(void))

Enable the interrupt of Data Register Empty of UART module 0 with passing the function pointer to the interrupt service routine.
- void `UART1_UDRE_InterruptEnable` (void(*const ptrCallback)(void))

Enable the interrupt of Data Register Empty of UART module 1 with passing the function pointer to the interrupt service routine.
- void `UART0_RX_InterruptDisable` (void)

Disable the receive interrupt of UART module 0.
- void `UART1_RX_InterruptDisable` (void)

Disable the receive interrupt of UART module 1.
- void `UART0_TX_InterruptDisable` (void)

Disable the transmit interrupt of UART module 0.

- void [UART1_TX_InterruptDisable](#) (void)

Disable the transmit interrupt of UART module 1.

- void [UART0_UDRE_InterruptDisable](#) (void)

Disable the interrupt of Data Register Empty of UART module 0.

- void [UART1_UDRE_InterruptDisable](#) (void)

Disable the interrupt of Data Register Empty of UART module 1.

5.60.1 Detailed Description

UART driver for ATMEGA128 microcontroller.

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-03

Copyright

Copyright (c) 2022

5.60.2 Typedef Documentation

5.60.2.1 [UART_Callback_typedef_fn](#) typedef void(* volatile UART_Callback_typedef_fn) (void)

Definition at line 27 of file UART.c.

5.60.3 Function Documentation

5.60.3.1 [__vector_18\(\)](#) void __vector_18 (void)

Definition at line 40 of file UART.c.

5.60.3.2 `__vector_19()` `void __vector_19 (`
`void)`

Definition at line 70 of file UART.c.

5.60.3.3 `__vector_20()` `void __vector_20 (`
`void)`

Definition at line 55 of file UART.c.

5.60.3.4 `__vector_30()` `void __vector_30 (`
`void)`

Definition at line 85 of file UART.c.

5.60.3.5 `__vector_31()` `void __vector_31 (`
`void)`

Definition at line 115 of file UART.c.

5.60.3.6 `__vector_32()` `void __vector_32 (`
`void)`

Definition at line 100 of file UART.c.

5.60.3.7 `UART0_Available()` `STATE_t UART0_Available (`
`void)`

Check if there is a byte available in UART module 0.

Returns

HIGH if there is a byte available, LOW otherwise

Definition at line 625 of file UART.c.

Here is the caller graph for this function:



5.60.3.8 UART0_Disable() `void UART0_Disable (`
`void)`

Disable UART module 0 if it is enabled.

Definition at line 546 of file UART.c.

5.60.3.9 UART0_Enable() `void UART0_Enable (`
`void)`

Enable UART module 0 if it is disabled previously by [UART0_Disable\(\)](#)

Definition at line 535 of file UART.c.

5.60.3.10 UART0_Flush() `void UART0_Flush (`
`void)`

The receiver buffer FIFO will be flushed when the receiver is disabled, i.e. the buffer will be emptied of its contents. Unread data will be lost. If the buffer has to be flushed during normal operation, due to for instance an error condition, read the UDR I/O location until the RXC flag is cleared.

Flush the receive buffer of UART module 0.

Definition at line 640 of file UART.c.

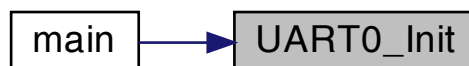
5.60.3.11 UART0_Init() `void UART0_Init (`
`void)`

Initialize UART module 0 as configured in [UART_cfg.h](#) and [UART_cfg.c](#).

It configure UART module 0 according to [UART_cfg.h](#) and [UART_cfg.c](#)
and then enable UART module 0

Definition at line 482 of file UART.c.

Here is the caller graph for this function:



5.60.3.12 UART0_Receive9BitData() `ERROR_STATUS_t UART0_Receive9BitData (`
`u16 * data)`

Receive a string of 16 bits using UART module 0 (9 bits data)

For Example: `UART0_Receive9BitString(string);` will receive a string of 16 bits

from UART module 0 and store it in variable `<string>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 599 of file UART.c.

5.60.3.13 UART0_Receive9BitData_NoBlock() `ERROR_STATUS_t UART0_Receive9BitData_NoBlock (`
`u16 *const data)`

Definition at line 616 of file UART.c.

5.60.3.14 UART0_ReceiveByte() `ERROR_STATUS_t UART0_ReceiveByte (`
`u8 * data)`

Receive a byte using UART module 0.

For Example: `UART0_ReceiveByte(&data);` will receive a byte from UART module 0

and store it in variable `<data>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 591 of file UART.c.

Here is the caller graph for this function:



5.60.3.15 UART0_ReceiveByte_NoBlock() `ERROR_STATUS_t UART0_ReceiveByte_NoBlock (u8 * data)`

Receive a byte using UART module 0 Asynchronously.

For Example: `UART0_ReceiveByte(&data);` will receive a byte from UART module 0

and store it in variable <data> without checking receive complete flag and without blocking the calling thread

Returns

The error code: ERROR_NO if no error, ERROR_YES if error

Definition at line 608 of file UART.c.

5.60.3.16 UART0_RX_InterruptDisable() `void UART0_RX_InterruptDisable (void)`

Disable the receive interrupt of UART module 0.

For Example: `UART0_RX_InterruptDisable();` will disable the receive interrupt

of UART module 0

Definition at line 714 of file UART.c.

5.60.3.17 UART0_RX_InterruptEnable() `void UART0_RX_InterruptEnable (void(*) (void) ptrCallback)`

Enable the receive interrupt of UART module 0 with passing the function pointer to the interrupt service routine.

Parameters

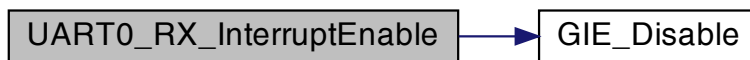
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The interrupt is triggered when a byte is received

Definition at line 648 of file UART.c.

Here is the call graph for this function:



5.60.3.18 UART0_Send9BitData() `void UART0_Send9BitData (`
`const u16 data)`

Send 9 bits using UART module 0.

For Example: `UART0_Send9Bits(0x123);` will send 9 bits 0x123 to UART module 0

Definition at line 565 of file UART.c.

5.60.3.19 UART0_Send9BitData_NoBlock() `void UART0_Send9BitData_NoBlock (`
`const u16 data)`

Send 9 bits using UART module 0 Asynchronously.

For Example: `UART0_Send9Bits_NoBlock(0x123);` will send 9 bits 0x123 to UART module 0
without blocking the calling thread

Definition at line 582 of file UART.c.

5.60.3.20 UART0_SendByte() `void UART0_SendByte (`
`const u8 data)`

Send a byte using UART module 0 Synchronously.

For Example: `UART0_SendByte('a');` will send character 'a' to UART module 0

Definition at line 557 of file UART.c.

Here is the caller graph for this function:



5.60.3.21 UART0_SendByte_NoBlock() `void UART0_SendByte_NoBlock (`
`const u8 data)`

Send a byte using UART module 0 Asynchronously.

For Example: `UART0_SendByte_NoBlock('a');` will send character 'a' to UART module 0
without blocking the calling thread

Definition at line 574 of file UART.c.

5.60.3.22 UART0_TX_InterruptDisable() `void UART0_TX_InterruptDisable (`
`void)`

Disable the transmit interrupt of UART module 0.

For Example: `UART0_TX_InterruptDisable();` will disable the transmit interrupt
of UART module 0

Definition at line 722 of file UART.c.

5.60.3.23 UART0_TX_InterruptEnable() `void UART0_TX_InterruptEnable (`
`void(*) (void) ptrCallback)`

Enable the transmit interrupt of UART module 1 with passing the function pointer to the interrupt service routine.

Parameters

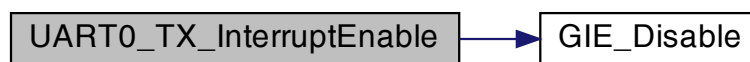
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The transmit interrupt will be triggered after transmission complete.

Definition at line 681 of file UART.c.

Here is the call graph for this function:



5.60.3.24 UART0_UDRE_InterruptDisable()

```
void UART0_UDRE_InterruptDisable (
    void )
```

Disable the interrupt of Data Register Empty of UART module 0.

For Example: `UART0_UDRE_InterruptDisable();` will disable the interrupt
of Data Register Empty of UART module 0

Definition at line 730 of file UART.c.

5.60.3.25 UART0_UDRE_InterruptEnable()

```
void UART0_UDRE_InterruptEnable (
    void(*) (void) ptrCallback )
```

Enable the interrupt of Data Register Empty of UART module 0 with passing the function pointer to the interrupt service routine.

Parameters

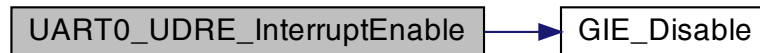
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The interrupt is triggered when the data register is empty

Definition at line 692 of file UART.c.

Here is the call graph for this function:



5.60.3.26 UART1_Available() `STATE_t UART1_Available (void)`

Check if there is a byte available in UART module 1.

Returns

HIGH if there is a byte available, LOW otherwise

Definition at line 629 of file UART.c.

Here is the caller graph for this function:



5.60.3.27 UART1_Disable() `void UART1_Disable (void)`

Disable UART module 1 if it is enabled.

Definition at line 551 of file UART.c.

5.60.3.28 UART1_Enable() `void UART1_Enable (`
`void)`

Enable UART module 1 if it is disabled previously by [UART1_Disable\(\)](#)

It enable UART module 1 if it is disabled previously by [UART1_Disable\(\)](#)

Definition at line 540 of file UART.c.

5.60.3.29 UART1_Flush() `void UART1_Flush (`
`void)`

Flush the receive buffer of UART module 1.

For Example: `UART1_Flush();` will flush the receive buffer of UART module 1

Definition at line 644 of file UART.c.

5.60.3.30 UART1_Init() `void UART1_Init (`
`void)`

Initialize UART module 1 as configured in [UART_cfg.h](#) and [UART_cfg.c](#).

It configure UART module 1 accorring to [UART_cfg.h](#) and [UART_cfg.c](#)
and then enable UART module 1

Definition at line 508 of file UART.c.

Here is the caller graph for this function:



5.60.3.31 UART1_Receive9BitData() `ERROR_STATUS_t UART1_Receive9BitData (`
`u16 * data)`

Receive a string of 16 bits using UART module 1 (9 bits data)

For Example: `UART1_Receive9BitString(string);` will receive a string of 16 bits
from UART module 1 and store it in variable `<string>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 603 of file UART.c.

5.60.3.32 UART1_Receive9BitData_NoBlock() `ERROR_STATUS_t UART1_Receive9BitData_NoBlock (`
`u16 *const data)`

Definition at line 620 of file UART.c.

5.60.3.33 UART1_ReceiveByte() `ERROR_STATUS_t UART1_ReceiveByte (`
`u8 * data)`

Receive a byte using UART module 1.

For Example: `UART1_ReceiveByte(&data);` will receive a byte from UART module 1
and store it in variable `<data>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 595 of file UART.c.

Here is the caller graph for this function:



5.60.3.34 UART1_ReceiveByte_NoBlock() `ERROR_STATUS_t UART1_ReceiveByte_NoBlock (u8 * data)`

Receive a byte using UART module 1 Asynchronously.

For Example: `UART1_ReceiveByte(&data);` will receive a byte from UART module 1

and store it in variable `<data>` without checking receive complete flag and without blocking the calling thread

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Definition at line 612 of file UART.c.

5.60.3.35 UART1_RX_InterruptDisable() `void UART1_RX_InterruptDisable (void)`

Disable the receive interrupt of UART module 1.

For Example: `UART1_RX_InterruptDisable();` will disable the receive interrupt

of UART module 1

Definition at line 718 of file UART.c.

5.60.3.36 UART1_RX_InterruptEnable() `void UART1_RX_InterruptEnable (void(*) (void) ptrCallback)`

Enable the receive interrupt of UART module 1 with passing the function pointer to the interrupt service routine.

Parameters

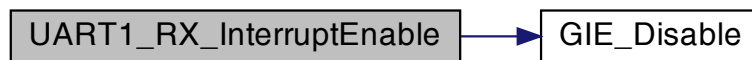
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The interrupt is triggered when a byte is received

Definition at line 659 of file UART.c.

Here is the call graph for this function:



5.60.3.37 UART1_Send9BitData() `void UART1_Send9BitData (`
`const u16 data)`

Send 9 bits using UART module 1.

Parameters

in	data	9 bits data to be sent. Should be between 0 and 0x1FF
----	------	---

Note

Size of data should be 2 Bytes to be able to send 9 bits data

For Example: `UART1_Send9Bits(0x123);` will send 9 bits 0x123 to UART module 1

Definition at line 569 of file UART.c.

5.60.3.38 UART1_Send9BitData_NoBlock() `void UART1_Send9BitData_NoBlock (`
`const u16 data)`

Send 9 bits using UART module 1 Asynchronously.

Parameters

in	data	9 bits data to be sent. Should be between 0 and 0x1FF
----	------	---

Note

Size of data should be 2 Bytes to be able to send 9 bits data

For Example: `UART1_Send9Bits_NoBlock(0x123);` will send 9 bits 0x123 to UART module 1

without blocking the calling thread

Definition at line 586 of file UART.c.

5.60.3.39 UART1_SendByte() `void UART1_SendByte (`
`const u8 data)`

Send a byte using UART module 1.

For Example: `UART1_SendByte('y');` will send character 'y' to UART module 1

Definition at line 561 of file UART.c.

Here is the caller graph for this function:



5.60.3.40 UART1_SendByte_NoBlock() `void UART1_SendByte_NoBlock (`
`const u8 data)`

Send a byte using UART module 1 Asynchronously.

For Example: `UART1_SendByte_NoBlock('a');` will send character 'a' to UART module 1

without blocking the calling thread

Definition at line 578 of file UART.c.

5.60.3.41 UART1_TX_InterruptDisable() `void UART1_TX_InterruptDisable (`
`void)`

Disable the transmit interrupt of UART module 1.

For Example: `UART1_TX_InterruptDisable();` will disable the transmit interrupt

of UART module 1

Definition at line 726 of file UART.c.

5.60.3.42 UART1_TX_InterruptEnable() `void UART1_TX_InterruptEnable (`
`void(*) (void) ptrCallback)`

Enable the transmit interrupt of UART module 0 with passing the function pointer to the interrupt service routine.

Parameters

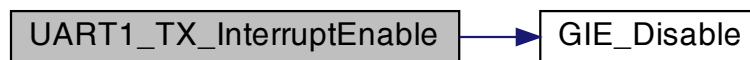
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The transmit interrupt will be triggered after transmission complete.

Definition at line 670 of file UART.c.

Here is the call graph for this function:



5.60.3.43 UART1_UDRE_InterruptDisable() `void UART1_UDRE_InterruptDisable (void)`

Disable the interrupt of Data Register Empty of UART module 1.

For Example: `UART1_UDRE_InterruptDisable();` will disable the interrupt of Data Register Empty of UART module 1

Definition at line 734 of file UART.c.

5.60.3.44 UART1_UDRE_InterruptEnable() `void UART1_UDRE_InterruptEnable (void(*) (void) ptrCallback)`

Enable the interrupt of Data Register Empty of UART module 1 with passing the function pointer to the interrupt service routine.

Parameters

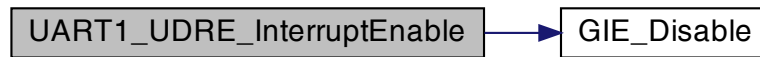
in	<i>ptrCallback</i>	Pointer to the interrupt service routine
----	--------------------	--

Note

The interrupt is triggered when the data register is empty

Definition at line 703 of file UART.c.

Here is the call graph for this function:

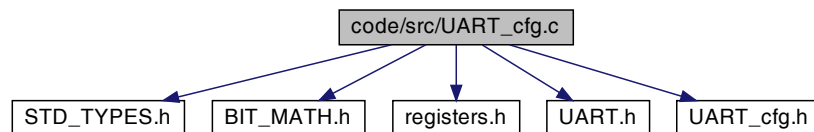


5.61 code/src/UART_cfg.c File Reference

Configuration source file for [UART.c](#).

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "registers.h"
#include "UART.h"
#include "UART_cfg.h"
```

Include dependency graph for UART_cfg.c:



Variables

- [UART_CFG_t UART0_Configs](#)
- [UART_CFG_t UART1_Configs](#)

5.61.1 Detailed Description

Configuration source file for [UART.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-03

Copyright

Copyright (c) 2022

5.61.2 Variable Documentation

5.61.2.1 UART0_Configs [UART_CFG_t](#) UART0_Configs

Initial value:

```
= {  
    .baud_rate   = 9600,  
    .data_bits   = UART_DATA_8_BITS,  
    .stop_bits   = UART_STOP_1_BIT,  
    .parity      = UART_PARITY_DISABLE,  
    .mode        = UART_MODE_ASYNCHRONOUS_NORMAL,  
    .mode_type   = UART_MODE_TX_RX,  
}
```

Note

Baud rate options:

- 2400UL --> 2400 bits per second
- 4800UL --> 4800 bits per second
- 9600UL --> 9600 bits per second
- 14400UL --> 14400 bits per second
- 19200UL --> 19200 bits per second
- 28800UL --> 28800 bits per second
- 38400UL --> 38400 bits per second
- 57600UL --> 57600 bits per second
- 76800UL --> 76800 bits per second
- 115200UL --> 115200 bits per second
- 230400UL --> 230400 bits per second
- 250000UL --> 250000 bits per second
- 500000UL --> 500000 bits per second
- 1000000UL --> 1000000 bits per second

Definition at line 34 of file UART_cfg.c.

5.61.2.2 UART1_Configs [UART_CFG_t](#) UART1_Configs

Initial value:

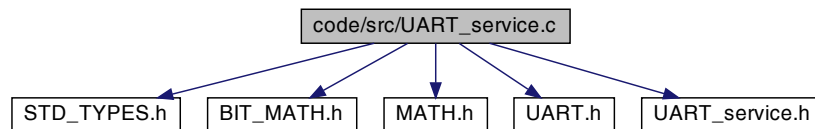
```
= {  
    .baud_rate   = 9600,  
    .data_bits   = UART_DATA_8_BITS,  
    .stop_bits   = UART_STOP_1_BIT,  
    .parity      = UART_PARITY_DISABLE,  
    .mode        = UART_MODE_ASYNCHRONOUS_NORMAL,  
    .mode_type   = UART_MODE_TX_RX,  
}
```

Definition at line 43 of file UART_cfg.c.

5.62 code/src/UART_service.c File Reference

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "MATH.h"
#include "UART.h"
#include "UART_service.h"
```

Include dependency graph for UART_service.c:



Functions

- void [UART0_SendString](#) (const [u8](#) *const string)
Send a string using UART module 0.
- void [UART1_SendString](#) (const [u8](#) *const string)
Send a string using UART module 1.
- void [UART0_Send9BitString](#) (const [u16](#) *const string)
Send a sequence of elements, each element has 9 bits, using UART module 0.
- void [UART1_Send9BitString](#) (const [u16](#) *const string)
Send a sequence of elements, each element has 9 bits, using UART module 1.
- void [UART0_SendString_Asynchronous](#) (const [u8](#) *const string)
- void [UART1_SendString_Asynchronous](#) (const [u8](#) *const string)
- void [UART0_Send9BitString_Asynchronous](#) (const [u16](#) *const string)
- void [UART1_Send9BitString_Asynchronous](#) (const [u16](#) *const string)
- void [UART0_SendString_Checksum](#) (const [u8](#) *const string)
- void [UART1_SendString_Checksum](#) (const [u8](#) *const string)
- void [UART0_SendInteger](#) ([s32](#) integer)
Write Integer to UART0 as a string.
- void [UART1_SendInteger](#) ([s32](#) integer)
Write Integer to UART1 as a string.
- void [UART0_SendFloat](#) (const float number, const [u8](#) precision)
Write Float to UART0 as a string.
- void [UART1_SendFloat](#) (const float number, const [u8](#) precision)
Write Float to UART1 as a string.
- [ERROR_STATUS_t](#) [UART0_ReceiveString](#) ([u8](#) *const string)
Receive a string of characters (unsigned 8 bits data) using UART module 0.
- [ERROR_STATUS_t](#) [UART1_ReceiveString](#) ([u8](#) *const string)
Receive a string of characters (unsigned 8 bits data) using UART module 1.
- [ERROR_STATUS_t](#) [UART0_Receive9BitString](#) ([u16](#) *const string)
Receive a string of elements, each element has 9 bits, using UART module 0.
- [ERROR_STATUS_t](#) [UART1_Receive9BitString](#) ([u16](#) *const string)
Receive a string of elements, each element has 9 bits, using UART module 1.
- [ERROR_STATUS_t](#) [UART0_ReceiveString_Checksum](#) ([u8](#) *const string)
- [ERROR_STATUS_t](#) [UART1_ReceiveString_Checksum](#) ([u8](#) *const string)

5.62.1 Function Documentation

5.62.1.1 **UART0_Receive9BitString()** `ERROR_STATUS_t UART0_Receive9BitString (u16 *const string)`

Receive a string of elements, each element has 9 bits, using UART module 0.

Parameters

in	<i>string</i>	Pointer to the first element of the sequence (pointer to 16 bits)
----	---------------	---

Note

Size of each element should be 2 Bytes to be able to receive 9 bits data

For Example: `UART0_Receive9BitDataSequence(string);` will receive elements of 9 bits each from UART module 0.

Warning

The sequence will be ended with a NULL pointer(0): The last element of the sequence will be a NULL pointer. So, consider the length of the sequence as `strlen(string) + 1`.

Definition at line 394 of file `UART_service.c`.

5.62.1.2 **UART0_ReceiveString()** `ERROR_STATUS_t UART0_ReceiveString (u8 *const string)`

Receive a string of characters (unsigned 8 bits data) using UART module 0.

For Example: `UART0_ReceiveString(string);` will receive a string of characters from UART module 0 and store it in variable `<string>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Warning

The string will be null terminated. So, the last character will be '\0'. So, the length of the string should be `strlen(string) + 1`.

Definition at line 386 of file `UART_service.c`.

Here is the caller graph for this function:



5.62.1.3 UART0_ReceiveString_Checksum()

```

ERROR_STATUS_t UART0_ReceiveString_Checksum (
    u8 *const string )
  
```

Definition at line 403 of file `UART_service.c`.

Here is the caller graph for this function:



5.62.1.4 UART0_Send9BitString()

```

void UART0_Send9BitString (
    const u16 *const string )
  
```

Send a sequence of elements, each element has 9 bits, using UART module 0.

Parameters

in	<i>string</i>	Pointer to the first element of the sequence
----	---------------	--

Note

Size of each element should be 2 Bytes to be able to send 9 bits data

For Example: `UART0_Send9BitDataSequence(string);` will send elements of 9 bits

each to UART module 0.

Note

The sequence should be ended with a NULL pointer(0): The last element of the sequence should be a NULL pointer.

But if you are sending an array, you must explicitly null terminate it by adding '\0' at the end of the array --> `string[lenght-1] = '\0'`

Definition at line 331 of file `UART_service.c`.

5.62.1.5 UART0_Send9BitString_Asynchronous() `void UART0_Send9BitString_Asynchronous (`
`const u16 *const string)`

Definition at line 349 of file `UART_service.c`.

5.62.1.6 UART0_SendFloat() `void UART0_SendFloat (`
`float number,`
`const u8 precision)`

Write Float to UART0 as a string.

Parameters

in	<i>number</i>	The number to be written
----	---------------	--------------------------

For Example:

- `UART0_WriteFloat(123.456, 3)` will write the string "123.456" to UART0
- `UART0_WriteFloat(123.456, 2)` will write the string "123.45" to UART0
- `UART0_WriteFloat(123.456, 5)` will write the string "123.45600" to UART0
- `UART0_WriteFloat(-123.456, 3)` will write the string "-123.456" to UART0
- `UART0_WriteFloat(-123.456, 2)` will write the string "-123.45" to UART0
- `UART0_WriteFloat(-123.456, 5)` will write the string "-123.45600" to UART0

Definition at line 377 of file `UART_service.c`.

5.62.1.7 UART0_SendInteger() void UART0_SendInteger (
s32 integer)

Write Integer to UART0 as a string.

Parameters

in	<i>integer</i>	The integer to be written
----	----------------	---------------------------

For Example:

- UART0_WriteInt(123) will write the string "123" to UART0
- UART0_WriteInt(-123) will write the string "-123" to UART0

Definition at line 368 of file UART_service.c.

Here is the caller graph for this function:



5.62.1.8 UART0_SendString() `void UART0_SendString (`
`const u8 *const string)`

Send a string using UART module 0.

For Example: `UART0_SendString("Hello World");` will send string "Hello World" to UART module 0

Definition at line 323 of file UART_service.c.

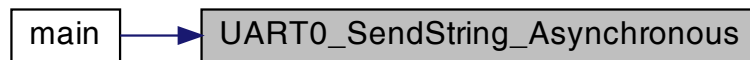
Here is the caller graph for this function:



5.62.1.9 UART0_SendString_Asynchronous() `void UART0_SendString_Asynchronous (const u8 *const string)`

Definition at line 340 of file UART_service.c.

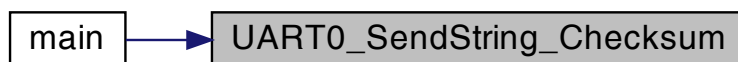
Here is the caller graph for this function:



5.62.1.10 UART0_SendString_Checksum() `void UART0_SendString_Checksum (const u8 *const string)`

Definition at line 359 of file UART_service.c.

Here is the caller graph for this function:



5.62.1.11 UART1_Receive9BitString() `ERROR_STATUS_t UART1_Receive9BitString (u16 *const string)`

Receive a string of elements, each element has 9 bits, using UART module 1.

Parameters

in	string	Pointer to the first element of the sequence (pointer to 16 bits)
----	--------	---

Note

Size of each element should be 2 Bytes to be able to receive 9 bits data

For Example: `UART1_Receive9BitDataSequence(string);` will receive elements of 9 bits each from UART module 1.

Warning

The sequence will be ended with a NULL pointer(0): The last element of the sequence will be a NULL pointer. So, consider the length of the sequence as `strlen(string) + 1`.

Definition at line 398 of file `UART_service.c`.

5.62.1.12 UART1_ReceiveString() `ERROR_STATUS_t UART1_ReceiveString (u8 *const string)`

Receive a string of characters (unsigned 8 bits data) using UART module 1.

For Example: `UART1_ReceiveString(string);` will receive a string of characters from UART module 1 and store it in variable `<string>`

Returns

The error code: `ERROR_NO` if no error, `ERROR_YES` if error

Warning

The string will be null terminated. So, the last character will be `'\0'`. So, the length of the string should be `strlen(string) + 1`.

Definition at line 390 of file `UART_service.c`.

5.62.1.13 UART1_ReceiveString_Checksum() `ERROR_STATUS_t UART1_ReceiveString_Checksum (u8 *const string)`

Definition at line 407 of file `UART_service.c`.

5.62.1.14 UART1_Send9BitString() `void UART1_Send9BitString (const u16 *const string)`

Send a sequence of elements, each element has 9 bits, using UART module 1.

Parameters

in	<i>string</i>	Pointer to the first element of the sequence
----	---------------	--

Note

Size of each element should be 2 Bytes to be able to send 9 bits data

For Example: `UART1_Send9BitDataSequence(string);` will send elements of 9 bits

each to UART module 1.

Note

The sequence should be ended with a NULL pointer(0): The last element of the sequence should be a NULL pointer.

But if you are sending an array, you must explicitly null terminate it by adding '\0' at the end of the array --> `string[lenght-1] = '\0'`

Definition at line 335 of file `UART_service.c`.

5.62.1.15 UART1_Send9BitString_Asynchronous() `void UART1_Send9BitString_Asynchronous (const u16 *const string)`

Definition at line 354 of file `UART_service.c`.

5.62.1.16 UART1_SendFloat() `void UART1_SendFloat (float number, const u8 precision)`

Write Float to UART1 as a string.

Parameters

in	<i>number</i>	The number to be written
----	---------------	--------------------------

For Example:

- `UART1_WriteFloat(123.456, 3)` will write the string "123.456" to UART1
- `UART1_WriteFloat(123.456, 2)` will write the string "123.45" to UART1
- `UART1_WriteFloat(123.456, 5)` will write the string "123.45600" to UART1
- `UART1_WriteFloat(-123.456, 3)` will write the string "-123.456" to UART1
- `UART1_WriteFloat(-123.456, 2)` will write the string "-123.45" to UART1
- `UART1_WriteFloat(-123.456, 5)` will write the string "-123.45600" to UART1

Definition at line 381 of file `UART_service.c`.

Here is the caller graph for this function:



5.62.1.17 UART1_SendInteger() `void UART1_SendInteger (`
`s32 integer)`

Write Integer to UART1 as a string.

Parameters

in	<i>integer</i>	The integer to be written
----	----------------	---------------------------

For Example:

- UART1_WriteInt(123) will write the string "123" to UART1
- UART1_WriteInt(-123) will write the string "-123" to UART1

Definition at line 372 of file UART_service.c.

Here is the caller graph for this function:



5.62.1.18 UART1_SendString() `void UART1_SendString (`
`const u8 *const string)`

Send a string using UART module 1.

For Example: `UART1_SendString("Hello World");` will send string "Hello World"

to UART module 1.

Note

String must be null terminated: "Hello World\0". So, the last character must be '\0'.

Sending "Hello World" will send "Hello World\0" because it is null terminated implicitly. But if you are sending an array of characters, you must explicitly null terminate it by adding '\0' at the end of the array. --> `string[lenght-1] = '\0'`

Definition at line 327 of file `UART_service.c`.

Here is the caller graph for this function:



5.62.1.19 UART1_SendString_Asynchronous() `void UART1_SendString_Asynchronous (`
`const u8 *const string)`

Definition at line 344 of file `UART_service.c`.

5.62.1.20 UART1_SendString_Checksum() `void UART1_SendString_Checksum (`
`const u8 *const string)`

Definition at line 363 of file `UART_service.c`.

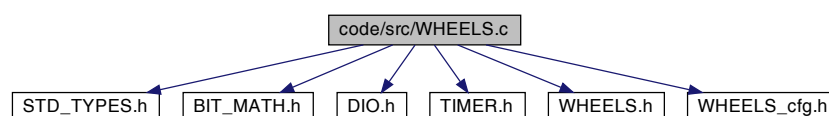
5.63 code/src/WHEELS.c File Reference

This is the source file for the WHEELS module based on the WHEELS module.

```

#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "DIO.h"
#include "TIMER.h"
#include "WHEELS.h"
#include "WHEELS_cfg.h"
  
```

Include dependency graph for `WHEELS.c`:



Functions

- void `WHEELS_Init` (void)
Initialize the wheels module.
- void `WHEELS_GoForward` (void)
Turn the wheels to go forward.
- void `WHEELS_GoBackward` (void)
Turn the wheels to go backward.
- void `WHEELS_TurnLeft` (const `WHEELS_TURN_t` smoothOrSharp)
Turn the wheels to turn left.
- void `WHEELS_TurnRight` (const `WHEELS_TURN_t` smoothOrSharp)
Turn the wheels to turn right.
- void `WHEELS_Brake` (void)
Brake the wheels.
- void `WHEELS_Coast` (void)
Disables wheels and allows for free rolling.
- void `WHEELS_SetSpeed` (const `u8` Speed)
Set the speed of the wheels.
- void `WHEELS_SetWheelsPosition` (const `WHEELS_POSITION_t` wheelsPositions)
Set the position of the wheels: on the front or on the back.
- `WHEELS_POSITION_t` `WHEELS_GetWheelsPosition` (void)
Get the current position of the wheels.

5.63.1 Detailed Description

This is the source file for the WHEELS module based on the WHEELS module.

Author

Mahmoud Karam (ma.karam272@gmail.com)

This is the source file for the WHEELS module based on the WHEELS module. The WHEELS module is a module that controls the wheels of the robot. WHEELS is a dual H-bridge module that controls the motors of the robot. There are 6 control pins: IN1A, IN2A, IN1B, IN2B, ENA, and ENB
There are 2 feedback pins: CTA and CTB.

- IN1A and IN2A control the speed of motor A. (PWM)
- IN1B and IN2B control the speed of motor B. (PWM)
- ENA enables the driver's output for motor A. (Digital)
- ENB enables the driver's output for motor B. (Digital)
 - CTA reports the current draw of motor A back to the microcontroller. (Analog)
- CTB reports the current draw of motor B back to the microcontroller. (Analog)

Note

The PWM frequency is recommended to be 16 kHz for normal motors, 80 kHz for Coreless motors.
CTA and CTB are Optional.

Example:

When IN1A is driven LOW and IN2A is provided a PWM signal, the motor will go in reverse. When IN1A is provided a PWM signal and IN2A is driven LOW, the motor will go forward.

Warning

The driver requires a maximum duty cycle on the PWM input of no more than 98%.
Any higher might damage the driver / result in instability.

Library usage:**Option 1:**

```
WHEELS.Init(); This will use the default pins
```

Option 2:

```
WHEELS.init(IN1A, IN1B, IN2A, IN2B, ENA, ENB, CTA, CTB); This will define which pins are used on the microcontroller.
```

Once defined the motor can be controlled by using:

```
WHEELS.GoForward(); Moves both motors forward
WHEELS.GoBackward(); Moves both motors reverse
```

WHEELS_TurnLeft(SMOOTH_TURN); Moves the left and right motors forward, but the left motor is slower than the right motor WHEELS_TurnRight(SMOOTH_TURN); Moves the left and right motors forward, but the right motor is slower than the left motor WHEELS_TurnLeft(SHARP_TURN); Moves the left motor backward and right motors forward WHEELS_TurnRight(SHARP_TURN); Moves the right motor backward and left motors forward WHEELS.Brake(); UNTESTED - USE WITH CAUTION WHEELS.Coast(); Disables motor output WHEELS_SetSpeed(x); Sets the speed of both motors to x WHEELS_SetWheelsPosition(WHEELS_ON_FRONT); Sets the wheels position on the front WHEELS_SetWheelsPosition(WHEELS_ON_BACK); Sets the wheels position on the back [WHEELS_GetCurrentConsumption\(\)](#); Returns the current consumption of the wheels

Version

1.0.0

Date

2022-02-09

Copyright

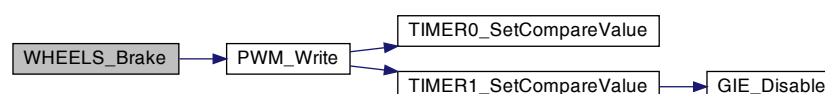
Copyright (c) 2022

5.63.2 Function Documentation**5.63.2.1 WHEELS_Brake()** void WHEELS_Brake (void)

Brake the wheels.

Definition at line 153 of file WHEELS.c.

Here is the call graph for this function:

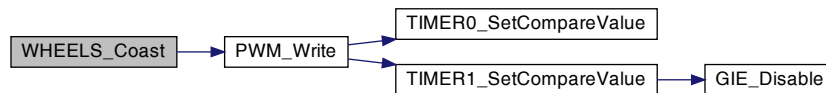


5.63.2.2 WHEELS_Coast() `void WHEELS_Coast (`
`void)`

Disables wheels and allows for free rolling.

Definition at line 164 of file WHEELS.c.

Here is the call graph for this function:



5.63.2.3 WHEELS_GetWheelsPosition() `WHEELS_POSITION_t WHEELS_GetWheelsPosition (`
`void)`

Get the current position of the wheels.

Returns

f32 The current consumption of the wheels in A.

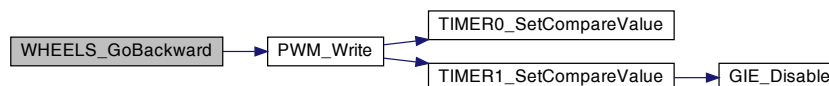
Definition at line 202 of file WHEELS.c.

5.63.2.4 WHEELS_GoBackward() `void WHEELS_GoBackward (`
`void)`

Turn the wheels to go backward.

Definition at line 91 of file WHEELS.c.

Here is the call graph for this function:



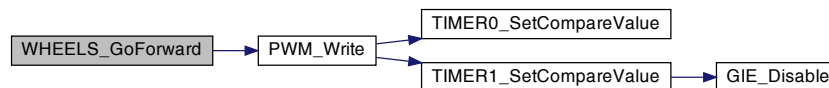
5.63.2.5 WHEELS_GoForward()

```
void WHEELS_GoForward (
    void )
```

Turn the wheels to go forward.

Definition at line 79 of file WHEELS.c.

Here is the call graph for this function:



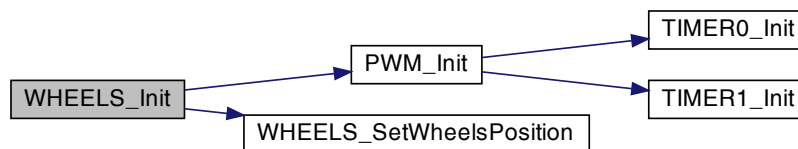
5.63.2.6 WHEELS_Init()

```
void WHEELS_Init (
    void )
```

Initialize the wheels module.

Definition at line 68 of file WHEELS.c.

Here is the call graph for this function:



Here is the caller graph for this function:



5.63.2.7 WHEELS_SetSpeed()

```
void WHEELS_SetSpeed (
    u8 Speed )
```

Set the speed of the wheels.

Parameters

in	<i>Speed</i>	The speed of the wheels.
----	--------------	--------------------------

Definition at line 176 of file WHEELS.c.

5.63.2.8 WHEELS_SetWheelsPosition() `void WHEELS_SetWheelsPosition (
WHEELS_POSITION_t wheelsPositions)`

Set the position of the wheels: on the front or on the back.

Parameters

in	<i>wheelsPositions</i>	The poition of the wheels: WHEELS_ON_FRONT or WHEELS_ON_BACK.
----	------------------------	---

Definition at line 181 of file WHEELS.c.

Here is the caller graph for this function:

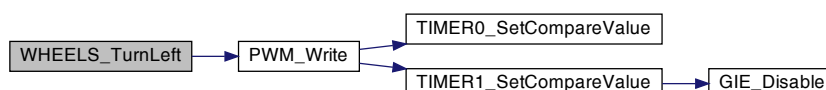


5.63.2.9 WHEELS_TurnLeft() `void WHEELS_TurnLeft (
const WHEELS_TURN_t smoothOrSharp)`

Turn the wheels to turn left.

Definition at line 103 of file WHEELS.c.

Here is the call graph for this function:

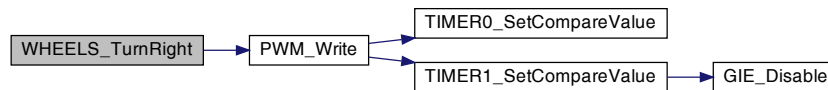


5.63.2.10 WHEELS_TurnRight() `void WHEELS_TurnRight (`
`const WHEELS_TURN_t smoothOrSharp)`

Turn the wheels to turn right.

Definition at line 128 of file WHEELS.c.

Here is the call graph for this function:

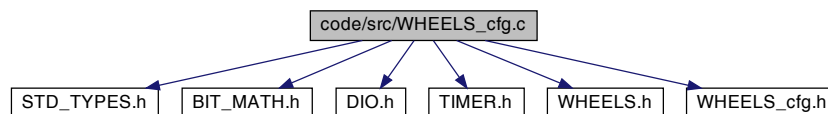


5.64 code/src/WHEELS_cfg.c File Reference

Configuration source file for [WHEELS.c](#).

```
#include "STD_TYPES.h"
#include "BIT_MATH.h"
#include "DIO.h"
#include "TIMER.h"
#include "WHEELS.h"
#include "WHEELS_cfg.h"
```

Include dependency graph for WHEELS_cfg.c:



Variables

- [WHEELS_CONFIG_t WHEELS_Config](#)

5.64.1 Detailed Description

Configuration source file for [WHEELS.c](#).

Author

Mahmoud Karam (ma.karam272@gmail.com)

Version

1.0.0

Date

2022-03-19

Copyright

Copyright (c) 2022

5.64.2 Variable Documentation

5.64.2.1 WHEELS_Config [WHEELS_CONFIG_t](#) WHEELS_Config

Initial value:

```
= {  
    .ENA_pin    = PIN_3,  
    .ENA_port   = PORT_G,  
    .ENB_pin    = PIN_4,  
    .ENB_port   = PORT_G,  
  
    .IN1A_channel = PWM_0,  
    .IN2A_channel = PWM_1,  
    .IN1B_channel = PWM_2,  
    .IN2B_channel = PWM_3,  
    .CTA_pin     = PIN_6,  
    .CTA_port    = PORT_A,  
    .CTB_pin     = PIN_7,  
    .CTB_port    = PORT_A,  
    .currentSensitivity = 155,  
    .SpeedPercentage    = 60,  
    .WHEELS_Position    = WHEELS_ON_FRONT,  
}
```

Definition at line 16 of file WHEELS_cfg.c.

6 Example Documentation

6.1 LED_Toggle

Toggle state of a specific LED

Parameters

<code>in</code>	<code>led</code>	the LED to be turned on/off (LED_0);
-----------------	------------------	--------------------------------------

Index

- __vector_1
 - EXTI.c, [211](#)
- __vector_10
 - TIMER.c, [241](#)
- __vector_11
 - TIMER.c, [241](#)
- __vector_12
 - TIMER.c, [241](#)
- __vector_13
 - TIMER.c, [241](#)
- __vector_14
 - TIMER.c, [241](#)
- __vector_15
 - TIMER.c, [241](#)
- __vector_16
 - TIMER.c, [241](#)
- __vector_17
 - SPI.c, [234](#)
- __vector_18
 - UART.c, [256](#)
- __vector_19
 - UART.c, [256](#)
- __vector_2
 - EXTI.c, [211](#)
- __vector_20
 - UART.c, [257](#)
- __vector_24
 - TIMER.c, [241](#)
- __vector_25
 - TIMER.c, [242](#)
- __vector_26
 - TIMER.c, [242](#)
- __vector_27
 - TIMER.c, [242](#)
- __vector_28
 - TIMER.c, [242](#)
- __vector_29
 - TIMER.c, [242](#)
- __vector_3
 - EXTI.c, [211](#)
- __vector_30
 - UART.c, [257](#)
- __vector_31
 - UART.c, [257](#)
- __vector_32
 - UART.c, [257](#)
- __vector_4
 - EXTI.c, [211](#)
- __vector_5
 - EXTI.c, [211](#)
- __vector_6
 - EXTI.c, [211](#)
- __vector_7
 - EXTI.c, [212](#)
- __vector_8
 - EXTI.c, [212](#)
- __vector_9
 - TIMER.c, [242](#)
- ABS
 - MATH.h, [89](#)
- ACOS
 - MATH.h, [90](#)
- ACTIVATE
 - NRF24_reg.h, [106](#)
- ACTIVATION_STATUS_t
 - STD_TYPES.h, [130](#)
- ACTIVE_HIGH
 - STD_TYPES.h, [131](#)
- ACTIVE_LOW
 - STD_TYPES.h, [131](#)
- ActiveHighOrLow
 - BUTTON_CONFIGS_t, [6](#)
 - IR_CONFIG_t, [8](#)
- ADC_u8ADCH_REG
 - registers.h, [117](#)
- ADC_u8ADCL_REG
 - registers.h, [117](#)
- ADC_u8ADCSRA_REG
 - registers.h, [117](#)
- ADC_u8ADCSRB_REG
 - registers.h, [117](#)
- ADC_u8ADMUX_REG
 - registers.h, [117](#)
- addressWidth
 - NRF24_t, [13](#)
- app.h
 - AUTONOMOUS_MODE, [29](#)
 - autonomousMode, [31](#)
 - avoidObstacle, [31](#)
 - checkAutoOrManualNavigation, [31](#)
 - CLIENT_PICKUP_NO, [28](#)
 - CLIENT_PICKUP_t, [28](#)
 - CLIENT_PICKUP_YES, [28](#)
 - EAST, [30](#)
 - getOrderInfo, [31](#)
 - gotoNextJunction, [31](#)
 - handleOrder, [31](#)
 - MANUAL_MODE, [29](#)
 - manualNavigation, [31](#)
 - MASTER_PickItem, [32](#)
 - modifyOrientation, [32](#)
 - monitorBattery, [32](#)
 - navigate, [32](#)
 - navigateColumn, [32](#)
 - navigateRow, [32](#)
 - NAVIGATION_MODE_t, [28](#)
 - NO_ORDER_RECIEVED, [29](#)
 - NORTH, [30](#)
 - onJunction, [32](#)

- onLeftSharp, 32
- onLeftSmooth, 32
- onObstacle, 33
- onRightSharp, 33
- onRightSmooth, 33
- onTrack, 33
- ORDER_AVAILABILITY_t, 29
- ORDER_DELIVERED, 30
- ORDER_GOING_BACK, 30
- ORDER_OUT_FOR_DELIVER, 30
- ORDER_PROCESSING, 30
- ORDER_RECIEVED, 29
- ORDER_RETURNED, 30
- ORDER_TRACKING_t, 29
- ORIENTATION_t, 30
- readSerial, 33
- ROBOT_AVAILABLE, 31
- ROBOT_CHARGING, 31
- ROBOT_OBSTACLE_AVOIDANCE, 31
- ROBOT_PROCESSING_OEDER, 31
- ROBOT_STATE_t, 30
- sendOrderTrackInfo, 33
- sendPosition, 33
- sendRobotState, 33
- SOUTH, 30
- waitForClientPickup, 33
- WEST, 30
- app_cfg.h
 - CHARGER_COLUMN, 35
 - CHARGER_ROW, 36
 - CLIENT_COLUMN, 35
 - CLIENT_ROW, 36
 - COLUMN_t, 35
 - DEFAULT_COLUMN, 35
 - DEFAULT_ROW, 36
 - ITEM_PICKUP_BUTTON, 34
 - NAVIGATION_MODE_BUTTON, 34
 - NEON_LED, 35
 - NUM_OF_COLUMNS, 35
 - NUM_OF_ROWS, 35
 - OBSTACLE_DISTANCE_THRESHOLD, 35
 - ROW_t, 36
- appTest.c
 - main, 197
- ARC0
 - NRF24_reg.h, 115
- ARC1
 - NRF24_reg.h, 115
- ARC2
 - NRF24_reg.h, 115
- ARC3
 - NRF24_reg.h, 115
- ARC_CNT0
 - NRF24_reg.h, 113
- ARC_CNT1
 - NRF24_reg.h, 113
- ARC_CNT2
 - NRF24_reg.h, 113
- ARC_CNT3
 - NRF24_reg.h, 113
- ARD0
 - NRF24_reg.h, 115
- ARD1
 - NRF24_reg.h, 115
- ARD2
 - NRF24_reg.h, 115
- ARD3
 - NRF24_reg.h, 115
- AS0
 - TIMER_reg.h, 152
- ASIN
 - MATH.h, 90
- ASSR
 - TIMER_reg.h, 146
- ATAN
 - MATH.h, 91
- AUTONOMOUS_MODE
 - app.h, 29
- autonomousMode
 - app.h, 31
- avoidObstacle
 - app.h, 31
- AW0
 - NRF24_reg.h, 115
- AW1
 - NRF24_reg.h, 115
- BATTERY.h
 - BATTERY_GetState, 37
 - BATTERY_Init, 37
- BATTERY_CAPACITY
 - BATTERY_cfg.h, 38
- BATTERY_cfg.c
 - BatteryConfigs, 199
- BATTERY_cfg.h
 - BATTERY_CAPACITY, 38
 - BATTERY_EMPTY_VOLTAGE, 38
 - BATTERY_FULL_VOLTAGE, 39
 - BATTERY_VOLTAGE, 39
 - BATTERY_VOLTAGE_DIVIDER_RATIO, 39
 - BatteryConfigs, 39
- BATTERY_EMPTY_VOLTAGE
 - BATTERY_cfg.h, 38
- BATTERY_FULL_VOLTAGE
 - BATTERY_cfg.h, 39
- BATTERY_GetState
 - BATTERY.h, 37
- BATTERY_Init
 - BATTERY.h, 37
- BATTERY_VOLTAGE
 - BATTERY_cfg.h, 39
- BATTERY_VOLTAGE_DIVIDER_RATIO
 - BATTERY_cfg.h, 39
- BatteryConfigs
 - BATTERY_cfg.c, 199
 - BATTERY_cfg.h, 39
- BatteryConfigs_t, 5

- maxVoltage, 5
- minVoltage, 5
- pin, 6
- port, 6
- baud_rate
 - UART_CFG_t, 23
- BIT_IS_CLEAR
 - BIT_MATH.h, 40
- BIT_IS_SET
 - BIT_MATH.h, 41
- BIT_MATH.h
 - BIT_IS_CLEAR, 40
 - BIT_IS_SET, 41
 - CLR_BIT, 41
 - CONCAT_2BITS, 42
 - CONCAT_3BITS, 42
 - CONCAT_4BITS, 42
 - CONCAT_5BITS, 42
 - CONCAT_6BITS, 42
 - CONCAT_7BITS, 42
 - CONCAT_8BITS, 43
 - GET_BIT, 43
 - SET_BIT, 43
 - TOG_BIT, 44
- BOOL_t
 - STD_TYPES.h, 131
- button
 - BUTTON_CONFIGS_t, 6
- BUTTON.c
 - BUTTON_GetStatus, 200
 - BUTTON_Init, 201
- BUTTON.h
 - BUTTON_0, 45
 - BUTTON_1, 45
 - BUTTON_2, 45
 - BUTTON_3, 45
 - BUTTON_4, 45
 - BUTTON_5, 45
 - BUTTON_6, 45
 - BUTTON_7, 45
 - BUTTON_GetStatus, 46
 - BUTTON_Init, 46
 - BUTTON_t, 45
- BUTTON_0
 - BUTTON.h, 45
- BUTTON_1
 - BUTTON.h, 45
- BUTTON_2
 - BUTTON.h, 45
- BUTTON_3
 - BUTTON.h, 45
- BUTTON_4
 - BUTTON.h, 45
- BUTTON_5
 - BUTTON.h, 45
- BUTTON_6
 - BUTTON.h, 45
- BUTTON_7
 - BUTTON.h, 45
- BUTTON.h, 45
- BUTTON_cfg.c
 - buttonsConfigs, 202
 - countButtonsConfigured, 202
- BUTTON_cfg.h
 - buttonsConfigs, 48
 - countButtonsConfigured, 48
 - DEBOUNCE_OFF, 48
 - DEBOUNCE_ON, 48
 - DEBOUNCE_t, 47
- BUTTON_CONFIGS_t, 6
 - ActiveHighOrLow, 6
 - button, 6
 - debounceStatus, 7
 - pin, 7
 - port, 7
- BUTTON_GetStatus
 - BUTTON.c, 200
 - BUTTON.h, 46
- BUTTON_Init
 - BUTTON.c, 201
 - BUTTON.h, 46
- BUTTON_t
 - BUTTON.h, 45
- buttonsConfigs
 - BUTTON_cfg.c, 202
 - BUTTON_cfg.h, 48
- C_BIT
 - SREG.h, 128
- CD
 - NRF24_reg.h, 107
- ce
 - PINS_t, 18
- CEIL
 - MATH.h, 91
- channel
 - NRF24_t, 13
- CHARGER_COLUMN
 - app_cfg.h, 35
- CHARGER_ROW
 - app_cfg.h, 36
- checkAutoOrManualNavigation
 - app.h, 31
- CLAMP
 - MATH.h, 91
- CLEAR_OC
 - TIMER.h, 135
- CLIENT_COLUMN
 - app_cfg.h, 35
- CLIENT_PICKUP_NO
 - app.h, 28
- CLIENT_PICKUP_t
 - app.h, 28
- CLIENT_PICKUP_YES
 - app.h, 28
- CLIENT_ROW
 - app_cfg.h, 36
- clock_polarity

- UART_CFG_t, [23](#)
- clockDivider
 - SPI_CONFIG_t, [19](#)
- clockMode
 - SPI_CONFIG_t, [20](#)
- CLR_BIT
 - BIT_MATH.h, [41](#)
- code/include/app.h, [26](#)
- code/include/app_cfg.h, [34](#)
- code/include/BATTERY.h, [36](#)
- code/include/BATTERY_cfg.h, [37](#)
- code/include/BIT_MATH.h, [39](#)
- code/include/BUTTON.h, [44](#)
- code/include/BUTTON_cfg.h, [47](#)
- code/include/DIO.h, [48](#)
- code/include/DIO_cfg.h, [55](#)
- code/include/DIO_reg.h, [56](#)
- code/include/EXTI.h, [60](#)
- code/include/EXTI_cfg.h, [64](#)
- code/include/EXTI_reg.h, [65](#)
- code/include/GIE.h, [69](#)
- code/include/IR.h, [71](#)
- code/include/IR_cfg.h, [75](#)
- code/include/LED.h, [77](#)
- code/include/LED_cfg.h, [80](#)
- code/include/LIFTER.h, [82](#)
- code/include/LIFTER_cfg.h, [85](#)
- code/include/MATH.h, [88](#)
- code/include/NRF24.h, [100](#)
- code/include/NRF24_cfg.h, [103](#)
- code/include/NRF24_reg.h, [104](#)
- code/include/registers.h, [116](#)
- code/include/SPI.h, [118](#)
- code/include/SPI_cfg.h, [122](#)
- code/include/SPI_reg.h, [125](#)
- code/include/SREG.h, [127](#)
- code/include/STD_TYPES.h, [128](#)
- code/include/TIMER.h, [131](#)
- code/include/TIMER_cfg.h, [144](#)
- code/include/TIMER_reg.h, [144](#)
- code/include/UART.h, [156](#)
- code/include/UART_cfg.h, [172](#)
- code/include/UART_reg.h, [176](#)
- code/include/UART_service.h, [179](#)
- code/include/WHEELS.h, [189](#)
- code/include/WHEELS_cfg.h, [195](#)
- code/src/appTest.c, [196](#)
- code/src/BATTERY.c, [198](#)
- code/src/BATTERY_cfg.c, [198](#)
- code/src/BUTTON.c, [200](#)
- code/src/BUTTON_cfg.c, [202](#)
- code/src/DIO.c, [203](#)
- code/src/DIO_cfg.c, [208](#)
- code/src/EXTI.c, [209](#)
- code/src/GIE.c, [214](#)
- code/src/IR.c, [216](#)
- code/src/IR_cfg.c, [219](#)
- code/src/LED.c, [220](#)
- code/src/LED_cfg.c, [223](#)
- code/src/LIFTER.c, [224](#)
- code/src/LIFTER_cfg.c, [228](#)
- code/src/NRF24.c, [230](#)
- code/src/NRF24_cfg.c, [232](#)
- code/src/SPI.c, [233](#)
- code/src/SPI_cfg.c, [238](#)
- code/src/TIMER.c, [239](#)
- code/src/UART.c, [254](#)
- code/src/UART_cfg.c, [271](#)
- code/src/UART_service.c, [273](#)
- code/src/WHEELS.c, [283](#)
- code/src/WHEELS_cfg.c, [289](#)
- column
 - ORDER_INFO_t, [16](#)
- COLUMN_t
 - app_cfg.h, [35](#)
- COM00
 - TIMER_reg.h, [151](#)
- COM01
 - TIMER_reg.h, [151](#)
- COM1A0
 - TIMER_reg.h, [152](#)
- COM1A1
 - TIMER_reg.h, [152](#)
- COM1B0
 - TIMER_reg.h, [152](#)
- COM1B1
 - TIMER_reg.h, [152](#)
- COM1C0
 - TIMER_reg.h, [152](#)
- COM1C1
 - TIMER_reg.h, [152](#)
- COM20
 - TIMER_reg.h, [154](#)
- COM21
 - TIMER_reg.h, [154](#)
- COM3A0
 - TIMER_reg.h, [154](#)
- COM3A1
 - TIMER_reg.h, [154](#)
- COM3B0
 - TIMER_reg.h, [154](#)
- COM3B1
 - TIMER_reg.h, [154](#)
- COM3C0
 - TIMER_reg.h, [154](#)
- COM3C1
 - TIMER_reg.h, [154](#)
- CONCAT_2BITS
 - BIT_MATH.h, [42](#)
- CONCAT_3BITS
 - BIT_MATH.h, [42](#)
- CONCAT_4BITS
 - BIT_MATH.h, [42](#)
- CONCAT_5BITS
 - BIT_MATH.h, [42](#)
- CONCAT_6BITS

- BIT_MATH.h, [42](#)
- CONCAT_7BITS
 - BIT_MATH.h, [42](#)
- CONCAT_8BITS
 - BIT_MATH.h, [43](#)
- CONFIG
 - NRF24_reg.h, [107](#)
- connections
 - SPI_CONFIG_t, [20](#)
- CONNECTIONS_t, [7](#)
 - pin, [7](#)
 - port, [7](#)
- CONT_WAVE
 - NRF24_reg.h, [112](#)
- COS
 - MATH.h, [92](#)
- countButtonsConfigured
 - BUTTON_cfg.c, [202](#)
 - BUTTON_cfg.h, [48](#)
- countIR SensorsConfigured
 - IR_cfg.c, [219](#)
 - IR_cfg.h, [76](#)
- countLedsConfigured
 - LED_cfg.c, [224](#)
 - LED_cfg.h, [81](#)
- countPinsConfigured
 - DIO_cfg.c, [209](#)
 - DIO_cfg.h, [56](#)
- CPHA
 - SPI_reg.h, [126](#)
- CPOL
 - SPI_reg.h, [126](#)
- CRCO
 - NRF24_reg.h, [114](#)
- CS00
 - TIMER_reg.h, [151](#)
- CS01
 - TIMER_reg.h, [151](#)
- CS02
 - TIMER_reg.h, [151](#)
- CS10
 - TIMER_reg.h, [152](#)
- CS11
 - TIMER_reg.h, [152](#)
- CS12
 - TIMER_reg.h, [152](#)
- CS20
 - TIMER_reg.h, [154](#)
- CS21
 - TIMER_reg.h, [154](#)
- CS22
 - TIMER_reg.h, [154](#)
- CS30
 - TIMER_reg.h, [154](#)
- CS31
 - TIMER_reg.h, [155](#)
- CS32
 - TIMER_reg.h, [155](#)
- csn
 - PINS_t, [18](#)
- CTA_pin
 - WHEELS_CONFIG_t, [24](#)
- CTA_port
 - WHEELS_CONFIG_t, [25](#)
- CTB_pin
 - WHEELS_CONFIG_t, [25](#)
- CTB_port
 - WHEELS_CONFIG_t, [25](#)
- currentSensitivity
 - WHEELS_CONFIG_t, [25](#)
- data_bits
 - UART_CFG_t, [23](#)
- dataOrder
 - SPI_CONFIG_t, [20](#)
- DDRA
 - DIO_reg.h, [57](#)
- DDRB
 - DIO_reg.h, [57](#)
- DDRC
 - DIO_reg.h, [57](#)
- DDRD
 - DIO_reg.h, [57](#)
- DDRE
 - DIO_reg.h, [57](#)
- DDRF
 - DIO_reg.h, [58](#)
- DDRG
 - DIO_reg.h, [58](#)
- DEBOUNCE_OFF
 - BUTTON_cfg.h, [48](#)
- DEBOUNCE_ON
 - BUTTON_cfg.h, [48](#)
- DEBOUNCE_t
 - BUTTON_cfg.h, [47](#)
- debounceStatus
 - BUTTON_CONFIGS_t, [7](#)
- DEFAULT_COLUMN
 - app_cfg.h, [35](#)
- DEFAULT_ROW
 - app_cfg.h, [36](#)
- DEG_TO_RAD
 - MATH.h, [92](#)
- deliveryState
 - ORDER_INFO_t, [16](#)
- DIO.c
 - DIO_Init, [205](#)
 - DIO_InitPin, [205](#)
 - DIO_ReadPin, [206](#)
 - DIO_ReadPort, [206](#)
 - DIO_TogglePin, [207](#)
 - DIO_WritePin, [207](#)
 - DIO_WritePort, [207](#)
 - INIT_PIN, [204](#)
 - WRITE_PIN, [205](#)
- DIO.h
 - DIO_Init, [50](#)

- DIO_InitPin, [51](#)
- DIO_ReadPin, [51](#)
- DIO_ReadPort, [52](#)
- DIO_WritePin, [53](#)
- DIO_WritePort, [54](#)
- DIR_t, [49](#)
- INPUT, [49](#)
- OUTPUT, [49](#)
- PIN_0, [49](#)
- PIN_1, [49](#)
- PIN_2, [49](#)
- PIN_3, [49](#)
- PIN_4, [49](#)
- PIN_5, [49](#)
- PIN_6, [49](#)
- PIN_7, [49](#)
- PIN_t, [49](#)
- PORT_A, [50](#)
- PORT_B, [50](#)
- PORT_C, [50](#)
- PORT_D, [50](#)
- PORT_E, [50](#)
- PORT_F, [50](#)
- PORT_G, [50](#)
- PORT_t, [50](#)
- PULLUP_FALSE, [50](#)
- PULLUP_t, [50](#)
- PULLUP_TRUE, [50](#)
- DIO_cfg.c
 - countPinsConfigured, [209](#)
 - pinConfigs, [209](#)
- DIO_cfg.h
 - countPinsConfigured, [56](#)
 - pinConfigs, [56](#)
- DIO_Init
 - DIO.c, [205](#)
 - DIO.h, [50](#)
- DIO_InitPin
 - DIO.c, [205](#)
 - DIO.h, [51](#)
- DIO_ReadPin
 - DIO.c, [206](#)
 - DIO.h, [51](#)
- DIO_ReadPort
 - DIO.c, [206](#)
 - DIO.h, [52](#)
- DIO_reg.h
 - DDRA, [57](#)
 - DDRB, [57](#)
 - DDRC, [57](#)
 - DDRD, [57](#)
 - DDRE, [57](#)
 - DDRF, [58](#)
 - DDRG, [58](#)
 - PINA, [58](#)
 - PINB, [58](#)
 - PINC, [58](#)
 - PIND, [58](#)
 - PINE, [59](#)
 - PINF, [59](#)
 - PING, [59](#)
 - PORTA, [59](#)
 - PORTB, [59](#)
 - PORTC, [59](#)
 - PORTD, [60](#)
 - PORTE, [60](#)
 - PORTF, [60](#)
 - PORTG, [60](#)
- DIO_TogglePin
 - DIO.c, [207](#)
- DIO_WritePin
 - DIO.c, [207](#)
 - DIO.h, [53](#)
- DIO_WritePort
 - DIO.c, [207](#)
 - DIO.h, [54](#)
- DIR_t
 - DIO.h, [49](#)
- direction
 - PinConfig_t, [17](#)
- directionPin
 - LIFTER_CONFIGS_t, [10](#)
- directionPinPort
 - LIFTER_CONFIGS_t, [10](#)
- DOR
 - UART_reg.h, [179](#)
- DORD
 - SPI_reg.h, [126](#)
- doubleSpeed
 - SPI_CONFIG_t, [20](#)
- DPL_P0
 - NRF24_reg.h, [114](#)
- DPL_P1
 - NRF24_reg.h, [114](#)
- DPL_P2
 - NRF24_reg.h, [114](#)
- DPL_P3
 - NRF24_reg.h, [114](#)
- DPL_P4
 - NRF24_reg.h, [114](#)
- DPL_P5
 - NRF24_reg.h, [114](#)
- DYNPD
 - NRF24_reg.h, [107](#)
- DYNPD0
 - NRF24_reg.h, [113](#)
- DYNPD1
 - NRF24_reg.h, [113](#)
- DYNPD2
 - NRF24_reg.h, [113](#)
- DYNPD3
 - NRF24_reg.h, [113](#)
- DYNPD4
 - NRF24_reg.h, [113](#)
- DYNPD5
 - NRF24_reg.h, [113](#)

EAST
 app.h, [30](#)
 EEPROM_u8EEARH_REG
 registers.h, [117](#)
 EEPROM_u8EEARL_REG
 registers.h, [117](#)
 EEPROM_u8EECR_REG
 registers.h, [117](#), [118](#)
 EEPROM_u8EEDR_REG
 registers.h, [118](#)
 EICRA
 EXTI_reg.h, [66](#)
 EICRB
 EXTI_reg.h, [67](#)
 EIFR
 EXTI_reg.h, [67](#)
 EIMSK
 EXTI_reg.h, [67](#)
 EN_AA
 NRF24_reg.h, [107](#)
 EN_ACK_PAY
 NRF24_reg.h, [114](#)
 EN_CRC
 NRF24_reg.h, [114](#)
 EN_DPL
 NRF24_reg.h, [114](#)
 EN_DYN_ACK
 NRF24_reg.h, [114](#)
 EN_RXADDR
 NRF24_reg.h, [107](#)
 ENA_pin
 WHEELS_CONFIG_t, [25](#)
 ENA_port
 WHEELS_CONFIG_t, [25](#)
 ENAA_P0
 NRF24_reg.h, [115](#)
 ENAA_P1
 NRF24_reg.h, [115](#)
 ENAA_P2
 NRF24_reg.h, [115](#)
 ENAA_P3
 NRF24_reg.h, [115](#)
 ENAA_P4
 NRF24_reg.h, [115](#)
 ENAA_P5
 NRF24_reg.h, [115](#)
 ENABLE_CONNECTION
 LIFTER_cfg.h, [87](#)
 enablePin
 LIFTER_CONFIGS_t, [10](#)
 enablePinPort
 LIFTER_CONFIGS_t, [10](#)
 ENB_pin
 WHEELS_CONFIG_t, [25](#)
 ENB_port
 WHEELS_CONFIG_t, [25](#)
 ERROR_NO
 STD_TYPES.h, [131](#)
 ERROR_STATUS_t
 STD_TYPES.h, [131](#)
 ERROR_TIMEOUT
 STD_TYPES.h, [131](#)
 ERROR_YES
 STD_TYPES.h, [131](#)
 ERX_P0
 NRF24_reg.h, [115](#)
 ERX_P1
 NRF24_reg.h, [115](#)
 ERX_P2
 NRF24_reg.h, [115](#)
 ERX_P3
 NRF24_reg.h, [115](#)
 ERX_P4
 NRF24_reg.h, [115](#)
 ERX_P5
 NRF24_reg.h, [115](#)
 EXP
 MATH.h, [93](#)
 EXTI.c
 __vector_1, [211](#)
 __vector_2, [211](#)
 __vector_3, [211](#)
 __vector_4, [211](#)
 __vector_5, [211](#)
 __vector_6, [211](#)
 __vector_7, [212](#)
 __vector_8, [212](#)
 EXTI_DisableExternalInterrupt, [212](#)
 EXTI_EnableExternalInterrupt, [212](#)
 EXTI_Init, [213](#)
 EXTI.h
 EXTI_0, [62](#)
 EXTI_1, [62](#)
 EXTI_2, [62](#)
 EXTI_3, [62](#)
 EXTI_4, [62](#)
 EXTI_5, [62](#)
 EXTI_6, [62](#)
 EXTI_7, [62](#)
 EXTI_DisableExternalInterrupt, [62](#)
 EXTI_EnableExternalInterrupt, [62](#)
 EXTI_Init, [63](#)
 EXTI_SENSITIVITY_t, [61](#)
 EXTI_t, [62](#)
 FALLING_EDGE, [61](#)
 LOGIC_CHANGE, [61](#)
 LOW_LEVEL_DETECT, [61](#)
 RISING_EDGE, [61](#)
 EXTI_0
 EXTI.h, [62](#)
 EXTI_1
 EXTI.h, [62](#)
 EXTI_2
 EXTI.h, [62](#)
 EXTI_3
 EXTI.h, [62](#)

EXTI_4
 EXTI.h, 62
 EXTI_5
 EXTI.h, 62
 EXTI_6
 EXTI.h, 62
 EXTI_7
 EXTI.h, 62
 EXTI_cfg.h
 NESTING, 65
 NESTING_DISABLED, 65
 NESTING_ENABLED, 65
 EXTI_DisableExternalInterrupt
 EXTI.c, 212
 EXTI.h, 62
 EXTI_EnableExternalInterrupt
 EXTI.c, 212
 EXTI.h, 62
 EXTI_Init
 EXTI.c, 213
 EXTI.h, 63
 EXTI_reg.h
 EICRA, 66
 EICRB, 67
 EIFR, 67
 EIMSK, 67
 INT0, 68
 INT1, 68
 INT2, 68
 INT3, 68
 INT4, 68
 INT5, 68
 INT6, 68
 INT7, 68
 INTF0, 69
 INTF1, 69
 INTF2, 69
 INTF3, 69
 INTF4, 69
 INTF5, 69
 INTF6, 69
 INTF7, 69
 ISC00, 68
 ISC01, 68
 ISC10, 68
 ISC11, 68
 ISC20, 68
 ISC21, 68
 ISC30, 68
 ISC31, 68
 ISC40, 68
 ISC41, 68
 ISC50, 68
 ISC51, 68
 ISC60, 68
 ISC61, 68
 ISC70, 68
 ISC71, 68
 IVCE, 67
 IVSEL, 67
 MCUCR, 67
 SE, 67
 SM0, 67
 SM1, 67
 SM2, 67
 SRE, 67
 SRW10, 67
 EXTI_SENSITIVITY_t
 EXTI.h, 61
 EXTI_t
 EXTI.h, 62

 f32
 STD_TYPES.h, 129
 f64
 STD_TYPES.h, 129
 F_CPU_1024
 TIMER.h, 135
 F_CPU_128
 TIMER.h, 135
 F_CPU_256
 TIMER.h, 135
 F_CPU_32
 TIMER.h, 135
 F_CPU_64
 TIMER.h, 135
 F_CPU_8
 TIMER.h, 135
 F_CPU_CLOCK
 TIMER.h, 135
 F_CPU_EXT_CLK_FALLING
 TIMER.h, 135
 F_CPU_EXT_CLK_RISING
 TIMER.h, 135
 FALLING_EDGE
 EXTI.h, 61
 FALSE
 STD_TYPES.h, 131
 FE
 UART_reg.h, 179
 FEATURE
 NRF24_reg.h, 107
 FIFO_STATUS
 NRF24_reg.h, 107
 FLOOR
 MATH.h, 93
 FLUSH_RX
 NRF24_reg.h, 107
 FLUSH_TX
 NRF24_reg.h, 108
 FOC0
 TIMER_reg.h, 151
 FOC1A
 TIMER_reg.h, 154
 FOC1B
 TIMER_reg.h, 154
 FOC1C

TIMER_reg.h, 154
FOC2
 TIMER_reg.h, 154
FOC3A
 TIMER_reg.h, 155
FOC3B
 TIMER_reg.h, 155
FOC3C
 TIMER_reg.h, 155
FRACTION
 MATH.h, 93

GET_BIT
 BIT_MATH.h, 43
getOrderInfo
 app.h, 31
GIE.c
 GIE_Disable, 215
 GIE_Enable, 215
GIE.h
 GIE_Disable, 70
 GIE_Enable, 70
GIE_Disable
 GIE.c, 215
 GIE.h, 70
GIE_Enable
 GIE.c, 215
 GIE.h, 70
gotoNextJunction
 app.h, 31

H_BIT
 SREG.h, 128
handleOrder
 app.h, 31
HIGH
 STD_TYPES.h, 131

I_BIT
 SREG.h, 128
ICES1
 TIMER_reg.h, 152
ICES3
 TIMER_reg.h, 155
ICF1
 TIMER_reg.h, 156
ICF3
 TIMER_reg.h, 156
ICNC1
 TIMER_reg.h, 152
ICNC3
 TIMER_reg.h, 155
ICR1H
 TIMER_reg.h, 146
ICR1L
 TIMER_reg.h, 147
ICR3H
 TIMER_reg.h, 147
ICR3L
 TIMER_reg.h, 147
IN1A_channel
 WHEELS_CONFIG_t, 26
IN1B_channel
 WHEELS_CONFIG_t, 26
IN2A_channel
 WHEELS_CONFIG_t, 26
IN2B_channel
 WHEELS_CONFIG_t, 26
INIT_PIN
 DIO.c, 204
INPUT
 DIO.h, 49
INT0
 EXTI_reg.h, 68
INT1
 EXTI_reg.h, 68
INT2
 EXTI_reg.h, 68
INT3
 EXTI_reg.h, 68
INT4
 EXTI_reg.h, 68
INT5
 EXTI_reg.h, 68
INT6
 EXTI_reg.h, 68
INT7
 EXTI_reg.h, 68
INTF0
 EXTI_reg.h, 69
INTF1
 EXTI_reg.h, 69
INTF2
 EXTI_reg.h, 69
INTF3
 EXTI_reg.h, 69
INTF4
 EXTI_reg.h, 69
INTF5
 EXTI_reg.h, 69
INTF6
 EXTI_reg.h, 69
INTF7
 EXTI_reg.h, 69
IR.c
 IR_GetCount, 217
 IR_GetStatus, 217
 IR_Init, 218
 IR_ScanAll, 218
IR.h
 IR_0, 72
 IR_1, 72
 IR_2, 72
 IR_GetCount, 72
 IR_GetStatus, 73
 IR_Init, 74
 IR_ScanAll, 74

- IR_SENSOR_t, 72
- IR_0
 - IR.h, 72
- IR_1
 - IR.h, 72
- IR_2
 - IR.h, 72
- IR_cfg.c
 - countIRSensorsConfigured, 219
 - IR_configs, 220
- IR_cfg.h
 - countIRSensorsConfigured, 76
 - IR_configs, 76
- IR_CONFIG_t, 8
 - ActiveHighOrLow, 8
 - pin, 8
 - port, 8
 - sensor, 8
- IR_configs
 - IR_cfg.c, 220
 - IR_cfg.h, 76
- IR_GetCount
 - IR.c, 217
 - IR.h, 72
- IR_GetStatus
 - IR.c, 217
 - IR.h, 73
- IR_Init
 - IR.c, 218
 - IR.h, 74
- IR_ScanAll
 - IR.c, 218
 - IR.h, 74
- IR_SENSOR_t
 - IR.h, 72
- irq
 - PINS_t, 18
- IS_NEGATIVE
 - MATH.h, 94
- IS_POSITIVE
 - MATH.h, 94
- ISC00
 - EXTI_reg.h, 68
- ISC01
 - EXTI_reg.h, 68
- ISC10
 - EXTI_reg.h, 68
- ISC11
 - EXTI_reg.h, 68
- ISC20
 - EXTI_reg.h, 68
- ISC21
 - EXTI_reg.h, 68
- ISC30
 - EXTI_reg.h, 68
- ISC31
 - EXTI_reg.h, 68
- ISC40
 - EXTI_reg.h, 68
- ISC41
 - EXTI_reg.h, 68
- ISC50
 - EXTI_reg.h, 68
- ISC51
 - EXTI_reg.h, 68
- ISC60
 - EXTI_reg.h, 68
- ISC61
 - EXTI_reg.h, 68
- ISC70
 - EXTI_reg.h, 68
- ISC71
 - EXTI_reg.h, 68
- ITEM_PICKUP_BUTTON
 - app_cfg.h, 34
- IVCE
 - EXTI_reg.h, 67
- IVSEL
 - EXTI_reg.h, 67
- led
 - LED_CONFIGS_t, 9
- LED.c
 - LED_Init, 221
 - LED_Toggle, 221
 - LED_TurnOnOff, 222
- LED.h
 - LED_0, 78
 - LED_1, 78
 - LED_2, 78
 - LED_3, 78
 - LED_4, 78
 - LED_5, 78
 - LED_6, 78
 - LED_7, 78
 - LED_Init, 78
 - LED_OFF, 78
 - LED_ON, 78
 - LED_STATE_t, 77
 - LED_t, 78
 - LED_Toggle, 79
 - LED_TurnOnOff, 79
- LED_0
 - LED.h, 78
- LED_1
 - LED.h, 78
- LED_2
 - LED.h, 78
- LED_3
 - LED.h, 78
- LED_4
 - LED.h, 78
- LED_5
 - LED.h, 78
- LED_6
 - LED.h, 78
- LED_7

- LED.h, 78
- LED_cfg.c
 - countLedsConfigured, 224
 - ledConfigs, 224
- LED_cfg.h
 - countLedsConfigured, 81
 - ledConfigs, 81
- LED_CONFIGS_t, 9
 - led, 9
 - pin, 9
 - port, 9
- LED_Init
 - LED.c, 221
 - LED.h, 78
- LED_OFF
 - LED.h, 78
- LED_ON
 - LED.h, 78
- LED_STATE_t
 - LED.h, 77
- LED_t
 - LED.h, 78
- LED_Toggle
 - LED.c, 221
 - LED.h, 79
- LED_TurnOnOff
 - LED.c, 222
 - LED.h, 79
- ledConfigs
 - LED_cfg.c, 224
 - LED_cfg.h, 81
- LERP
 - MATH.h, 95
- LIFTER.c
 - LIFTER_Disable, 225
 - LIFTER_Enable, 225
 - LIFTER_Init, 226
 - LIFTER_MoveDown, 226
 - LIFTER_MoveUp, 226
 - LIFTER_SetOverallStroke, 227
 - LIFTER_SetPulsesPerRevolution, 227
 - LIFTER_SetRevolutionStroke, 227
 - LIFTER_SetSpeed, 227
- LIFTER.h
 - LIFTER_Disable, 83
 - LIFTER_Enable, 83
 - LIFTER_Init, 83
 - LIFTER_MoveDown, 84
 - LIFTER_MoveUp, 84
 - LIFTER_SetOverallStroke, 84
 - LIFTER_SetPulsesPerRevolution, 85
 - LIFTER_SetRevolutionStroke, 85
 - LIFTER_SetSpeed, 85
- LIFTER_cfg.c
 - LifterConfigs, 229
- LIFTER_cfg.h
 - ENABLE_CONNECTION, 87
 - LIFTER_DIR_t, 87
 - LIFTER_DOWN, 87
 - LIFTER_UP, 87
 - LifterConfigs, 87
- LIFTER_CONFIGS_t, 10
 - directionPin, 10
 - directionPinPort, 10
 - enablePin, 10
 - enablePinPort, 10
 - overallStroke, 10
 - pulsePin, 11
 - pulsePinPort, 11
 - pulsesPerRevolution, 11
 - revolutionStroke, 11
 - speed, 11
- LIFTER_DIR_t
 - LIFTER_cfg.h, 87
- LIFTER_Disable
 - LIFTER.c, 225
 - LIFTER.h, 83
- LIFTER_DOWN
 - LIFTER_cfg.h, 87
- LIFTER_Enable
 - LIFTER.c, 225
 - LIFTER.h, 83
- LIFTER_Init
 - LIFTER.c, 226
 - LIFTER.h, 83
- LIFTER_MoveDown
 - LIFTER.c, 226
 - LIFTER.h, 84
- LIFTER_MoveUp
 - LIFTER.c, 226
 - LIFTER.h, 84
- LIFTER_SetOverallStroke
 - LIFTER.c, 227
 - LIFTER.h, 84
- LIFTER_SetPulsesPerRevolution
 - LIFTER.c, 227
 - LIFTER.h, 85
- LIFTER_SetRevolutionStroke
 - LIFTER.c, 227
 - LIFTER.h, 85
- LIFTER_SetSpeed
 - LIFTER.c, 227
 - LIFTER.h, 85
- LIFTER_UP
 - LIFTER_cfg.h, 87
- LifterConfigs
 - LIFTER_cfg.c, 229
 - LIFTER_cfg.h, 87
- LOG
 - MATH.h, 95
- LOG10
 - MATH.h, 95
- LOG2
 - MATH.h, 95
- LOGIC_CHANGE
 - EXTI.h, 61

LOW
 STD_TYPES.h, 131
 LOW_LEVEL_DETECT
 EXTI.h, 61

 main
 appTest.c, 197
 MANUAL_MODE
 app.h, 29
 manualNavigation
 app.h, 31
 MASK_MAX_RT
 NRF24_reg.h, 114
 MASK_RX_DR
 NRF24_reg.h, 114
 MASK_TX_DS
 NRF24_reg.h, 114
 MASTER_PickItem
 app.h, 32
 MATH.h
 ABS, 89
 ACOS, 90
 ASIN, 90
 ATAN, 91
 CEIL, 91
 CLAMP, 91
 COS, 92
 DEG_TO_RAD, 92
 EXP, 93
 FLOOR, 93
 FRACTION, 93
 IS_NEGATIVE, 94
 IS_POSITIVE, 94
 LERP, 95
 LOG, 95
 LOG10, 95
 LOG2, 95
 MAX, 96
 MIN, 96
 PI, 97
 POW, 97
 RAD_TO_DEG, 97
 ROUND, 98
 SIGN, 98
 SIN, 99
 SIZE_OF_ARRAY, 99
 SQRT, 99
 TAN, 100
 MAX
 MATH.h, 96
 MAX_RT
 NRF24_reg.h, 112
 maxVoltage
 BatteryConfigs_t, 5
 MCUCR
 EXTI_reg.h, 67
 MIN
 MATH.h, 96
 minVoltage
 BatteryConfigs_t, 5
 MISO
 SPI_CONNECTIONS_t, 21
 mode
 SPI_CONFIG_t, 20
 UART_CFG_t, 23
 mode_type
 UART_CFG_t, 23
 modifyOrientation
 app.h, 32
 monitorBattery
 app.h, 32
 MOSI
 SPI_CONNECTIONS_t, 21
 MPCM
 UART_reg.h, 179
 MSTR
 SPI_reg.h, 126

 N_BIT
 SREG.h, 128
 navigate
 app.h, 32
 navigateColumn
 app.h, 32
 navigateRow
 app.h, 32
 NAVIGATION_MODE_BUTTON
 app_cfg.h, 34
 NAVIGATION_MODE_t
 app.h, 28
 NEON_LED
 app_cfg.h, 35
 NESTING
 EXTI_cfg.h, 65
 NESTING_DISABLED
 EXTI_cfg.h, 65
 NESTING_ENABLED
 EXTI_cfg.h, 65
 NO_CLOCK
 TIMER.h, 135
 NO_OC
 TIMER.h, 135
 NO_ORDER_RECIEVED
 app.h, 29
 NOP
 NRF24_reg.h, 108
 NORMAL
 STD_TYPES.h, 131
 NORTH
 app.h, 30
 NRF24.c
 NRF24_Available, 231
 NRF24_Init, 231
 NRF24_ReceiveString, 231
 NRF24_RxMode, 231
 NRF24_SendString, 232
 NRF24_TxMode, 232
 NRF24.h

- NRF24_Available, [101](#)
- NRF24_Init, [101](#)
- NRF24_ReceiveString, [102](#)
- NRF24_RxMode, [102](#)
- NRF24_SendString, [102](#)
- NRF24_TxMode, [102](#)
- NRF24_Available
 - NRF24.c, [231](#)
 - NRF24.h, [101](#)
- NRF24_cfg
 - NRF24_cfg.c, [233](#)
 - NRF24_cfg.h, [104](#)
- NRF24_cfg.c
 - NRF24_cfg, [233](#)
- NRF24_cfg.h
 - NRF24_cfg, [104](#)
 - PIPE_t, [104](#)
 - RX_PIPE0, [104](#)
 - RX_PIPE1, [104](#)
 - RX_PIPE2, [104](#)
 - RX_PIPE3, [104](#)
 - RX_PIPE4, [104](#)
 - RX_PIPE5, [104](#)
- NRF24_Init
 - NRF24.c, [231](#)
 - NRF24.h, [101](#)
- NRF24_ReceiveString
 - NRF24.c, [231](#)
 - NRF24.h, [102](#)
- NRF24_reg.h
 - ACTIVATE, [106](#)
 - ARC0, [115](#)
 - ARC1, [115](#)
 - ARC2, [115](#)
 - ARC3, [115](#)
 - ARC_CNT0, [113](#)
 - ARC_CNT1, [113](#)
 - ARC_CNT2, [113](#)
 - ARC_CNT3, [113](#)
 - ARD0, [115](#)
 - ARD1, [115](#)
 - ARD2, [115](#)
 - ARD3, [115](#)
 - AW0, [115](#)
 - AW1, [115](#)
 - CD, [107](#)
 - CONFIG, [107](#)
 - CONT_WAVE, [112](#)
 - CRCO, [114](#)
 - DPL_P0, [114](#)
 - DPL_P1, [114](#)
 - DPL_P2, [114](#)
 - DPL_P3, [114](#)
 - DPL_P4, [114](#)
 - DPL_P5, [114](#)
 - DYNPD, [107](#)
 - DYNPD0, [113](#)
 - DYNPD1, [113](#)
 - DYNPD2, [113](#)
 - DYNPD3, [113](#)
 - DYNPD4, [113](#)
 - DYNPD5, [113](#)
 - EN_AA, [107](#)
 - EN_ACK_PAY, [114](#)
 - EN_CRC, [114](#)
 - EN_DPL, [114](#)
 - EN_DYN_ACK, [114](#)
 - EN_RXADDR, [107](#)
 - ENAA_P0, [115](#)
 - ENAA_P1, [115](#)
 - ENAA_P2, [115](#)
 - ENAA_P3, [115](#)
 - ENAA_P4, [115](#)
 - ENAA_P5, [115](#)
 - ERX_P0, [115](#)
 - ERX_P1, [115](#)
 - ERX_P2, [115](#)
 - ERX_P3, [115](#)
 - ERX_P4, [115](#)
 - ERX_P5, [115](#)
 - FEATURE, [107](#)
 - FIFO_STATUS, [107](#)
 - FLUSH_RX, [107](#)
 - FLUSH_TX, [108](#)
 - MASK_MAX_RT, [114](#)
 - MASK_RX_DR, [114](#)
 - MASK_TX_DS, [114](#)
 - MAX_RT, [112](#)
 - NOP, [108](#)
 - OBSERVE_TX, [108](#)
 - Obsolete, [112](#)
 - PLL_LOCK, [112](#)
 - PLOS_CNT0, [113](#)
 - PLOS_CNT1, [113](#)
 - PLOS_CNT2, [113](#)
 - PLOS_CNT3, [113](#)
 - PRIM_RX, [114](#)
 - PWR_UP, [114](#)
 - R_REGISTER, [108](#)
 - R_RX_PAYLOAD, [108](#)
 - R_RX_PL_WID, [108](#)
 - REGISTER_MASK, [108](#)
 - RESERVED, [112](#)
 - REUSE_TX_PL, [108](#)
 - RF_CH, [109](#)
 - RF_CH0, [112](#)
 - RF_CH1, [112](#)
 - RF_CH2, [112](#)
 - RF_CH3, [112](#)
 - RF_CH4, [112](#)
 - RF_CH5, [112](#)
 - RF_CH6, [112](#)
 - RF_DR_HIGH, [112](#)
 - RF_DR_LOW, [112](#)
 - RF_PWR0, [112](#)
 - RF_PWR1, [112](#)

- RF_SETUP, 109
- RPD, 113
- RX_ADDR_P0, 109
- RX_ADDR_P1, 109
- RX_ADDR_P2, 109
- RX_ADDR_P3, 109
- RX_ADDR_P4, 109
- RX_ADDR_P5, 109
- RX_DR, 112
- RX_EMPTY, 113
- RX_FULL1, 113
- RX_P_NO0, 112
- RX_P_NO1, 112
- RX_P_NO2, 112
- RX_PW_P0, 110
- RX_PW_P1, 110
- RX_PW_P2, 110
- RX_PW_P3, 110
- RX_PW_P4, 110
- RX_PW_P5, 110
- SETUP_AW, 110
- SETUP_RETR, 110
- STATUS, 111
- TX_ADDR, 111
- TX_DS, 112
- TX_EMPTY, 113
- TX_FULL, 113
- TX_FULL0, 112
- TX_REUSE, 113
- W_ACK_PAYLOAD, 111
- W_REGISTER, 111
- W_TX_PAYLOAD, 111
- W_TX_PAYLOAD_NOACK, 111
- NRF24_RxMode
 - NRF24.c, 231
 - NRF24.h, 102
- NRF24_SendString
 - NRF24.c, 232
 - NRF24.h, 102
- NRF24_t, 12
 - addressWidth, 13
 - channel, 13
 - payload_len, 13
 - pins, 13
 - rx0Address, 13
 - rx0Payload, 13
 - rx1Address, 13
 - rx1Payload, 13
 - rx2Address, 14
 - rx2Payload, 14
 - rx3Address, 14
 - rx3Payload, 14
 - rx4Address, 14
 - rx4Payload, 14
 - rx5Address, 14
 - rx5Payload, 14
 - rxPipe, 15
 - txAddress, 15
 - txPayload, 15
- NRF24_TxMode
 - NRF24.c, 232
 - NRF24.h, 102
- NULL
 - STD_TYPES.h, 129
- NULL_BYTE
 - STD_TYPES.h, 129
- NUM_OF_COLUMNS
 - app_cfg.h, 35
- NUM_OF_ROWS
 - app_cfg.h, 35
- OBSERVE_TX
 - NRF24_reg.h, 108
- Obsolete
 - NRF24_reg.h, 112
- OBSTACLE_DISTANCE_THRESHOLD
 - app_cfg.h, 35
- OCF0
 - TIMER_reg.h, 156
- OCF1A
 - TIMER_reg.h, 156
- OCF1B
 - TIMER_reg.h, 156
- OCF1C
 - TIMER_reg.h, 156
- OCF2
 - TIMER_reg.h, 156
- OCF3A
 - TIMER_reg.h, 156
- OCF3B
 - TIMER_reg.h, 156
- OCF3C
 - TIMER_reg.h, 156
- OCIE0
 - TIMER_reg.h, 155
- OCIE1A
 - TIMER_reg.h, 155
- OCIE1B
 - TIMER_reg.h, 155
- OCIE1C
 - TIMER_reg.h, 155
- OCIE2
 - TIMER_reg.h, 155
- OCIE3A
 - TIMER_reg.h, 156
- OCIE3B
 - TIMER_reg.h, 156
- OCIE3C
 - TIMER_reg.h, 155
- OCR0
 - TIMER_reg.h, 147
- OCR0UB
 - TIMER_reg.h, 152
- OCR1AH
 - TIMER_reg.h, 147
- OCR1AL
 - TIMER_reg.h, 147

OCR1BH
 TIMER_reg.h, 147
 OCR1BL
 TIMER_reg.h, 147
 OCR1CH
 TIMER_reg.h, 148
 OCR1CL
 TIMER_reg.h, 148
 OCR2
 TIMER_reg.h, 148
 OCR3AH
 TIMER_reg.h, 148
 OCR3AL
 TIMER_reg.h, 148
 OCR3BH
 TIMER_reg.h, 148
 OCR3BL
 TIMER_reg.h, 148
 OCR3CH
 TIMER_reg.h, 148
 OCR3CL
 TIMER_reg.h, 149
 onJunction
 app.h, 32
 onLeftSharp
 app.h, 32
 onLeftSmooth
 app.h, 32
 onObstacle
 app.h, 33
 onRightSharp
 app.h, 33
 onRightSmooth
 app.h, 33
 onTrack
 app.h, 33
 ORDER_AVAILABILITY_t
 app.h, 29
 ORDER_DELIVERED
 app.h, 30
 ORDER_GOING_BACK
 app.h, 30
 ORDER_INFO_t, 15
 column, 16
 deliveryState, 16
 recievedOrNot, 16
 row, 16
 ORDER_OUT_FOR_DELIVER
 app.h, 30
 ORDER_PROCESSING
 app.h, 30
 ORDER_RECIEVED
 app.h, 29
 ORDER_RETURNED
 app.h, 30
 ORDER_TRACKING_t
 app.h, 29
 ORIENTATION_t
 app.h, 30
 OUTPUT
 DIO.h, 49
 overallStroke
 LIFTER_CONFIGS_t, 10
 parity
 UART_CFG_t, 24
 payload_len
 NRF24_t, 13
 PI
 MATH.h, 97
 pin
 BatteryConfigs_t, 6
 BUTTON_CONFIGS_t, 7
 CONNECTIONS_t, 7
 IR_CONFIG_t, 8
 LED_CONFIGS_t, 9
 PinConfig_t, 17
 SPI_PINS_t, 22
 PIN_0
 DIO.h, 49
 PIN_1
 DIO.h, 49
 PIN_2
 DIO.h, 49
 PIN_3
 DIO.h, 49
 PIN_4
 DIO.h, 49
 PIN_5
 DIO.h, 49
 PIN_6
 DIO.h, 49
 PIN_7
 DIO.h, 49
 PIN_t
 DIO.h, 49
 PINA
 DIO_reg.h, 58
 PINB
 DIO_reg.h, 58
 PINC
 DIO_reg.h, 58
 PinConfig_t, 16
 direction, 17
 pin, 17
 port, 17
 pullup, 17
 pinConfigs
 DIO_cfg.c, 209
 DIO_cfg.h, 56
 PIND
 DIO_reg.h, 58
 PINE
 DIO_reg.h, 59
 PINF
 DIO_reg.h, 59
 PING

- DIO_reg.h, 59
- pins
 - NRF24_t, 13
- PINS_t, 17
 - ce, 18
 - csn, 18
 - irq, 18
- PIPE_t
 - NRF24_cfg.h, 104
- PLL_LOCK
 - NRF24_reg.h, 112
- PLOS_CNT0
 - NRF24_reg.h, 113
- PLOS_CNT1
 - NRF24_reg.h, 113
- PLOS_CNT2
 - NRF24_reg.h, 113
- PLOS_CNT3
 - NRF24_reg.h, 113
- port
 - BatteryConfigs_t, 6
 - BUTTON_CONFIGS_t, 7
 - CONNECTIONS_t, 7
 - IR_CONFIG_t, 8
 - LED_CONFIGS_t, 9
 - PinConfig_t, 17
 - SPI_PINS_t, 22
- PORT_A
 - DIO.h, 50
- PORT_B
 - DIO.h, 50
- PORT_C
 - DIO.h, 50
- PORT_D
 - DIO.h, 50
- PORT_E
 - DIO.h, 50
- PORT_F
 - DIO.h, 50
- PORT_G
 - DIO.h, 50
- PORT_t
 - DIO.h, 50
- PORTA
 - DIO_reg.h, 59
- PORTB
 - DIO_reg.h, 59
- PORTC
 - DIO_reg.h, 59
- PORTD
 - DIO_reg.h, 60
- PORTE
 - DIO_reg.h, 60
- PORTF
 - DIO_reg.h, 60
- PORTG
 - DIO_reg.h, 60
- POW
 - MATH.h, 97
- PRIM_RX
 - NRF24_reg.h, 114
- pullup
 - PinConfig_t, 17
- PULLUP_FALSE
 - DIO.h, 50
- PULLUP_t
 - DIO.h, 50
- PULLUP_TRUE
 - DIO.h, 50
- pulsePin
 - LIFTER_CONFIGS_t, 11
- pulsePinPort
 - LIFTER_CONFIGS_t, 11
- pulsesPerRevolution
 - LIFTER_CONFIGS_t, 11
- PWM0_Write
 - TIMER.c, 242
- PWM_0
 - TIMER.h, 133
- PWM_1
 - TIMER.h, 133
- PWM_2
 - TIMER.h, 133
- PWM_3
 - TIMER.h, 133
- PWM_4
 - TIMER.h, 133
- PWM_5
 - TIMER.h, 133
- PWM_6
 - TIMER.h, 133
- PWM_7
 - TIMER.h, 133
- PWM_Init
 - TIMER.c, 243
 - TIMER.h, 136
- PWM_t
 - TIMER.h, 133
- PWM_Write
 - TIMER.c, 243
 - TIMER.h, 136
- PWR_UP
 - NRF24_reg.h, 114
- R_REGISTER
 - NRF24_reg.h, 108
- R_RX_PAYLOAD
 - NRF24_reg.h, 108
- R_RX_PL_WID
 - NRF24_reg.h, 108
- RAD_TO_DEG
 - MATH.h, 97
- readSerial
 - app.h, 33
- recievedOrNot
 - ORDER_INFO_t, 16
- REGISTER_MASK

- NRF24_reg.h, 108
- registers.h
 - ADC_u8ADCH_REG, 117
 - ADC_u8ADCL_REG, 117
 - ADC_u8ADCSRA_REG, 117
 - ADC_u8ADCSR_B_REG, 117
 - ADC_u8ADMUX_REG, 117
 - EEPROM_u8EEARH_REG, 117
 - EEPROM_u8EEARL_REG, 117
 - EEPROM_u8EECR_REG, 117, 118
 - EEPROM_u8EEDR_REG, 118
- RESERVED
 - NRF24_reg.h, 112
- REUSE_TX_PL
 - NRF24_reg.h, 108
- revolutionStroke
 - LIFTER_CONFIGS_t, 11
- RF_CH
 - NRF24_reg.h, 109
- RF_CH0
 - NRF24_reg.h, 112
- RF_CH1
 - NRF24_reg.h, 112
- RF_CH2
 - NRF24_reg.h, 112
- RF_CH3
 - NRF24_reg.h, 112
- RF_CH4
 - NRF24_reg.h, 112
- RF_CH5
 - NRF24_reg.h, 112
- RF_CH6
 - NRF24_reg.h, 112
- RF_DR_HIGH
 - NRF24_reg.h, 112
- RF_DR_LOW
 - NRF24_reg.h, 112
- RF_PWR0
 - NRF24_reg.h, 112
- RF_PWR1
 - NRF24_reg.h, 112
- RF_SETUP
 - NRF24_reg.h, 109
- RISING_EDGE
 - EXTI.h, 61
- ROBOT_AVAILABLE
 - app.h, 31
- ROBOT_CHARGING
 - app.h, 31
- ROBOT_OBSTACLE_AVOIDANCE
 - app.h, 31
- ROBOT_PROCESSING_OEDER
 - app.h, 31
- ROBOT_STATE_t
 - app.h, 30
- ROTATE_BACK
 - WHEELS_cfg.h, 196
- ROTATE_DIR_t
 - WHEELS_cfg.h, 195
- ROTATE_FWD
 - WHEELS_cfg.h, 196
- ROTATE_LEFT_SHARP
 - WHEELS_cfg.h, 196
- ROTATE_LEFT_SMOOTH
 - WHEELS_cfg.h, 196
- ROTATE_RIGHT_SHARP
 - WHEELS_cfg.h, 196
- ROTATE_RIGHT_SMOOTH
 - WHEELS_cfg.h, 196
- ROTATE_STOP
 - WHEELS_cfg.h, 196
- ROUND
 - MATH.h, 98
- row
 - ORDER_INFO_t, 16
- ROW_t
 - app_cfg.h, 36
- RPD
 - NRF24_reg.h, 113
- rx0Address
 - NRF24_t, 13
- rx0Payload
 - NRF24_t, 13
- rx1Address
 - NRF24_t, 13
- rx1Payload
 - NRF24_t, 13
- rx2Address
 - NRF24_t, 14
- rx2Payload
 - NRF24_t, 14
- rx3Address
 - NRF24_t, 14
- rx3Payload
 - NRF24_t, 14
- rx4Address
 - NRF24_t, 14
- rx4Payload
 - NRF24_t, 14
- rx5Address
 - NRF24_t, 14
- rx5Payload
 - NRF24_t, 14
- RX_ADDR_P0
 - NRF24_reg.h, 109
- RX_ADDR_P1
 - NRF24_reg.h, 109
- RX_ADDR_P2
 - NRF24_reg.h, 109
- RX_ADDR_P3
 - NRF24_reg.h, 109
- RX_ADDR_P4
 - NRF24_reg.h, 109
- RX_ADDR_P5
 - NRF24_reg.h, 109
- RX_DR

- NRF24_reg.h, 112
- RX_EMPTY
 - NRF24_reg.h, 113
- RX_FULL1
 - NRF24_reg.h, 113
- RX_P_NO0
 - NRF24_reg.h, 112
- RX_P_NO1
 - NRF24_reg.h, 112
- RX_P_NO2
 - NRF24_reg.h, 112
- RX_PIPE0
 - NRF24_cfg.h, 104
- RX_PIPE1
 - NRF24_cfg.h, 104
- RX_PIPE2
 - NRF24_cfg.h, 104
- RX_PIPE3
 - NRF24_cfg.h, 104
- RX_PIPE4
 - NRF24_cfg.h, 104
- RX_PIPE5
 - NRF24_cfg.h, 104
- RX_PW_P0
 - NRF24_reg.h, 110
- RX_PW_P1
 - NRF24_reg.h, 110
- RX_PW_P2
 - NRF24_reg.h, 110
- RX_PW_P3
 - NRF24_reg.h, 110
- RX_PW_P4
 - NRF24_reg.h, 110
- RX_PW_P5
 - NRF24_reg.h, 110
- RXB8
 - UART_reg.h, 179
- RXC
 - UART_reg.h, 179
- RXCIE
 - UART_reg.h, 179
- RXEN
 - UART_reg.h, 179
- rxPipe
 - NRF24_t, 15
- s16
 - STD_TYPES.h, 129
- s32
 - STD_TYPES.h, 130
- s64
 - STD_TYPES.h, 130
- s8
 - STD_TYPES.h, 130
- S_BIT
 - SREG.h, 128
- SCK
 - SPI_CONNECTIONS_t, 21
- SE
 - EXTI_reg.h, 67
- sendOrderTrackInfo
 - app.h, 33
- sendPosition
 - app.h, 33
- sendRobotState
 - app.h, 33
- sensor
 - IR_CONFIG_t, 8
- SET_BIT
 - BIT_MATH.h, 43
- SET_OC
 - TIMER.h, 135
- SETUP_AW
 - NRF24_reg.h, 110
- SETUP_RETR
 - NRF24_reg.h, 110
- SHARP_TURN
 - WHEELS.h, 190
- SIGN
 - MATH.h, 98
- SIN
 - MATH.h, 99
- SIZE_OF_ARRAY
 - MATH.h, 99
- size_t
 - STD_TYPES.h, 130
- SM0
 - EXTI_reg.h, 67
- SM1
 - EXTI_reg.h, 67
- SM2
 - EXTI_reg.h, 67
- SMOOTH_TURN
 - WHEELS.h, 190
- SOUTH
 - app.h, 30
- SPCR
 - SPI_reg.h, 126
- SPDR
 - SPI_reg.h, 126
- SPE
 - SPI_reg.h, 126
- speed
 - LIFTER_CONFIGS_t, 11
- SpeedPercentage
 - WHEELS_CONFIG_t, 26
- SPI.c
 - __vector_17, 234
 - SPI_DisableInterrupt, 235
 - SPI_EnableInterrupt, 235
 - SPI_Init, 235
 - SPI_ReceiveByte, 235
 - SPI_ReceiveString, 236
 - SPI_SendByte, 236
 - SPI_SendString, 237
 - SPI_SetCallBack, 237
 - SPI_StcCallBack, 238

- SPI_TransceiveByte, [237](#)
- SPI.h
 - SPI_DisableInterrupt, [119](#)
 - SPI_EnableInterrupt, [119](#)
 - SPI_Init, [119](#)
 - SPI_ReceiveByte, [120](#)
 - SPI_ReceiveString, [120](#)
 - SPI_SendByte, [121](#)
 - SPI_SendString, [121](#)
 - SPI_TransceiveByte, [121](#)
- SPI2X
 - SPI_reg.h, [126](#)
- SPI_cfg.c
 - SPI_Config, [239](#)
- SPI_cfg.h
 - SPI_CLOCK_MODE_t, [123](#)
 - SPI_Config, [124](#)
 - SPI_DATA_ORDER_LSB_FIRST, [123](#)
 - SPI_DATA_ORDER_MSB_FIRST, [123](#)
 - SPI_DATA_ORDER_t, [123](#)
 - SPI_DOUBLE_SPEED_DISABLE, [124](#)
 - SPI_DOUBLE_SPEED_ENABLE, [124](#)
 - SPI_DOUBLE_SPEED_t, [123](#)
 - SPI_MASTER, [124](#)
 - SPI_MODE0, [123](#)
 - SPI_MODE1, [123](#)
 - SPI_MODE2, [123](#)
 - SPI_MODE3, [123](#)
 - SPI_MODE_t, [124](#)
 - SPI_PRESCALER_128, [124](#)
 - SPI_PRESCALER_16, [124](#)
 - SPI_PRESCALER_2, [124](#)
 - SPI_PRESCALER_32, [124](#)
 - SPI_PRESCALER_4, [124](#)
 - SPI_PRESCALER_64, [124](#)
 - SPI_PRESCALER_8, [124](#)
 - SPI_PRESCALER_t, [124](#)
 - SPI_SLAVE, [124](#)
- SPI_CLOCK_MODE_t
 - SPI_cfg.h, [123](#)
- SPI_Config
 - SPI_cfg.c, [239](#)
 - SPI_cfg.h, [124](#)
- SPI_CONFIG_t, [19](#)
 - clockDivider, [19](#)
 - clockMode, [20](#)
 - connections, [20](#)
 - dataOrder, [20](#)
 - doubleSpeed, [20](#)
 - mode, [20](#)
- SPI_CONNECTIONS_t, [21](#)
 - MISO, [21](#)
 - MOSI, [21](#)
 - SCK, [21](#)
 - SS, [22](#)
- SPI_DATA_ORDER_LSB_FIRST
 - SPI_cfg.h, [123](#)
- SPI_DATA_ORDER_MSB_FIRST
 - SPI_cfg.h, [123](#)
- SPI_DATA_ORDER_t
 - SPI_cfg.h, [123](#)
- SPI_DisableInterrupt
 - SPI.c, [235](#)
 - SPI.h, [119](#)
- SPI_DOUBLE_SPEED_DISABLE
 - SPI_cfg.h, [124](#)
- SPI_DOUBLE_SPEED_ENABLE
 - SPI_cfg.h, [124](#)
- SPI_DOUBLE_SPEED_t
 - SPI_cfg.h, [123](#)
- SPI_EnableInterrupt
 - SPI.c, [235](#)
 - SPI.h, [119](#)
- SPI_Init
 - SPI.c, [235](#)
 - SPI.h, [119](#)
- SPI_MASTER
 - SPI_cfg.h, [124](#)
- SPI_MODE0
 - SPI_cfg.h, [123](#)
- SPI_MODE1
 - SPI_cfg.h, [123](#)
- SPI_MODE2
 - SPI_cfg.h, [123](#)
- SPI_MODE3
 - SPI_cfg.h, [123](#)
- SPI_MODE_t
 - SPI_cfg.h, [124](#)
- SPI_PINS_t, [22](#)
 - pin, [22](#)
 - port, [22](#)
- SPI_PRESCALER_128
 - SPI_cfg.h, [124](#)
- SPI_PRESCALER_16
 - SPI_cfg.h, [124](#)
- SPI_PRESCALER_2
 - SPI_cfg.h, [124](#)
- SPI_PRESCALER_32
 - SPI_cfg.h, [124](#)
- SPI_PRESCALER_4
 - SPI_cfg.h, [124](#)
- SPI_PRESCALER_64
 - SPI_cfg.h, [124](#)
- SPI_PRESCALER_8
 - SPI_cfg.h, [124](#)
- SPI_PRESCALER_t
 - SPI_cfg.h, [124](#)
- SPI_ReceiveByte
 - SPI.c, [235](#)
 - SPI.h, [120](#)
- SPI_ReceiveString
 - SPI.c, [236](#)
 - SPI.h, [120](#)
- SPI_reg.h
 - CPHA, [126](#)
 - CPOL, [126](#)

- DORD, [126](#)
- MSTR, [126](#)
- SPCR, [126](#)
- SPDR, [126](#)
- SPE, [126](#)
- SPI2X, [126](#)
- SPIE, [126](#)
- SPIF, [126](#)
- SPR0, [126](#)
- SPR1, [126](#)
- SPSR, [126](#)
- WCOL, [126](#)
- SPI_SendByte
 - SPI.c, [236](#)
 - SPI.h, [121](#)
- SPI_SendString
 - SPI.c, [237](#)
 - SPI.h, [121](#)
- SPI_SetCallBack
 - SPI.c, [237](#)
- SPI_SLAVE
 - SPI_cfg.h, [124](#)
- SPI_StcCallBack
 - SPI.c, [238](#)
- SPI_TrancieveByte
 - SPI.c, [237](#)
 - SPI.h, [121](#)
- SPIE
 - SPI_reg.h, [126](#)
- SPIF
 - SPI_reg.h, [126](#)
- SPR0
 - SPI_reg.h, [126](#)
- SPR1
 - SPI_reg.h, [126](#)
- SPSR
 - SPI_reg.h, [126](#)
- SQRT
 - MATH.h, [99](#)
- SRE
 - EXTI_reg.h, [67](#)
- SREG
 - SREG.h, [127](#)
- SREG.h
 - C_BIT, [128](#)
 - H_BIT, [128](#)
 - I_BIT, [128](#)
 - N_BIT, [128](#)
 - S_BIT, [128](#)
 - SREG, [127](#)
 - T_BIT, [128](#)
 - V_BIT, [128](#)
 - Z_BIT, [128](#)
- SRW10
 - EXTI_reg.h, [67](#)
- SS
 - SPI_CONNECTIONS_t, [22](#)
- STATE_t
 - STD_TYPES.h, [131](#)
- STATUS
 - NRF24_reg.h, [111](#)
- STD_TYPES.h
 - ACTIVATION_STATUS_t, [130](#)
 - ACTIVE_HIGH, [131](#)
 - ACTIVE_LOW, [131](#)
 - BOOL_t, [131](#)
 - ERROR_NO, [131](#)
 - ERROR_STATUS_t, [131](#)
 - ERROR_TIMEOUT, [131](#)
 - ERROR_YES, [131](#)
 - f32, [129](#)
 - f64, [129](#)
 - FALSE, [131](#)
 - HIGH, [131](#)
 - LOW, [131](#)
 - NORMAL, [131](#)
 - NULL, [129](#)
 - NULL_BYTE, [129](#)
 - s16, [129](#)
 - s32, [130](#)
 - s64, [130](#)
 - s8, [130](#)
 - size_t, [130](#)
 - STATE_t, [131](#)
 - TRUE, [131](#)
 - u16, [130](#)
 - u32, [130](#)
 - u64, [130](#)
 - u8, [130](#)
- stop_bits
 - UART_CFG_t, [24](#)
- T_BIT
 - SREG.h, [128](#)
- TAN
 - MATH.h, [100](#)
- TCCR0
 - TIMER_reg.h, [149](#)
- TCCR1A
 - TIMER_reg.h, [149](#)
- TCCR1B
 - TIMER_reg.h, [149](#)
- TCCR1C
 - TIMER_reg.h, [149](#)
- TCCR2
 - TIMER_reg.h, [149](#)
- TCCR3A
 - TIMER_reg.h, [149](#)
- TCCR3B
 - TIMER_reg.h, [150](#)
- TCCR3C
 - TIMER_reg.h, [150](#)
- TCN0UB
 - TIMER_reg.h, [152](#)
- TCNT0
 - TIMER_reg.h, [150](#)
- TCNT1H

- TIMER_reg.h, [150](#)
- TCNT1L
 - TIMER_reg.h, [150](#)
- TCNT2
 - TIMER_reg.h, [150](#)
- TCNT3H
 - TIMER_reg.h, [150](#)
- TCNT3L
 - TIMER_reg.h, [150](#)
- TCR0UB
 - TIMER_reg.h, [152](#)
- TICIE1
 - TIMER_reg.h, [155](#)
- TICIE3
 - TIMER_reg.h, [156](#)
- TIMER.c
 - __vector_10, [241](#)
 - __vector_11, [241](#)
 - __vector_12, [241](#)
 - __vector_13, [241](#)
 - __vector_14, [241](#)
 - __vector_15, [241](#)
 - __vector_16, [241](#)
 - __vector_24, [241](#)
 - __vector_25, [242](#)
 - __vector_26, [242](#)
 - __vector_27, [242](#)
 - __vector_28, [242](#)
 - __vector_29, [242](#)
 - __vector_9, [242](#)
 - PWM0_Write, [242](#)
 - PWM_Init, [243](#)
 - PWM_Write, [243](#)
 - TIMER0_DisableCompareMatchInterrupt, [244](#)
 - TIMER0_DisableOverflowInterrupt, [244](#)
 - TIMER0_EnableCompareMatchInterrupt, [245](#)
 - TIMER0_EnableOverflowInterrupt, [245](#)
 - TIMER0_GetTimerValue, [245](#)
 - TIMER0_Init, [245](#)
 - TIMER0_SetCallBack, [246](#)
 - TIMER0_SetCompareCallBack, [246](#)
 - TIMER0_SetCompareValue, [246](#)
 - TIMER0_SetTimer, [247](#)
 - TIMER1_DisableCaptureInterrupt, [247](#)
 - TIMER1_DisableCompareMatchInterrupt, [247](#)
 - TIMER1_DisableOverflowInterrupt, [248](#)
 - TIMER1_EnableCaptureInterrupt, [248](#)
 - TIMER1_EnableCompareMatchInterrupt, [248](#)
 - TIMER1_EnableOverflowInterrupt, [249](#)
 - TIMER1_GetTimerValue, [249](#)
 - TIMER1_Init, [249](#)
 - TIMER1_SetCallBack, [250](#)
 - TIMER1_SetCaptureCallBack, [250](#)
 - TIMER1_SetCompareCallBackA, [250](#)
 - TIMER1_SetCompareCallBackB, [251](#)
 - TIMER1_SetCompareCallBackC, [251](#)
 - TIMER1_SetCompareValue, [251](#)
 - TIMER1_SetTimer, [252](#)
- TIMER2_SetCallBack, [252](#)
- TIMER2_SetCompareCallBack, [253](#)
- TIMER3_SetCallBack, [253](#)
- TIMER3_SetCaptureCallBack, [253](#)
- TIMER3_SetCompareCallBackA, [253](#)
- TIMER3_SetCompareCallBackB, [253](#)
- TIMER3_SetCompareCallBackC, [253](#)
- TIMER.h
 - CLEAR_OC, [135](#)
 - F_CPU_1024, [135](#)
 - F_CPU_128, [135](#)
 - F_CPU_256, [135](#)
 - F_CPU_32, [135](#)
 - F_CPU_64, [135](#)
 - F_CPU_8, [135](#)
 - F_CPU_CLOCK, [135](#)
 - F_CPU_EXT_CLK_FALLING, [135](#)
 - F_CPU_EXT_CLK_RISING, [135](#)
 - NO_CLOCK, [135](#)
 - NO_OC, [135](#)
 - PWM_0, [133](#)
 - PWM_1, [133](#)
 - PWM_2, [133](#)
 - PWM_3, [133](#)
 - PWM_4, [133](#)
 - PWM_5, [133](#)
 - PWM_6, [133](#)
 - PWM_7, [133](#)
 - PWM_Init, [136](#)
 - PWM_t, [133](#)
 - PWM_Write, [136](#)
 - SET_OC, [135](#)
 - TIMER0_DisableCompareMatchInterrupt, [137](#)
 - TIMER0_DisableOverflowInterrupt, [137](#)
 - TIMER0_EnableCompareMatchInterrupt, [138](#)
 - TIMER0_EnableOverflowInterrupt, [138](#)
 - TIMER0_GetTimerValue, [138](#)
 - TIMER0_Init, [138](#)
 - TIMER0_SetCompareValue, [139](#)
 - TIMER0_SetTimer, [139](#)
 - TIMER1_DisableCaptureInterrupt, [139](#)
 - TIMER1_DisableCompareMatchInterrupt, [140](#)
 - TIMER1_DisableOverflowInterrupt, [140](#)
 - TIMER1_EnableCaptureInterrupt, [140](#)
 - TIMER1_EnableCompareMatchInterrupt, [141](#)
 - TIMER1_EnableOverflowInterrupt, [141](#)
 - TIMER1_GetTimerValue, [141](#)
 - TIMER1_Init, [142](#)
 - TIMER1_SetCompareValue, [142](#)
 - TIMER1_SetTimer, [143](#)
 - TIMER_CLOCK_t, [133](#)
 - TIMER_MODE_CTC, [135](#)
 - TIMER_MODE_CTC_ICR, [135](#)
 - TIMER_MODE_CTC_OCR, [135](#)
 - TIMER_MODE_FAST_PWM, [135](#)
 - TIMER_MODE_FAST_PWM_10, [135](#)
 - TIMER_MODE_FAST_PWM_8, [135](#)
 - TIMER_MODE_FAST_PWM_9, [135](#)

- TIMER_MODE_FAST_PWM_ICR, [135](#)
- TIMER_MODE_FAST_PWM_OCR, [135](#)
- TIMER_MODE_NORMAL, [135](#)
- TIMER_MODE_PHASE_CORRECT_PWM, [135](#)
- TIMER_MODE_PHASE_CORRECT_PWM_10, [135](#)
- TIMER_MODE_PHASE_CORRECT_PWM_8, [135](#)
- TIMER_MODE_PHASE_CORRECT_PWM_9, [135](#)
- TIMER_MODE_PHASE_CORRECT_PWM_ICR, [135](#)
- TIMER_MODE_PHASE_CORRECT_PWM_OCR, [135](#)
- TIMER_MODE_PHASE_FREQ_CORRECT_ICR, [135](#)
- TIMER_MODE_PHASE_FREQ_CORRECT_OCR, [135](#)
- TIMER_MODE_t, [135](#)
- TIMER_OC_t, [135](#)
- TIMER_OCA, [136](#)
- TIMER_OCB, [136](#)
- TIMER_OCC, [136](#)
- TIMER_OCx_t, [136](#)
- TOGGLE_OC, [135](#)
- TIMER0_DisableCompareMatchInterrupt
 - TIMER.c, [244](#)
 - TIMER.h, [137](#)
- TIMER0_DisableOverflowInterrupt
 - TIMER.c, [244](#)
 - TIMER.h, [137](#)
- TIMER0_EnableCompareMatchInterrupt
 - TIMER.c, [245](#)
 - TIMER.h, [138](#)
- TIMER0_EnableOverflowInterrupt
 - TIMER.c, [245](#)
 - TIMER.h, [138](#)
- TIMER0_GetTimerValue
 - TIMER.c, [245](#)
 - TIMER.h, [138](#)
- TIMER0_Init
 - TIMER.c, [245](#)
 - TIMER.h, [138](#)
- TIMER0_SetCallBack
 - TIMER.c, [246](#)
- TIMER0_SetCompareCallBack
 - TIMER.c, [246](#)
- TIMER0_SetCompareValue
 - TIMER.c, [246](#)
 - TIMER.h, [139](#)
- TIMER0_SetTimer
 - TIMER.c, [247](#)
 - TIMER.h, [139](#)
- TIMER1_DisableCaptureInterrupt
 - TIMER.c, [247](#)
 - TIMER.h, [139](#)
- TIMER1_DisableCompareMatchInterrupt
 - TIMER.c, [247](#)
 - TIMER.h, [140](#)
- TIMER1_DisableOverflowInterrupt
 - TIMER.c, [248](#)
 - TIMER.h, [140](#)
- TIMER1_EnableCaptureInterrupt
 - TIMER.c, [248](#)
 - TIMER.h, [140](#)
- TIMER1_EnableCompareMatchInterrupt
 - TIMER.c, [248](#)
 - TIMER.h, [141](#)
- TIMER1_EnableOverflowInterrupt
 - TIMER.c, [249](#)
 - TIMER.h, [141](#)
- TIMER1_GetTimerValue
 - TIMER.c, [249](#)
 - TIMER.h, [141](#)
- TIMER1_Init
 - TIMER.c, [249](#)
 - TIMER.h, [142](#)
- TIMER1_SetCallBack
 - TIMER.c, [250](#)
- TIMER1_SetCaptureCallBack
 - TIMER.c, [250](#)
- TIMER1_SetCompareCallBackA
 - TIMER.c, [250](#)
- TIMER1_SetCompareCallBackB
 - TIMER.c, [251](#)
- TIMER1_SetCompareCallBackC
 - TIMER.c, [251](#)
- TIMER1_SetCompareValue
 - TIMER.c, [251](#)
 - TIMER.h, [142](#)
- TIMER1_SetTimer
 - TIMER.c, [252](#)
 - TIMER.h, [143](#)
- TIMER2_SetCallBack
 - TIMER.c, [252](#)
- TIMER2_SetCompareCallBack
 - TIMER.c, [253](#)
- TIMER3_SetCallBack
 - TIMER.c, [253](#)
- TIMER3_SetCaptureCallBack
 - TIMER.c, [253](#)
- TIMER3_SetCompareCallBackA
 - TIMER.c, [253](#)
- TIMER3_SetCompareCallBackB
 - TIMER.c, [253](#)
- TIMER3_SetCompareCallBackC
 - TIMER.c, [253](#)
- TIMER_CLOCK_t
 - TIMER.h, [133](#)
- TIMER_MODE CTC
 - TIMER.h, [135](#)
- TIMER_MODE CTC_ICR
 - TIMER.h, [135](#)
- TIMER_MODE CTC_OCR
 - TIMER.h, [135](#)
- TIMER_MODE_FAST_PWM
 - TIMER.h, [135](#)
- TIMER_MODE_FAST_PWM_10

TIMER.h, [135](#)
TIMER_MODE_FAST_PWM_8
 TIMER.h, [135](#)
TIMER_MODE_FAST_PWM_9
 TIMER.h, [135](#)
TIMER_MODE_FAST_PWM_ICR
 TIMER.h, [135](#)
TIMER_MODE_FAST_PWM_OCR
 TIMER.h, [135](#)
TIMER_MODE_NORMAL
 TIMER.h, [135](#)
TIMER_MODE_PHASE_CORRECT_PWM
 TIMER.h, [135](#)
TIMER_MODE_PHASE_CORRECT_PWM_10
 TIMER.h, [135](#)
TIMER_MODE_PHASE_CORRECT_PWM_8
 TIMER.h, [135](#)
TIMER_MODE_PHASE_CORRECT_PWM_9
 TIMER.h, [135](#)
TIMER_MODE_PHASE_CORRECT_PWM_ICR
 TIMER.h, [135](#)
TIMER_MODE_PHASE_CORRECT_PWM_OCR
 TIMER.h, [135](#)
TIMER_MODE_PHASE_FREQ_CORRECT_ICR
 TIMER.h, [135](#)
TIMER_MODE_PHASE_FREQ_CORRECT_OCR
 TIMER.h, [135](#)
TIMER_MODE_t
 TIMER.h, [135](#)
TIMER_OC_t
 TIMER.h, [135](#)
TIMER_OCA
 TIMER.h, [136](#)
TIMER_OCB
 TIMER.h, [136](#)
TIMER_OCC
 TIMER.h, [136](#)
TIMER_OCx_t
 TIMER.h, [136](#)
TIMER_reg.h
 AS0, [152](#)
 ASSR, [146](#)
 COM00, [151](#)
 COM01, [151](#)
 COM1A0, [152](#)
 COM1A1, [152](#)
 COM1B0, [152](#)
 COM1B1, [152](#)
 COM1C0, [152](#)
 COM1C1, [152](#)
 COM20, [154](#)
 COM21, [154](#)
 COM3A0, [154](#)
 COM3A1, [154](#)
 COM3B0, [154](#)
 COM3B1, [154](#)
 COM3C0, [154](#)
 COM3C1, [154](#)
 CS00, [151](#)
 CS01, [151](#)
 CS02, [151](#)
 CS10, [152](#)
 CS11, [152](#)
 CS12, [152](#)
 CS20, [154](#)
 CS21, [154](#)
 CS22, [154](#)
 CS30, [154](#)
 CS31, [155](#)
 CS32, [155](#)
 FOC0, [151](#)
 FOC1A, [154](#)
 FOC1B, [154](#)
 FOC1C, [154](#)
 FOC2, [154](#)
 FOC3A, [155](#)
 FOC3B, [155](#)
 FOC3C, [155](#)
 ICES1, [152](#)
 ICES3, [155](#)
 ICF1, [156](#)
 ICF3, [156](#)
 ICNC1, [152](#)
 ICNC3, [155](#)
 ICR1H, [146](#)
 ICR1L, [147](#)
 ICR3H, [147](#)
 ICR3L, [147](#)
 OCF0, [156](#)
 OCF1A, [156](#)
 OCF1B, [156](#)
 OCF1C, [156](#)
 OCF2, [156](#)
 OCF3A, [156](#)
 OCF3B, [156](#)
 OCF3C, [156](#)
 OCIE0, [155](#)
 OCIE1A, [155](#)
 OCIE1B, [155](#)
 OCIE1C, [155](#)
 OCIE2, [155](#)
 OCIE3A, [156](#)
 OCIE3B, [156](#)
 OCIE3C, [155](#)
 OCR0, [147](#)
 OCR0UB, [152](#)
 OCR1AH, [147](#)
 OCR1AL, [147](#)
 OCR1BH, [147](#)
 OCR1BL, [147](#)
 OCR1CH, [148](#)
 OCR1CL, [148](#)
 OCR2, [148](#)
 OCR3AH, [148](#)
 OCR3AL, [148](#)
 OCR3BH, [148](#)

- OCR3BL, [148](#)
- OCR3CH, [148](#)
- OCR3CL, [149](#)
- TCCR0, [149](#)
- TCCR1A, [149](#)
- TCCR1B, [149](#)
- TCCR1C, [149](#)
- TCCR2, [149](#)
- TCCR3A, [149](#)
- TCCR3B, [150](#)
- TCCR3C, [150](#)
- TCN0UB, [152](#)
- TCNT0, [150](#)
- TCNT1H, [150](#)
- TCNT1L, [150](#)
- TCNT2, [150](#)
- TCNT3H, [150](#)
- TCNT3L, [150](#)
- TCR0UB, [152](#)
- TICIE1, [155](#)
- TICIE3, [156](#)
- TIMER_u8ETIFR_REG, [151](#)
- TIMER_u8ETIMSK_REG, [151](#)
- TIMER_u8TIFR_REG, [151](#)
- TIMER_u8TIMSK_REG, [151](#)
- TOIE0, [155](#)
- TOIE1, [155](#)
- TOIE2, [155](#)
- TOIE3, [155](#)
- TOV0, [156](#)
- TOV1, [156](#)
- TOV2, [156](#)
- TOV3, [156](#)
- WGM00, [151](#)
- WGM01, [151](#)
- WGM10, [152](#)
- WGM11, [152](#)
- WGM12, [152](#)
- WGM13, [152](#)
- WGM20, [154](#)
- WGM21, [154](#)
- WGM30, [154](#)
- WGM31, [154](#)
- WGM32, [155](#)
- WGM33, [155](#)
- TIMER_u8ETIFR_REG
 - TIMER_reg.h, [151](#)
- TIMER_u8ETIMSK_REG
 - TIMER_reg.h, [151](#)
- TIMER_u8TIFR_REG
 - TIMER_reg.h, [151](#)
- TIMER_u8TIMSK_REG
 - TIMER_reg.h, [151](#)
- TOG_BIT
 - BIT_MATH.h, [44](#)
- TOGGLE_OC
 - TIMER.h, [135](#)
- TOIE0
 - TIMER_reg.h, [155](#)
- TOIE1
 - TIMER_reg.h, [155](#)
- TOIE2
 - TIMER_reg.h, [155](#)
- TOIE3
 - TIMER_reg.h, [155](#)
- TOV0
 - TIMER_reg.h, [156](#)
- TOV1
 - TIMER_reg.h, [156](#)
- TOV2
 - TIMER_reg.h, [156](#)
- TOV3
 - TIMER_reg.h, [156](#)
- TRUE
 - STD_TYPES.h, [131](#)
- TX_ADDR
 - NRF24_reg.h, [111](#)
- TX_DS
 - NRF24_reg.h, [112](#)
- TX_EMPTY
 - NRF24_reg.h, [113](#)
- TX_FULL
 - NRF24_reg.h, [113](#)
- TX_FULL0
 - NRF24_reg.h, [112](#)
- TX_REUSE
 - NRF24_reg.h, [113](#)
- txAddress
 - NRF24_t, [15](#)
- TXB8
 - UART_reg.h, [179](#)
- TXC
 - UART_reg.h, [179](#)
- TXCIE
 - UART_reg.h, [179](#)
- TXEN
 - UART_reg.h, [179](#)
- txPayload
 - NRF24_t, [15](#)
- u16
 - STD_TYPES.h, [130](#)
- U2X
 - UART_reg.h, [179](#)
- u32
 - STD_TYPES.h, [130](#)
- u64
 - STD_TYPES.h, [130](#)
- u8
 - STD_TYPES.h, [130](#)
- UART.c
 - __vector_18, [256](#)
 - __vector_19, [256](#)
 - __vector_20, [257](#)
 - __vector_30, [257](#)
 - __vector_31, [257](#)
 - __vector_32, [257](#)

- UART0_Available, [257](#)
- UART0_Disable, [257](#)
- UART0_Enable, [258](#)
- UART0_Flush, [258](#)
- UART0_Init, [258](#)
- UART0_Receive9BitData, [258](#)
- UART0_Receive9BitData_NoBlock, [259](#)
- UART0_ReceiveByte, [259](#)
- UART0_ReceiveByte_NoBlock, [259](#)
- UART0_RX_InterruptDisable, [260](#)
- UART0_RX_InterruptEnable, [260](#)
- UART0_Send9BitData, [261](#)
- UART0_Send9BitData_NoBlock, [261](#)
- UART0_SendByte, [261](#)
- UART0_SendByte_NoBlock, [262](#)
- UART0_TX_InterruptDisable, [262](#)
- UART0_TX_InterruptEnable, [262](#)
- UART0_UDRE_InterruptDisable, [263](#)
- UART0_UDRE_InterruptEnable, [263](#)
- UART1_Available, [264](#)
- UART1_Disable, [264](#)
- UART1_Enable, [264](#)
- UART1_Flush, [265](#)
- UART1_Init, [265](#)
- UART1_Receive9BitData, [265](#)
- UART1_Receive9BitData_NoBlock, [266](#)
- UART1_ReceiveByte, [266](#)
- UART1_ReceiveByte_NoBlock, [266](#)
- UART1_RX_InterruptDisable, [267](#)
- UART1_RX_InterruptEnable, [267](#)
- UART1_Send9BitData, [268](#)
- UART1_Send9BitData_NoBlock, [268](#)
- UART1_SendByte, [269](#)
- UART1_SendByte_NoBlock, [269](#)
- UART1_TX_InterruptDisable, [269](#)
- UART1_TX_InterruptEnable, [269](#)
- UART1_UDRE_InterruptDisable, [270](#)
- UART1_UDRE_InterruptEnable, [270](#)
- UART_Callback_typedef_fn, [256](#)
- UART.h
 - UART0_Available, [159](#)
 - UART0_Disable, [159](#)
 - UART0_Enable, [159](#)
 - UART0_Flush, [159](#)
 - UART0_Init, [160](#)
 - UART0_Receive9BitData, [160](#)
 - UART0_ReceiveByte, [160](#)
 - UART0_ReceiveByte_NoBlock, [161](#)
 - UART0_RX_InterruptDisable, [161](#)
 - UART0_RX_InterruptEnable, [161](#)
 - UART0_Send9BitData, [162](#)
 - UART0_Send9BitData_NoBlock, [162](#)
 - UART0_SendByte, [162](#)
 - UART0_SendByte_NoBlock, [163](#)
 - UART0_TX_InterruptDisable, [163](#)
 - UART0_TX_InterruptEnable, [163](#)
 - UART0_UDRE_InterruptDisable, [164](#)
 - UART0_UDRE_InterruptEnable, [164](#)
 - UART1_Available, [165](#)
 - UART1_Disable, [165](#)
 - UART1_Enable, [165](#)
 - UART1_Flush, [166](#)
 - UART1_Init, [166](#)
 - UART1_Receive9BitData, [166](#)
 - UART1_ReceiveByte, [167](#)
 - UART1_ReceiveByte_NoBlock, [167](#)
 - UART1_RX_InterruptDisable, [168](#)
 - UART1_RX_InterruptEnable, [168](#)
 - UART1_Send9BitData, [169](#)
 - UART1_Send9BitData_NoBlock, [169](#)
 - UART1_SendByte, [170](#)
 - UART1_SendByte_NoBlock, [170](#)
 - UART1_TX_InterruptDisable, [170](#)
 - UART1_TX_InterruptEnable, [170](#)
 - UART1_UDRE_InterruptDisable, [171](#)
 - UART1_UDRE_InterruptEnable, [171](#)
- UART0_Available
 - UART.c, [257](#)
 - UART.h, [159](#)
- UART0_Configs
 - UART_cfg.c, [272](#)
 - UART_cfg.h, [175](#)
- UART0_Disable
 - UART.c, [257](#)
 - UART.h, [159](#)
- UART0_Enable
 - UART.c, [258](#)
 - UART.h, [159](#)
- UART0_Flush
 - UART.c, [258](#)
 - UART.h, [159](#)
- UART0_Init
 - UART.c, [258](#)
 - UART.h, [160](#)
- UART0_Receive9BitData
 - UART.c, [258](#)
 - UART.h, [160](#)
- UART0_Receive9BitData_NoBlock
 - UART.c, [259](#)
- UART0_Receive9BitString
 - UART_service.c, [274](#)
 - UART_service.h, [181](#)
- UART0_ReceiveByte
 - UART.c, [259](#)
 - UART.h, [160](#)
- UART0_ReceiveByte_NoBlock
 - UART.c, [259](#)
 - UART.h, [161](#)
- UART0_ReceiveString
 - UART_service.c, [274](#)
 - UART_service.h, [181](#)
- UART0_ReceiveString_Checksum
 - UART_service.c, [275](#)
 - UART_service.h, [182](#)
- UART0_RX_InterruptDisable
 - UART.c, [260](#)

- UART.h, [161](#)
- UART0_RX_InterruptEnable
 - UART.c, [260](#)
 - UART.h, [161](#)
- UART0_Send9BitData
 - UART.c, [261](#)
 - UART.h, [162](#)
- UART0_Send9BitData_NoBlock
 - UART.c, [261](#)
 - UART.h, [162](#)
- UART0_Send9BitString
 - UART_service.c, [275](#)
 - UART_service.h, [182](#)
- UART0_Send9BitString_Asynchronous
 - UART_service.c, [276](#)
- UART0_SendByte
 - UART.c, [261](#)
 - UART.h, [162](#)
- UART0_SendByte_NoBlock
 - UART.c, [262](#)
 - UART.h, [163](#)
- UART0_SendFloat
 - UART_service.c, [276](#)
 - UART_service.h, [183](#)
- UART0_SendInteger
 - UART_service.c, [276](#)
 - UART_service.h, [183](#)
- UART0_SendString
 - UART_service.c, [278](#)
 - UART_service.h, [184](#)
- UART0_SendString_Asynchronous
 - UART_service.c, [278](#)
 - UART_service.h, [184](#)
- UART0_SendString_Checksum
 - UART_service.c, [279](#)
 - UART_service.h, [185](#)
- UART0_TX_InterruptDisable
 - UART.c, [262](#)
 - UART.h, [163](#)
- UART0_TX_InterruptEnable
 - UART.c, [262](#)
 - UART.h, [163](#)
- UART0_UDRE_InterruptDisable
 - UART.c, [263](#)
 - UART.h, [164](#)
- UART0_UDRE_InterruptEnable
 - UART.c, [263](#)
 - UART.h, [164](#)
- UART1_Available
 - UART.c, [264](#)
 - UART.h, [165](#)
- UART1_Configs
 - UART_cfg.c, [272](#)
 - UART_cfg.h, [175](#)
- UART1_Disable
 - UART.c, [264](#)
 - UART.h, [165](#)
- UART1_Enable
 - UART.c, [264](#)
 - UART.h, [165](#)
- UART1_Flush
 - UART.c, [265](#)
 - UART.h, [166](#)
- UART1_Init
 - UART.c, [265](#)
 - UART.h, [166](#)
- UART1_Receive9BitData
 - UART.c, [265](#)
 - UART.h, [166](#)
- UART1_Receive9BitData_NoBlock
 - UART.c, [266](#)
- UART1_Receive9BitString
 - UART_service.c, [279](#)
 - UART_service.h, [185](#)
- UART1_ReceiveByte
 - UART.c, [266](#)
 - UART.h, [167](#)
- UART1_ReceiveByte_NoBlock
 - UART.c, [266](#)
 - UART.h, [167](#)
- UART1_ReceiveString
 - UART_service.c, [280](#)
 - UART_service.h, [186](#)
- UART1_ReceiveString_Checksum
 - UART_service.c, [280](#)
- UART1_RX_InterruptDisable
 - UART.c, [267](#)
 - UART.h, [168](#)
- UART1_RX_InterruptEnable
 - UART.c, [267](#)
 - UART.h, [168](#)
- UART1_Send9BitData
 - UART.c, [268](#)
 - UART.h, [169](#)
- UART1_Send9BitData_NoBlock
 - UART.c, [268](#)
 - UART.h, [169](#)
- UART1_Send9BitString
 - UART_service.c, [280](#)
 - UART_service.h, [186](#)
- UART1_Send9BitString_Asynchronous
 - UART_service.c, [281](#)
- UART1_SendByte
 - UART.c, [269](#)
 - UART.h, [170](#)
- UART1_SendByte_NoBlock
 - UART.c, [269](#)
 - UART.h, [170](#)
- UART1_SendFloat
 - UART_service.c, [281](#)
 - UART_service.h, [187](#)
- UART1_SendInteger
 - UART_service.c, [282](#)
 - UART_service.h, [187](#)
- UART1_SendString
 - UART_service.c, [282](#)

- UART_service.h, 188
- UART1_SendString_Asynchrous
 - UART_service.c, 283
 - UART_service.h, 188
- UART1_SendString_Checksum
 - UART_service.c, 283
- UART1_TX_InterruptDisable
 - UART.c, 269
 - UART.h, 170
- UART1_TX_InterruptEnable
 - UART.c, 269
 - UART.h, 170
- UART1_UDRE_InterruptDisable
 - UART.c, 270
 - UART.h, 171
- UART1_UDRE_InterruptEnable
 - UART.c, 270
 - UART.h, 171
- UART_Callback_typedef_fn
 - UART.c, 256
- UART_cfg.c
 - UART0_Configs, 272
 - UART1_Configs, 272
- UART_cfg.h
 - UART0_Configs, 175
 - UART1_Configs, 175
 - UART_CLOCK_POLARITY_t, 173
 - UART_DATA_5_BITS, 174
 - UART_DATA_6_BITS, 174
 - UART_DATA_7_BITS, 174
 - UART_DATA_8_BITS, 174
 - UART_DATA_9_BITS, 174
 - UART_DATA_BITS_t, 174
 - UART_FALLING_EDGE_CLOCK, 173
 - UART_MODE_ASYNCHRONOUS_DOUBLE_SPEED, 174
 - UART_MODE_ASYNCHRONOUS_NORMAL, 174
 - UART_MODE_RX, 174
 - UART_MODE_SYNCHRONOUS_MASTER, 174
 - UART_MODE_SYNCHRONOUS_SLAVE, 174
 - UART_MODE_t, 174
 - UART_MODE_TX, 174
 - UART_MODE_TX_RX, 174
 - UART_MODE_TYPE_t, 174
 - UART_PARITY_DISABLE, 174
 - UART_PARITY_EVEN, 174
 - UART_PARITY_ODD, 174
 - UART_PARITY_t, 174
 - UART_RISING_EDGE_CLOCK, 173
 - UART_STOP_1_BIT, 175
 - UART_STOP_2_BIT, 175
 - UART_STOP_BITS_t, 175
 - UART_TIMEOUT_CYCLE_COUNT, 173
- UART_CFG_t, 23
 - baud_rate, 23
 - clock_polarity, 23
 - data_bits, 23
 - mode, 23
 - mode_type, 23
 - parity, 24
 - stop_bits, 24
- UART_CLOCK_POLARITY_t
 - UART_cfg.h, 173
- UART_DATA_5_BITS
 - UART_cfg.h, 174
- UART_DATA_6_BITS
 - UART_cfg.h, 174
- UART_DATA_7_BITS
 - UART_cfg.h, 174
- UART_DATA_8_BITS
 - UART_cfg.h, 174
- UART_DATA_9_BITS
 - UART_cfg.h, 174
- UART_DATA_BITS_t
 - UART_cfg.h, 174
- UART_FALLING_EDGE_CLOCK
 - UART_cfg.h, 173
- UART_MODE_ASYNCHRONOUS_DOUBLE_SPEED
 - UART_cfg.h, 174
- UART_MODE_ASYNCHRONOUS_NORMAL
 - UART_cfg.h, 174
- UART_MODE_RX
 - UART_cfg.h, 174
- UART_MODE_SYNCHRONOUS_MASTER
 - UART_cfg.h, 174
- UART_MODE_SYNCHRONOUS_SLAVE
 - UART_cfg.h, 174
- UART_MODE_t
 - UART_cfg.h, 174
- UART_MODE_TX
 - UART_cfg.h, 174
- UART_MODE_TX_RX
 - UART_cfg.h, 174
- UART_MODE_TYPE_t
 - UART_cfg.h, 174
- UART_PARITY_DISABLE
 - UART_cfg.h, 174
- UART_PARITY_EVEN
 - UART_cfg.h, 174
- UART_PARITY_ODD
 - UART_cfg.h, 174
- UART_PARITY_t
 - UART_cfg.h, 174
- UART_reg.h
 - DOR, 179
 - FE, 179
 - MPCM, 179
 - RXB8, 179
 - RXC, 179
 - RXCIE, 179
 - RXEN, 179
 - TXB8, 179
 - TXC, 179
 - TXCIE, 179
 - TXEN, 179
 - U2X, 179

- UBRR0H, [177](#)
- UBRR0L, [177](#)
- UBRR1H, [177](#)
- UBRR1L, [177](#)
- UCPOL, [179](#)
- UCSR0A, [177](#)
- UCSR0B, [177](#)
- UCSR0C, [178](#)
- UCSR1A, [178](#)
- UCSR1B, [178](#)
- UCSR1C, [178](#)
- UCSZ0, [179](#)
- UCSZ1, [179](#)
- UCSZ2, [179](#)
- UDR0, [178](#)
- UDR1, [178](#)
- UDRE, [179](#)
- UDRIE, [179](#)
- UMSEL, [179](#)
- UPE, [179](#)
- UPM0, [179](#)
- UPM1, [179](#)
- USBS, [179](#)
- UART_RISING_EDGE_CLOCK
 - UART_cfg.h, [173](#)
- UART_service.c
 - UART0_Receive9BitString, [274](#)
 - UART0_ReceiveString, [274](#)
 - UART0_ReceiveString_Checksum, [275](#)
 - UART0_Send9BitString, [275](#)
 - UART0_Send9BitString_Asynchronous, [276](#)
 - UART0_SendFloat, [276](#)
 - UART0_SendInteger, [276](#)
 - UART0_SendString, [278](#)
 - UART0_SendString_Asynchronous, [278](#)
 - UART0_SendString_Checksum, [279](#)
 - UART1_Receive9BitString, [279](#)
 - UART1_ReceiveString, [280](#)
 - UART1_ReceiveString_Checksum, [280](#)
 - UART1_Send9BitString, [280](#)
 - UART1_Send9BitString_Asynchronous, [281](#)
 - UART1_SendFloat, [281](#)
 - UART1_SendInteger, [282](#)
 - UART1_SendString, [282](#)
 - UART1_SendString_Asynchronous, [283](#)
 - UART1_SendString_Checksum, [283](#)
- UART_service.h
 - UART0_Receive9BitString, [181](#)
 - UART0_ReceiveString, [181](#)
 - UART0_ReceiveString_Checksum, [182](#)
 - UART0_Send9BitString, [182](#)
 - UART0_SendFloat, [183](#)
 - UART0_SendInteger, [183](#)
 - UART0_SendString, [184](#)
 - UART0_SendString_Asynchronous, [184](#)
 - UART0_SendString_Checksum, [185](#)
 - UART1_Receive9BitString, [185](#)
 - UART1_ReceiveString, [186](#)
 - UART1_Send9BitString, [186](#)
 - UART1_SendFloat, [187](#)
 - UART1_SendInteger, [187](#)
 - UART1_SendString, [188](#)
 - UART1_SendString_Asynchronous, [188](#)
- UART_STOP_1_BIT
 - UART_cfg.h, [175](#)
- UART_STOP_2_BIT
 - UART_cfg.h, [175](#)
- UART_STOP_BITS_t
 - UART_cfg.h, [175](#)
- UART_TIMEOUT_CYCLE_COUNT
 - UART_cfg.h, [173](#)
- UBRR0H
 - UART_reg.h, [177](#)
- UBRR0L
 - UART_reg.h, [177](#)
- UBRR1H
 - UART_reg.h, [177](#)
- UBRR1L
 - UART_reg.h, [177](#)
- UCPOL
 - UART_reg.h, [179](#)
- UCSR0A
 - UART_reg.h, [177](#)
- UCSR0B
 - UART_reg.h, [177](#)
- UCSR0C
 - UART_reg.h, [178](#)
- UCSR1A
 - UART_reg.h, [178](#)
- UCSR1B
 - UART_reg.h, [178](#)
- UCSR1C
 - UART_reg.h, [178](#)
- UCSZ0
 - UART_reg.h, [179](#)
- UCSZ1
 - UART_reg.h, [179](#)
- UCSZ2
 - UART_reg.h, [179](#)
- UDR0
 - UART_reg.h, [178](#)
- UDR1
 - UART_reg.h, [178](#)
- UDRE
 - UART_reg.h, [179](#)
- UDRIE
 - UART_reg.h, [179](#)
- UMSEL
 - UART_reg.h, [179](#)
- UPE
 - UART_reg.h, [179](#)
- UPM0
 - UART_reg.h, [179](#)
- UPM1
 - UART_reg.h, [179](#)
- USBS

- UART_reg.h, 179
- V_BIT
 - SREG.h, 128
- W_ACK_PAYLOAD
 - NRF24_reg.h, 111
- W_REGISTER
 - NRF24_reg.h, 111
- W_TX_PAYLOAD
 - NRF24_reg.h, 111
- W_TX_PAYLOAD_NOACK
 - NRF24_reg.h, 111
- waitForClientPickup
 - app.h, 33
- WCOL
 - SPI_reg.h, 126
- WEST
 - app.h, 30
- WGM00
 - TIMER_reg.h, 151
- WGM01
 - TIMER_reg.h, 151
- WGM10
 - TIMER_reg.h, 152
- WGM11
 - TIMER_reg.h, 152
- WGM12
 - TIMER_reg.h, 152
- WGM13
 - TIMER_reg.h, 152
- WGM20
 - TIMER_reg.h, 154
- WGM21
 - TIMER_reg.h, 154
- WGM30
 - TIMER_reg.h, 154
- WGM31
 - TIMER_reg.h, 154
- WGM32
 - TIMER_reg.h, 155
- WGM33
 - TIMER_reg.h, 155
- WHEELS.c
 - WHEELS_Brake, 285
 - WHEELS_Coast, 285
 - WHEELS_GetWheelsPosition, 286
 - WHEELS_GoBackward, 286
 - WHEELS_GoForward, 286
 - WHEELS_Init, 287
 - WHEELS_SetSpeed, 287
 - WHEELS_SetWheelsPosition, 288
 - WHEELS_TurnLeft, 288
 - WHEELS_TurnRight, 288
- WHEELS.h
 - SHARP_TURN, 190
 - SMOOTH_TURN, 190
 - WHEELS_Brake, 190
 - WHEELS_Coast, 191
 - WHEELS_GetCurrentConsumption, 191
 - WHEELS_GetWheelsPosition, 191
 - WHEELS_GoBackward, 192
 - WHEELS_GoForward, 192
 - WHEELS_Init, 192
 - WHEELS_ON_BACK, 190
 - WHEELS_ON_FRONT, 190
 - WHEELS_POSITION_t, 190
 - WHEELS_SetSpeed, 193
 - WHEELS_SetWheelsPosition, 193
 - WHEELS_TURN_t, 190
 - WHEELS_TurnLeft, 194
 - WHEELS_TurnRight, 194
- WHEELS_Brake
 - WHEELS.c, 285
 - WHEELS.h, 190
- WHEELS_cfg.c
 - WHEELS_Config, 290
- WHEELS_cfg.h
 - ROTATE_BACK, 196
 - ROTATE_DIR_t, 195
 - ROTATE_FWD, 196
 - ROTATE_LEFT_SHARP, 196
 - ROTATE_LEFT_SMOOTH, 196
 - ROTATE_RIGHT_SHARP, 196
 - ROTATE_RIGHT_SMOOTH, 196
 - ROTATE_STOP, 196
 - WHEELS_Config, 196
- WHEELS_Coast
 - WHEELS.c, 285
 - WHEELS.h, 191
- WHEELS_Config
 - WHEELS_cfg.c, 290
 - WHEELS_cfg.h, 196
- WHEELS_CONFIG_t, 24
 - CTA_pin, 24
 - CTA_port, 25
 - CTB_pin, 25
 - CTB_port, 25
 - currentSensitivity, 25
 - ENA_pin, 25
 - ENA_port, 25
 - ENB_pin, 25
 - ENB_port, 25
 - IN1A_channel, 26
 - IN1B_channel, 26
 - IN2A_channel, 26
 - IN2B_channel, 26
 - SpeedPercentage, 26
 - WHEELS_Position, 26
- WHEELS_GetCurrentConsumption
 - WHEELS.h, 191
- WHEELS_GetWheelsPosition
 - WHEELS.c, 286
 - WHEELS.h, 191
- WHEELS_GoBackward
 - WHEELS.c, 286
 - WHEELS.h, 192

- WHEELS_GoForward
 - WHEELS.c, [286](#)
 - WHEELS.h, [192](#)
- WHEELS_Init
 - WHEELS.c, [287](#)
 - WHEELS.h, [192](#)
- WHEELS_ON_BACK
 - WHEELS.h, [190](#)
- WHEELS_ON_FRONT
 - WHEELS.h, [190](#)
- WHEELS_Position
 - WHEELS_CONFIG_t, [26](#)
- WHEELS_POSITION_t
 - WHEELS.h, [190](#)
- WHEELS_SetSpeed
 - WHEELS.c, [287](#)
 - WHEELS.h, [193](#)
- WHEELS_SetWheelsPosition
 - WHEELS.c, [288](#)
 - WHEELS.h, [193](#)
- WHEELS_TURN_t
 - WHEELS.h, [190](#)
- WHEELS_TurnLeft
 - WHEELS.c, [288](#)
 - WHEELS.h, [194](#)
- WHEELS_TurnRight
 - WHEELS.c, [288](#)
 - WHEELS.h, [194](#)
- WRITE_PIN
 - DIO.c, [205](#)
- Z_BIT
 - SREG.h, [128](#)