



EMBEDDED SYSTEM DESIGN TECHNIQUES

An API Standard for MCU's

By Jacob Beningo

API Standard for MCUs

Jacob Beningo



Copyright © 2017 by Jacob Beningo

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review or scholarly journal.

The authors and publisher have made their best effort in preparing this book but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein. The advice contained herein may not be suitable for your situation.

It is recommended that you consult a professional where appropriate.

For additional copies or bulk purchases of this book in the U.S. please contact sales@beningo.com.

Beningo Embedded Group
P.O. Box 400
Linden, Mi 48451

www.beningo.com

Introduction

Embedded software has traditionally been developed as a one-off software development effort designed for an individual product. In recent years, embedded system complexity has dramatically increased and the microcontrollers capabilities have followed. What were once simple 8-bit computing machines running at a few dozen megahertz have now become full-fledged 32-bit processors executing at hundreds of megahertz's. Developing software from scratch or for use in a single application or processor has become extremely costly and problematic for design teams.

This API standard for microcontrollers is an example hardware abstraction layer designed to help embedded software developers designing products with microcontrollers create reusable software that abstracts out the hardware. This API standard has been developed and used in production systems for more than half a decade in devices ranging from automotive and medical devices to space systems. Each iteration that it has gone through has helped create a standard that flexible for developers and meets many general real-time design needs.

Using an API to abstract out the microcontroller has several major benefits to development teams such as:

- Removing the specialized need to master the microcontroller hardware
- Decreasing costs through reusable firmware
- Faster times to market
- Better planning and accuracy in the development cycle
- Portability and flexibility to handle numerous applications

Undoubtedly there are many more benefits but in this book the goal is to provide the reader with the standard. If you are interested in understanding how to develop reusable software and the processes that a developer would go through to create their own API's and HAL's, the companion book "Developing Reusable Firmware: A Practical Approach", can be found at www.beningo.com. Developing Reusable Firmware discusses the key ideas behind creating API's and HAL's along with the processes and design considerations that developers need to consider when creating their own.

This standard example has gone through many iterations and has become very stable but there is always an opportunity that changes will be made in the future. In order to stay up to date and receive the latest information and also receive the associated API template files, please visit the associated webpage at <https://www.beningo.com/api/index.php> to sign-up. When you sign-up you will receive Doxygen template source files that layout the entire standard in way that can be easily modified to implement in your own development cycle.

I wish you the best of luck in using this standard and dramatically transforming the way in which you develop and reuse your embedded software.

Best Regards,
Jacob Beningo

About the Author



Jacob Beningo is an embedded software consultant who currently works with clients in more than a dozen countries to dramatically transform their businesses by improving product quality, cost and time to market. He has published more than 200 articles on embedded software development techniques, is a sought-after technical advisor, coach and trainer. He has advised and worked with clients across several industries including automotive, defense, medical and space. He enjoys developing and teaching the latest and greatest real-time software development techniques which are held at conferences such as

Embedded World and the Embedded Systems Conference or online in his workshops and webinars. He blogs for DesignNews.com about embedded system design techniques and challenges. Jacob holds Bachelor's degrees in Electrical Engineering, Physics and Mathematics from Central Michigan University and a Master's degree in Space Systems Engineering from the University of Michigan.

Feel free to contact him at jacob@beningo.com, at his website www.beningo.com, and sign-up for his monthly Embedded Bytes Newsletter [here](#).

Table of Contents

Data Structure Index	2
File Index	3
Data Structure Documentation	5
AdcConfig_t	5
DioConfig_t	8
FlashConfig_t	9
I2CConfig_t	10
I2CTransfer_t	11
McuConfig_t	12
PwmConfig_t	15
SpiConfig_t	16
SpiTransfer_t	18
TmrConfig_t	20
UartBaud_t	22
UartConfig_t	23
WdtConfig_t	25
File Documentation	26
adc.c	26
adc.h	37
adc_cfg.c	44
adc_cfg.h	46
constants.h	52
dio.c	63
dio.h	71
dio_cfg.c	79
dio_cfg.h	81
flash.c	84
flash.h	90
flash_cfg.c	96
flash_cfg.h	98
i2c.c	100
i2c.h	111
i2c_cfg.h	115
isr.c	118
isr.h	124
mcu.c	129
mcu.h	133
mcu_cfg.c	137
mcu_cfg.h	139
pwm.c	143
pwm.h	150
pwm_cfg.c	157
pwm_cfg.h	159
spi.c	161
spi.h	173
spi_cfg.c	180
tmr.c	182
tmr.h	189
tmr_cfg.c	195
tmr_cfg.h	197
uart.c	200
uart.h	209
uart_cfg.c	217

uart_cfg.h	219
wdt.c	222
wdt.h	230
wdt_cfg.c	237
wdt_cfg.h	239
Index	241

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

- [AdcConfig_t](#) 5
- [DioConfig_t](#) 8
- [FlashConfig_t](#) 9
- [I2CConfig_t](#) 10
- [I2CTransfer_t](#) 11
- [McuConfig_t](#) 12
- [PwmConfig_t](#) 15
- [SpiConfig_t](#) 16
- [SpiTransfer_t](#) 18
- [TmrConfig_t](#) 20
- [UartBaud_t](#) 22
- [UartConfig_t](#) 23
- [WdtConfig_t](#) 25

File Index

File List

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

adc.c (The implementation for the adc)	26
adc.h (The interface definition for the adc)	37
adc_cfg.c (This module contains the implementation for the adc peripheral configuration)	44
adc_cfg.h (This module contains the configuration interface for the adc)	46
constants.h (This file contains standard constants used in the application)	52
dio.c (The implementation for the dio)	63
dio.h (The interface definition for the dio)	71
dio_cfg.c (This module contains the implementation for the digital input/output peripheral configuration)	79
dio_cfg.h (This module contains interface definition for the Dio configuration)	81
flash.c (The implementation for the flash)	84
flash.h (The interface definition for the flash)	90
flash_cfg.c (This module contains the flash configuration code)	96
flash_cfg.h (This contains the header for the flash configuration)	98
hal_config.h	Error! Bookmark not defined.
i2c.c (The implementation for the i2c)	100
i2c.h (The interface definition for i2c)	111
i2c_cfg.h (This module contains the configuration interface for i2c)	115
isr.c (The implementation for the interrupts)	118
isr.h (The interface definition for interrupts)	124
mcu.c (The implementation for the mcu)	129
mcu.h (The interface definition for the mcu)	133
mcu_cfg.c (This module contains the configuration for the mcu module)	137
mcu_cfg.h (This module contains the configuration interface for mcu)	139
pwm.c (The implementation for the pwm)	143
pwm.h (The interface definition for the pwm)	150
pwm_cfg.c (This module contains the configuration for the pwm module)	157
pwm_cfg.h (This module contains the configuration interface for pwm)	159
spi.c (The implementation for the spi)	161
spi.h (The interface definition for spi)	173
spi_cfg.c (This module contains the configuration for the spi module)	180
spi_cfg.h	Error! Bookmark not defined.
tmr.c (The implementation for the timer)	182
tmr.h (The interface definition for the timer)	189
tmr_cfg.c (This module contains the configuration for the tmr module)	195
tmr_cfg.h (This module contains the configuration interface for timer)	197
uart.c (The implementation for the uart)	200
uart.h (The interface definition for the uart)	209
uart_cfg.c (This module contains the uart configuration code)	217
uart_cfg.h (This file contains the header definitions for the uart configuration)	219
wdt.c (The implementation for the watchdog)	222
wdt.h (The interface definition for the watchdog)	230
wdt_cfg.c (This module contains the configuration for the wdt module)	237
wdt_cfg.h (This module contains the configuration interface for wdt)	239

Data Structure Documentation

AdcConfig_t Struct Reference

```
#include <adc_cfg.h>
```

Data Fields

- uint8_t [AdcEnable](#)
 - uint8_t [ConvMode](#)
 - uint8_t [Resolution](#)
 - uint8_t [AdcAvg](#)
 - uint8_t [ClkSrc](#)
 - uint8_t [ClkDiv](#)
 - uint8_t [PowerMode](#)
 - uint8_t [SampleTime](#)
 - uint8_t [LngSampleTime](#)
 - uint8_t [IntEnable](#)
 - uint8_t [OperationSpeed](#)
 - uint8_t [DiffMode](#)
 - uint8_t [MuxSelect](#)
 - uint8_t [ConvTrigger](#)
 - uint8_t [VRefSrc](#)
 - uint8_t [CompareMode](#)
 - uint8_t [TriggerSrc](#)
 - uint8_t [Pretrigger](#)
-

Detailed Description

Defines the adc configuration table elements that are used by Adc_Init to configure the adc registers.

Field Documentation

uint8_t AdcConfig_t::AdcEnable

Enable or disable the ADC Module

uint8_t AdcConfig_t::ConvMode

ADC Conversion mode sequence selection

uint8_t AdcConfig_t::Resolution

Select the Adc resolution

uint8_t AdcConfig_t::AdcAvg

Determine how many ADC conversions will be averaged to create the ADC average result

uint8_t AdcConfig_t::ClkSrc

ADC clock source selection

uint8_t AdcConfig_t::ClkDiv

ADC clock source divider

uint8_t AdcConfig_t::PowerMode

Enable or Disable low power mode.

uint8_t AdcConfig_t::SampleTime

Set the Adc sample time to short or long

uint8_t AdcConfig_t::LngSampleTime

Set the Adc long sample time

uint8_t AdcConfig_t::IntEnable

ADC Interrupt Enabled

uint8_t AdcConfig_t::OperationSpeed

Set adc operation speed to normal or high speed conversion

uint8_t AdcConfig_t::DiffMode

Enable or disable differential mode

uint8_t AdcConfig_t::MuxSelect

Set the ADC mux setting to select between ADC channels A or B

uint8_t AdcConfig_t::ConvTrigger

Select ADC conversion trigger mode, software or hardware

uint8_t AdcConfig_t::VRefSrc

Reference voltage source

uint8_t AdcConfig_t::CompareMode

Set the ADC compare mode

uint8_t AdcConfig_t::TriggerSrc

Selects the ADC trigger source

uint8_t AdcConfig_t::Pretrigger

Selects the ADC pretrigger mode

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

- [adc_cfg.h](#)

DioConfig_t Struct Reference

```
#include <dio_cfg.h>
```

Data Fields

- [DioChannel_t Channel](#)
 - [DioResistor_t Resistor](#)
 - DioDirection_t [Direction](#)
 - [DioPinState_t Data](#)
 - [DioMode_t Function](#)
-

Detailed Description

Defines the digital input/output configuration table elements that are used by Dio_Init to configure the Dio peripheral.

Field Documentation

[DioChannel_t](#) DioConfig_t::Channel

The I/O pin

[DioResistor_t](#) DioConfig_t::Resistor

Pullup/Pulldown Resistor - ENABLED or DISABLED

DioDirection_t DioConfig_t::Direction

Data Direction - OUTPUT or INPUT

[DioPinState_t](#) DioConfig_t::Data

Data - HIGH or LOW

[DioMode_t](#) DioConfig_t::Function

Mux Function - Dio_Peri_Select

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

- [dio_cfg.h](#)

FlashConfig_t Struct Reference

```
#include <flash_cfg.h>
```

Data Fields

- uint8_t [ClkSrc](#)
- uint8_t [ClkDiv](#)

Detailed Description

Struct Flash_ConfigType The Flash Config Type structure is used to set the configuration for the Flash module.

Field Documentation

uint8_t FlashConfig_t::ClkSrc

The Flash clock source

uint8_t FlashConfig_t::ClkDiv

Set the Clock divider to 1-64.

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

- [flash_cfg.h](#)

I2CConfig_t Struct Reference

```
#include <i2c_cfg.h>
```

Data Fields

- uint8_t [I2C_Channel](#)
 - uint8_t [ChannelEnable](#)
 - uint8_t [Mode](#)
 - uint8_t [AddrType](#)
 - uint16_t [I2CId](#)
 - uint32_t [BaudRate](#)
-

Detailed Description

Struct I2C_ConfigType The I2C Config Type structure is used to set the configuration for the I2C peripheral.

Field Documentation

uint8_t I2CConfig_t::I2C_Channel

The I2C Channel Name

uint8_t I2CConfig_t::ChannelEnable

Enable or Disable the I2C channel

uint8_t I2CConfig_t::Mode

Enable or Disable the I2C channel

uint8_t I2CConfig_t::AddrType

Used to set the address type to either 7 bit or 10 bit

uint16_t I2CConfig_t::I2CId

The Id of the microcontroller

uint32_t I2CConfig_t::BaudRate

The baud rate to run the communication at

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

- [i2c_cfg.h](#)

I2CTransfer_t Struct Reference

```
#include <i2c.h>
```

Data Fields

- uint8_t [Channel](#)
 - uint8_t [SlaveId](#)
 - uint8_t * [NumBytes](#)
 - uint8_t * [DataBuf](#)
 - uint8_t [Mode](#)
-

Detailed Description

Struct I2C_TransferType The I2C Transfer Type structure is used to set the configuration for transmitting and receiving I2C data to and from a slave device. The number of bytes is defined as a pointer so that a variable can be assigned to the NumBytes slot so that the number of bytes read or written can be changed on the fly in code without having to specify multiple TransferTypes. If this is not done then the compiler gives L1907 fixup overflow error.

Field Documentation

uint8_t I2CTransfer_t::Channel

The I2C channel being used

uint8_t I2CTransfer_t::SlaveId

The Id of the slave device to communicate with

uint8_t* I2CTransfer_t::NumBytes

The number of bytes to transmit/receive

uint8_t* I2CTransfer_t::DataBuf

The pointer to the data buffer

uint8_t I2CTransfer_t::Mode

Transmit(0) or Receive(1)

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

- [i2c.h](#)

McuConfig_t Struct Reference

```
#include <mcu_cfg.h>
```

Data Fields

- uint8_t [ClockMode](#)
- uint8_t [FLLDiv](#)
- uint8_t [FLLFrequency](#)
- uint8_t [ExRefSrc](#)
- uint8_t [ExRefStopMode](#)
- uint8_t [PLLDiv](#)
- uint8_t [PLLStopEnable](#)
- uint8_t [PLL_VCO_Div](#)
- uint8_t [IntClkEnable](#)
- uint8_t [IntRefStopMode](#)
- uint8_t [IntRefClkSelec](#)
- uint8_t [IntFastClkDiv](#)
- uint8_t [OscillatorMode](#)
- uint8_t [Osc2P](#)
- uint8_t [Osc4P](#)
- uint8_t [Osc8P](#)
- uint8_t [Osc16P](#)
- uint8_t [SysClkDiv](#)
- uint8_t [BusClkDiv](#)

Detailed Description

Defines the mcu configuration table elements that are used by Mcu_Init to configure the clock registers.

Field Documentation

uint8_t McuConfig_t::ClockMode

Select the MCG mode of operation.

uint8_t McuConfig_t::FLLDiv

Selects the amount to divide down the external reference clock for the FLL. The resulting frequency must be in the range 31.25 kHz to 39.0625 kHz

uint8_t McuConfig_t::FLLFrequency

Selects the DCO Frequency Range of the FLL output

uint8_t McuConfig_t::ExRefSrc

Selects the source for the external reference clock.

uint8_t McuConfig_t::ExRefStopMode

Enable or disable the external reference clock when MCU enters Stop mode.

uint8_t McuConfig_t::PLLDiv

Selects the amount to divide the external reference clock for the PLL. The resulting frequency must be in the range 2 MHz to 4 MHz

uint8_t McuConfig_t::PLLStopEnable

Enable or disable the PLL clock when MCU enters Stop mode.

uint8_t McuConfig_t::PLL_VCO_Div

Selects the amount to multiply the VCO output of the PLL.

uint8_t McuConfig_t::IntClkEnable

Enables the internal reference clock for use as MCGIRCLK.

uint8_t McuConfig_t::IntRefStopMode

Enable or disable the internal reference clock when MCU enters Stop mode.

uint8_t McuConfig_t::IntRefClkSelec

Selects between the fast or slow internal reference clock source.

uint8_t McuConfig_t::IntFastClkDiv

Selects the amount to divide down the fast internal reference clock. The resulting frequency will be in the range 31.25 kHz to 4 MHz

uint8_t McuConfig_t::OscillatorMode

Controls the crystal oscillator mode of operation.

uint8_t McuConfig_t::Osc2P

Add 2pF capacitor to the oscillator load.

uint8_t McuConfig_t::Osc4P

Add 4pF capacitor to the oscillator load.

uint8_t McuConfig_t::Osc8P

Add 8pF capacitor to the oscillator load.

uint8_t McuConfig_t::Osc16P

Add 16pF capacitor to the oscillator load.

uint8_t McuConfig_t::SysClkDiv

Sets the divide value for the core/system clock

uint8_t McuConfig_t::BusClkDiv

Sets the divide value for the bus and flash clock and is in addition to the System clock divide ratio.

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

- [mcu_cfg.h](#)

PwmConfig_t Struct Reference

```
#include <pwm_cfg.h>
```

Data Fields

- uint8_t [ChannelName](#)
 - uint8_t [PwmEnable](#)
 - uint8_t [IntEnable](#)
 - uint16_t [DutyCycle](#)
-

Detailed Description

Defines the Pwm configuration structure

Field Documentation

uint8_t PwmConfig_t::ChannelName

The name of the pwm channel

uint8_t PwmConfig_t::PwmEnable

Defines the mode of the pwm channel

uint8_t PwmConfig_t::IntEnable

Enable or Disable the capture/compare interrupt

uint16_t PwmConfig_t::DutyCycle

Defines the duty cycle of the pwm channel

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

- [pwm_cfg.h](#)

SpiConfig_t Struct Reference

```
#include <spi_cfg.h>
```

Data Fields

- uint8_t [ChannelName](#)
- uint8_t [SpiEnable](#)
- uint8_t [MasterMode](#)
- uint8_t [SSPinMode](#)
- uint8_t [Bidirection](#)
- uint8_t [WaitMode](#)
- uint32_t [BaudRate](#)

Detailed Description

Defines the configuration data required to initialize the SPI peripheral.

Field Documentation

uint8_t SpiConfig_t::ChannelName

Defines the name of the SPI channel

uint8_t SpiConfig_t::SpiEnable

ENABLE or DISABLE the SPI channel

uint8_t SpiConfig_t::MasterMode

Defines the peripheral Master/Slave mode

uint8_t SpiConfig_t::SSPinMode

Defines the slave select pin function

uint8_t SpiConfig_t::Bidirection

Bidirectional mode output enable

uint8_t SpiConfig_t::WaitMode

Enable or disable operation when CPU is in wait mode

uint32_t SpiConfig_t::BaudRate

Defines the baud rate in Hz

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

- spi_cfg.h

SpiTransfer_t Struct Reference

#include <spi.h>

Data Fields

- SpiChannel_t [SpiChannel](#)
- DioChannel_t [ChipSelect](#)
- SpiChipSelect_t [CsPolarity](#)
- uint16_t [NumBytes](#)
- uint8_t * [TxRxData](#)
- SpiPolarity_t [Polarity](#)
- SpiPhase_t [Phase](#)
- SpiBitOrder_t [Direction](#)

Detailed Description

The SPI Transfer Type structure is used to set the configuration for transmitting SPI data.

Field Documentation

SpiChannel_t SpiTransfer_t::SpiChannel

The SPI channel to be used

[DioChannel_t](#) SpiTransfer_t::ChipSelect

The DIO channel to be used for chip select

[SpiChipSelect_t](#) SpiTransfer_t::CsPolarity

The active state of chip select

uint16_t SpiTransfer_t::NumBytes

The number of bytes to send

uint8_t* SpiTransfer_t::TxRxData

Pointer to the data to transfer

[SpiPolarity_t](#) SpiTransfer_t::Polarity

Transfer data polarity

[SpiPhase_t](#) SpiTransfer_t::Phase

Transfer data phase

[SpiBitOrder_t](#) SpiTransfer_t::Direction

Bit direction

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

- [spi.h](#)

TmrConfig_t Struct Reference

```
#include <tmr_cfg.h>
```

Data Fields

- uint16_t [TimerChannel](#)
- uint16_t [TimerEnable](#)
- uint16_t [TimerMode](#)
- uint16_t [ClockSource](#)
- uint16_t [ClkMode](#)
- uint16_t [ClkPrescaler](#)
- uint16_t [IntEnabled](#)
- uint16_t [IntPriority](#)
- uint16_t [Interval](#)

Detailed Description

Defines the timer configuration table elements that are used by Tmr_Init to configure the timer registers.

Field Documentation

- uint16_t TmrConfig_t::TimerChannel**
Name of Timer
- uint16_t TmrConfig_t::TimerEnable**
ENABLED or DISABLED Timer channel
- uint16_t TmrConfig_t::TimerMode**
Timer Counter Mode Setting
- uint16_t TmrConfig_t::ClockSource**
Timer Clock Source Setting
- uint16_t TmrConfig_t::ClkMode**
Timer Clock Mode Select
- uint16_t TmrConfig_t::ClkPrescaler**
Clock input divider
- uint16_t TmrConfig_t::IntEnabled**
Timer Interrupt Enable - ENABLED or DISABLED
- uint16_t TmrConfig_t::IntPriority**
Timer Interrupt Priority
- uint16_t TmrConfig_t::Interval**
Timer interval in microseconds

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

- [tmr_cfg.h](#)

UartBaud_t Struct Reference

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

Data Fields

- uint32_t [ClkFreq](#)
- uint32_t [BaudRate](#)
- uint8_t [UCBRx](#)
- uint8_t [UCBRSx](#)
- uint8_t [UCBRFx](#)
- uint8_t [Oversampling](#)

Detailed Description

Defines the uart configuration table elements that are used by Uart_Init to configure the uart registers.

Field Documentation

uint32_t UartBaud_t::ClkFreq
System Clock Frequency

uint32_t UartBaud_t::BaudRate
Desired Baud Rate

uint8_t UartBaud_t::UCBRx
Value of UCBRx register bits

uint8_t UartBaud_t::UCBRSx
Value of UCBRSx register bits

uint8_t UartBaud_t::UCBRFx
Value of UCBRFx register bits

uint8_t UartBaud_t::Oversampling
Oversampling mode ENABLED or DISABLED

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

- [uart.h](#)

UartConfig_t Struct Reference

```
#include <uart_cfg.h>
```

Data Fields

- uint8_t [UartChannel](#)
- uint8_t [UartEnable](#)
- uint8_t [UartMode](#)
- uint8_t [ClkSrc](#)
- uint32_t [BaudRate](#)
- uint8_t [Loopback](#)
- uint8_t [BitDirection](#)
- uint8_t [DataLength](#)
- uint8_t [StopBits](#)
- uint8_t [ParityType](#)
- uint8_t [Delimiter](#)
- uint8_t [IntEnable](#)

Detailed Description

Defines the uart configuration table elements that are used by Uart_Init to configure the uart registers.

Field Documentation

uint8_t UartConfig_t::UartChannel

Name of UART

uint8_t UartConfig_t::UartEnable

Uart Enable - ENABLED or DISABLED

uint8_t UartConfig_t::UartMode

Uart Mode Selection

uint8_t UartConfig_t::ClkSrc

Slect the clock source for BRCLK

uint32_t UartConfig_t::BaudRate

Uart channel baud rate

uint8_t UartConfig_t::Loopback

ENABLE or DISABLE loopback mode

uint8_t UartConfig_t::BitDirection

Rx and Tx Shift register bit direction

uint8_t UartConfig_t::DataLength

Character length, 8 or 9 bits

uint8_t UartConfig_t::StopBits

Specify the number of stop bits, one or two

uint8_t UartConfig_t::ParityType

Parity Selection. EVEN, ODD, or DISABLED

uint8_t UartConfig_t::Delimiter

Break/synch delimiter length.

uint8_t UartConfig_t::IntEnable

Uart Receive Interrupt Enable - ENABLED or DISABLED

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

- [uart_cfg.h](#)

WdtConfig_t Struct Reference

```
#include <hal_config.h>
```

Data Fields

- uint8_t [WdtMode](#)
- uint8_t [ClockSource](#)
- uint8_t [Interval](#)

Detailed Description

Defines the timer configuration table elements that are used by Tmr_Init to configure the timer registers.

Field Documentation

uint8_t WdtConfig_t::WdtMode

Watchdog Timer Mode - RESET or INTERVAL

uint8_t WdtConfig_t::ClockSource

Timer Clock Source Select

uint8_t WdtConfig_t::Interval

Watchdog timer interval selection

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

- `hal_config.h`
- [wdt_cfg.h](#)

File Documentation

adc.c File Reference

The implementation for the adc.

```
#include <stdint.h>
#include "adc.h"
#include "constants.h"
#include "hal_config.h"
```

Functions

- [AdcError_t Adc_Calibrate](#) (void)
- [AdcError_t Adc_CompareSet](#) (uint8_t CompareMode)
- [AdcError_t Adc_AverageSet](#) (uint8_t AdcAvg)
- [AdcError_t Adc_PretriggerSet](#) (uint8_t Pretrigger)
- [AdcError_t Adc_Init](#) ([AdcConfig_t](#) const *const Config)
- [AdcError_t Adc_PowerDown](#) (void)
- [AdcError_t Adc_StartConversion](#) ([AdcChannel_t](#) const Channel)
- [AdcError_t Adc_EndConversion](#) (void)
- uint16_t [Adc_ResultGet](#) (void)
- void [Adc_RegisterWrite](#) (TYPE const Address, TYPE const Value)
- TYPE [Adc_RegisterRead](#) (TYPE const Address)
- void [Adc_CallbackRegister](#) ([AdcCallback_t](#) const Function, TYPE(*CallbackFunction)(type))

Function Documentation

[AdcError_t Adc_Calibrate](#) (void) [[inline](#)]

Description:

This function is used to calibrate the ADC module.

PRE-CONDITION: The [Adc_Init](#) function must be called with valid configuration data. PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The adc will calibrated and the offset register updated

Returns:

None

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_Calibrate();
5  Adc_StartConversion(ADC0);
6  ... // Delay
7  Adc_EndConversion();
8  Result = Adc_ResultGet()
```

See also:

[Adc_ConfigGet](#)

[Adc_Init](#)

[Adc_PowerDown](#)

[Adc_StartConversion](#)

[Adc_EndConversion](#)
[Adc_ResultGet](#)
[Adc_RegisterWrite](#)
[Adc_RegisterRead](#)
[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[AdcError_t](#) **Adc_CompareSet** (uint8_t const *CompareMode*) [inline]

Description:

This function is used to set the ADC compare mode.

PRE-CONDITION: The `Adc_Init` function must be called with valid configuration data. PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The adc will be in compare mode.

Parameters:

in	<i>CompareMode</i>	- uint8_t, value indicating Compare Mode of ADC module.
----	--------------------	---

Returns:

None

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_Calibrate();
5  Adc_StartConversion(ADC0);
6  ... // Delay
7  Adc_EndConversion();
8  Result = Adc_ResultGet()
```

See also:

[Adc_ConfigGet](#)
[Adc_Init](#)
[Adc_PowerDown](#)
[Adc_StartConversion](#)
[Adc_EndConversion](#)
[Adc_ResultGet](#)
[Adc_RegisterWrite](#)
[Adc_RegisterRead](#)
[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

--	--	--	--

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[AdcError_t](#) Adc_AverageSet (uint8_t const *AdcAvg*)[inline]

Description:

This function is used to set the number of ADC samples to average in hardware.

PRE-CONDITION: The Adc_Init function must be called with valid configuration data.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The adc will be set the average the configured number of samples.

Parameters:

in	<i>AdcAvg</i>	- uint8_t, value indicating the number of samples to average.
----	---------------	---

Returns:

None

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_Calibrate();
5  Adc_AverageSet(Config->AdcAvg);
6  Adc_StartConversion(ADC0);
7  ... // Delay
8  Adc_EndConversion();
9  Result = Adc_ResultGet()
```

See also:

- [Adc_ConfigGet](#)
- [Adc_Init](#)
- [Adc_PowerDown](#)
- [Adc_StartConversion](#)
- [Adc_EndConversion](#)
- [Adc_ResultGet](#)
- [Adc_RegisterWrite](#)
- [Adc_RegisterRead](#)
- [Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[AdcError_t](#) Adc_PretriggerSet (uint8_t const *Pretrigger*) [inline]

Description:

This function is used to set the pretrigger mode of the ADC module.

PRE-CONDITION: The Adc_Init function must be called with valid configuration data.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The adc pretrigger will be set to the configured value.

Parameters:

in	<i>Pretrigger</i>	- uint8_t, pretrigger mode value.
----	-------------------	-----------------------------------

Returns:

None

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_Calibrate();
5  Adc_AverageSet(Config->AdcAvg);
6  Adc_StartConversion(ADC0);
7  ... // Delay
8  Adc_EndConversion();
9  Result = Adc_ResultGet()
```

See also:

- [Adc_ConfigGet](#)
- [Adc_Init](#)
- [Adc_PowerDown](#)
- [Adc_StartConversion](#)
- [Adc_EndConversion](#)
- [Adc_ResultGet](#)
- [Adc_RegisterWrite](#)
- [Adc_RegisterRead](#)
- [Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[AdcError_t](#) Adc_Init ([AdcConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the adc based on the configuration tables that are in adc_cfg.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The Adc peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3 Adc_Init(AdcConfig);
```

See also:

[Adc_ConfigGet](#)

[Adc_Init](#)

[Adc_PowerDown](#)

[Adc_StartConversion](#)

[Adc_EndConversion](#)

[Adc_ResultGet](#)

[Adc_RegisterWrite](#)

[Adc_RegisterRead](#)

[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[AdcError_t](#) **Adc_PowerDown (void)**

Description:

This function is used to power down the adc peripheral.

PRE-CONDITION: The Adc_Init function must be called with valid configuration data. PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The Adc peripheral will be powered down.

Returns:

void

Example:

```
1 const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3 Adc_Init(AdcConfig);
4 Adc_PowerDown();
```

See also:

[Adc_ConfigGet](#)
[Adc_Init](#)
[Adc_PowerDown](#)
[Adc_StartConversion](#)
[Adc_EndConversion](#)
[Adc_ResultGet](#)
[Adc_RegisterWrite](#)
[Adc_RegisterRead](#)
[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[AdcError_t](#) Adc_StartConversion ([AdcChannel_t](#) const *Channel*)

Description:

This function is used to manually start an adc sample.

PRE-CONDITION: The Adc_Init function must be called with valid configuration data. PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The Adc peripheral will start converting an adc channel

Parameters:

<i>Channel</i>	- AdcChannel_t, the channel to enable
----------------	---------------------------------------

Returns:

void

Example:

```

1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_StartConversion(ADC0);

```

See also:

[Adc_ConfigGet](#)
[Adc_Init](#)
[Adc_PowerDown](#)
[Adc_StartConversion](#)
[Adc_EndConversion](#)
[Adc_ResultGet](#)
[Adc_RegisterWrite](#)
[Adc_RegisterRead](#)
[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[AdcError_t](#) Adc_EndConversion (void)

Description:

This function is used to manually end an adc sample.

PRE-CONDITION: The Adc_Init function must be called with valid configuration data. PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The adc must be sampling.

POST-CONDITION: The Adc peripheral will stop converting the adc channel

Returns:

void

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_StartConversion(ADC0);
5  ... // Delay
6  Adc_EndConversion();
```

See also:

[Adc_ConfigGet](#)

[Adc_Init](#)

[Adc_PowerDown](#)

[Adc_StartConversion](#)

[Adc_EndConversion](#)

[Adc_ResultGet](#)

[Adc_RegisterWrite](#)

[Adc_RegisterRead](#)

[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint16_t Adc_ResultGet (void)

Description:

This function is used to retrieve the adc result from memory.

PRE-CONDITION: The Adc_Init function must be called with valid configuration data.

PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The adc must be sampled.

POST-CONDITION: A valid adc reading will be returned.

Returns:

uint16_t, adc value of most recent sample.

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_StartConversion(ADC0);
5  ... // Delay
6  Adc_EndConversion();
7  Result = Adc_ResultGet()
```

See also:

- [Adc_ConfigGet](#)
- [Adc_Init](#)
- [Adc_PowerDown](#)
- [Adc_StartConversion](#)
- [Adc_EndConversion](#)
- [Adc_ResultGet](#)
- [Adc_RegisterWrite](#)
- [Adc_RegisterRead](#)
- [Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Adc_RegisterWrite (TYPE const Address, TYPE const Value)

Description:

This function is used to directly address and modify an Adc register. The function should be used to access specialized functionality in the Adc peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Adc register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Adc peripheral map
in	<i>Value</i>	is the value to set the Adc register to

Returns:

void

Example:

```
1 Adc_RegisterWrite(0x1000, 0x15);
```

See also:

[Adc_ConfigGet](#)

[Adc_Init](#)

[Adc_PowerDown](#)

[Adc_StartConversion](#)

[Adc_EndConversion](#)

[Adc_ResultGet](#)

[Adc_RegisterWrite](#)

[Adc_RegisterRead](#)

[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

TYPE Adc_RegisterRead (TYPE const *Address*)

Description:

This function is used to directly address an Adc register. The function should be used to access specialized functionality in the Adc peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Adc register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Adc register to read
----	----------------	--

Returns:

The current value of the Adc register.

Example:

```
1 AdcValue = Adc_RegisterRead(0x1000);
```

See also:

[Adc_ConfigGet](#)
[Adc_Init](#)
[Adc_PowerDown](#)
[Adc_StartConversion](#)
[Adc_EndConversion](#)
[Adc_ResultGet](#)
[Adc_RegisterWrite](#)
[Adc_RegisterRead](#)
[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Adc_CallbackRegister (AdcCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the adc driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The AdcCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 AdcCallback_t Adc_Function = ADC_SAMPLE_COMPLETE;  
2  
3 Adc_CallbackRegister(Adc_Function, Adc_SampleAverage);
```

See also:

[Adc_ConfigGet](#)
[Adc_Init](#)
[Adc_PowerDown](#)
[Adc_StartConversion](#)
[Adc_EndConversion](#)
[Adc_ResultGet](#)
[Adc_RegisterWrite](#)
[Adc_RegisterRead](#)
[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

adc.h File Reference

The interface definition for the adc.

```
#include "adc_cfg.h"
```

Macros

- #define [Adc_GetAdcDoneFlag\(\)](#)

Enumerations

- enum [AdcError_t](#)

Functions

- [AdcError_t Adc_PowerDown](#) (void)
- [AdcError_t Adc_StartConversion](#) ([AdcChannel_t](#) const Channel)
- [AdcError_t Adc_EndConversion](#) (void)
- uint16_t [Adc_ResultGet](#) (void)
- void [Adc_RegisterWrite](#) (TYPE const Address, TYPE const Value)
- TYPE [Adc_RegisterRead](#) (TYPE Address)
- void [Adc_CallbackRegister](#) (AdcCallback_t const Function, TYPE(*CallbackFunction)(type))

Detailed Description

This is the header file for the definition of the interface for the analog to digital convert.

Macro Definition Documentation

```
#define Adc_GetAdcDoneFlag()
```

Returns the value of the Adc conversion flag. 1 - Still converting 0 - Conversion complete

Enumeration Type Documentation

```
enum AdcError\_t
```

Defines the possible errors that can be returned by the adc driver

Function Documentation

```
AdcError\_t Adc_PowerDown (void )
```

Description:

This function is used to power down the adc peripheral.

PRE-CONDITION: The Adc_Init function must be called with valid configuration data.
PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The Adc peripheral will be powered down.

Returns:

void

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_PowerDown();
```

See also:

[Adc_ConfigGet](#)

[Adc_Init](#)

[Adc_PowerDown](#)

[Adc_StartConversion](#)

[Adc_EndConversion](#)

[Adc_ResultGet](#)

[Adc_RegisterWrite](#)

[Adc_RegisterRead](#)

[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[AdcError_t](#) **Adc_StartConversion** ([AdcChannel_t](#) const *Channel*)

Description:

This function is used to manually start an adc sample.

PRE-CONDITION: The `Adc_Init` function must be called with valid configuration data. PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The Adc peripheral will start converting an adc channel

Parameters:

<i>Channel</i>	- <code>AdcChannel_t</code> , the channel to enable
----------------	---

Returns:

void

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_StartConversion(ADC0);
```

See also:

[Adc_ConfigGet](#)

[Adc_Init](#)

[Adc_PowerDown](#)

[Adc_StartConversion](#)

[Adc_EndConversion](#)

[Adc_ResultGet](#)
[Adc_RegisterWrite](#)
[Adc_RegisterRead](#)
[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[AdcError_t](#) Adc_EndConversion (void)

Description:

This function is used to manually end an adc sample.

PRE-CONDITION: The Adc_Init function must be called with valid configuration data. PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The adc must be sampling.

POST-CONDITION: The Adc peripheral will stop converting the adc channel

Returns:

void

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_StartConversion(ADC0);
5  ... // Delay
6  Adc_EndConversion();
```

See also:

[Adc_ConfigGet](#)
[Adc_Init](#)
[Adc_PowerDown](#)
[Adc_StartConversion](#)
[Adc_EndConversion](#)
[Adc_ResultGet](#)
[Adc_RegisterWrite](#)
[Adc_RegisterRead](#)
[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint16_t Adc_ResultGet (void)

Description:

This function is used to retrieve the adc result from memory.

PRE-CONDITION: The Adc_Init function must be called with valid configuration data. PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The adc must be sampled.

POST-CONDITION: A valid adc reading will be returned.

Returns:

uint16_t, adc value of most recent sample.

Example:

```
1  const AdcConfig_t *AdcConfig = Adc_ConfigGet();
2
3  Adc_Init(AdcConfig);
4  Adc_StartConversion(ADC0);
5  ... // Delay
6  Adc_EndConversion();
7  Result = Adc_ResultGet()
```

See also:

- [Adc_ConfigGet](#)
- [Adc_Init](#)
- [Adc_PowerDown](#)
- [Adc_StartConversion](#)
- [Adc_EndConversion](#)
- [Adc_ResultGet](#)
- [Adc_RegisterWrite](#)
- [Adc_RegisterRead](#)
- [Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Adc_RegisterWrite (TYPE const Address, TYPE const Value)

Description:

This function is used to directly address and modify an Adc register. The function should be used to access specialized functionality in the Adc peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Adc register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Adc peripheral map
in	<i>Value</i>	is the value to set the Adc register to

Returns:

void

Example:

```
1 Adc_RegisterWrite(0x1000, 0x15);
```

See also:

[Adc_ConfigGet](#)

[Adc_Init](#)

[Adc_PowerDown](#)

[Adc_StartConversion](#)

[Adc_EndConversion](#)

[Adc_ResultGet](#)

[Adc_RegisterWrite](#)

[Adc_RegisterRead](#)

[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

TYPE Adc_RegisterRead (TYPE const *Address*)

Description:

This function is used to directly address an Adc register. The function should be used to access specialized functionality in the Adc peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Adc register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Adc register to read
----	----------------	--

Returns:

The current value of the Adc register.

Example:

```
1 AdcValue = Adc_RegisterRead(0x1000);
```

See also:

[Adc_ConfigGet](#)

[Adc_Init](#)

[Adc_PowerDown](#)

[Adc_StartConversion](#)

[Adc_EndConversion](#)

[Adc_ResultGet](#)

[Adc_RegisterWrite](#)

[Adc_RegisterRead](#)

[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Adc_CallbackRegister (AdcCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the adc driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The AdcCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 AdcCallback_t Adc_Function = ADC_SAMPLE_COMPLETE;  
2  
3 Adc_CallbackRegister(Adc_Function, Adc_SampleAverage);
```

See also:

[Adc_ConfigGet](#)

[Adc_Init](#)

[Adc_PowerDown](#)

[Adc_StartConversion](#)

[Adc_EndConversion](#)
[Adc_ResultGet](#)
[Adc_RegisterWrite](#)
[Adc_RegisterRead](#)
[Adc_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

adc_cfg.c File Reference

This module contains the implementation for the adc peripheral configuration.

```
#include "adc_cfg.h"
#include "constants.h"
```

Functions

- [AdcConfig_t](#) const *const [Adc_ConfigGet](#) (void)

Variables

- const [AdcConfig_t](#) [AdcConfig](#)

Detailed Description

This module contains the configuration for the i2c module.

Function Documentation

[AdcConfig_t](#) const* const [Adc_ConfigGet](#) (void)

Description:

This function return a pointer to the Adc configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const AdcConfig_t * AdcConfig = Adc_ConfigGet();
2
3 Adc_Init(AdcConfig);
```

See also:

- [Adc_ConfigGet](#)
- [Adc_Init](#)
- [Adc_PowerDown](#)
- [Adc_StartConversion](#)
- [Adc_EndConversion](#)
- [Adc_ResultGet](#)
- [Adc_RegisterWrite](#)
- [Adc_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Variable Documentation

const [AdcConfig_t](#) AdcConfig

```
Initial value:=
{
    ENABLED,
    SINGLE,
    12\_BIT\_SINGLE,
    NO\_AVG,
    BUS\_CLK,
    CLKDIV\_1,
    DISABLED,
    SHORT\_SAMPLE,
    X6,
    DISABLED,
    NORMAL\_SPEED,
    DISABLED,
    CH\_A,
    SOFTWARE,
    V\_REF,
    DISABLED,
    LPTMR0,
    DISABLED,
}
```

The Adc configuration settings to initialize the adc registers.

adc_cfg.h File Reference

This module contains the configuration interface for the adc.

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

Data Structures

- struct [AdcConfig_t](#)

Macros

- #define [NUM_CAL_REGISTERS](#) 8
- #define [COMPARE_VAL_1](#) 0
- #define [COMPARE_VAL_2](#) 0

Enumerations

- enum [AdcLongTime_t](#) { [X20](#), [X12](#), [X6](#), [X2](#) }
- enum [AdcVRefSrc_t](#) { [V_REF](#), [V_ALT](#) }
- enum [AdcClockSrc_t](#) { [BUS_CLK](#), [BUS_DIV2](#), [ALT_CLK](#), [ASYNC](#) }
- enum [AdcMode_t](#) { [SINGLE](#), [CONTINUOUS](#) }
- enum [AdcAvg_t](#) { [NO_AVG](#), [AVG_4](#), [AVG_8](#), [AVG_16](#), [AVG_32](#) }
- enum [AdcDiv_t](#) { [CLKDIV_1](#), [CLKDIV_2](#), [CLKDIV_4](#), [CLKDIV_8](#) }
- enum [AdcSampleTime_t](#) { [SHORT_SAMPLE](#), [LONG_SAMPLE](#) }
- enum [AdcResolution_t](#) { [_8_BIT_SINGLE](#), [_12_BIT_SINGLE](#), [_10_BIT_SINGLE](#), [_16_BIT_SINGLE](#), [_9_BIT_DIFF](#) = 0, [_13_BITT_DIFF](#), [_11_BITT_DIFF](#), [_16_BIT_DIFF](#) }
- enum [AdcConvSpeed_t](#) { [NORMAL_SPEED](#), [HIGH_SPEED](#) }
- enum [AdcMux_t](#) { [CH_A](#), [CH_B](#) }
- enum [AdcCompare_t](#) { [GREATER_THAN_BOTH](#) = 1, [LESS_THAN_BOTH](#), [GREATER_THAN_SINGLE](#), [LESS_THAN_SINGLE](#) }
- enum [AdcPretrigger_t](#) { [ENABLE_A](#) = 1, [ENABLE_B](#) = 2 }
- enum [AdcTrigger_t](#) { [EXT_TRIG](#) = 0, [CMP0](#) = 1, [PIT0](#) = 4, [PIT1](#) = 5, [TPM0](#) = 8, [TPM1](#) = 9, [TPM2](#) = 10, [RTC_ALARM](#) = 12, [RTC_SEC](#) = 13, [LPTMR0](#) = 14 }
- enum [AdcSampleMode_t](#) { [SOFTWARE](#), [HARDWARE](#) }
- enum [AdcChannel_t](#) { [AD0](#) = 0, [AD1](#) = 1, [AD2](#) = 2, [AD3](#) = 3, [AD4](#) = 4, [AD5](#) = 5, [AD6](#) = 6, [AD7](#) = 7, [AD8](#) = 8, [AD9](#) = 9, [AD10](#) = 10, [AD11](#) = 11, [AD12](#) = 12, [AD13](#) = 13, [AD14](#) = 14, [AD15](#) = 15, [AD16](#) = 16, [AD17](#) = 17, [AD18](#) = 18, [AD19](#) = 19, [AD20](#) = 20, [AD21](#) = 21, [AD22](#) = 22, [AD23](#) = 23, [TEMP](#) = 26, [BANDGAP](#) = 27, [V_REFSH](#) = 29, [V_REFSL](#) = 30, [ADC_DISABLE](#) = 31, [MAX_ADC_CHANNELS](#) = 32 }

Functions

- [AdcConfig_t](#) const *const [Adc_ConfigGet](#) (void)

Macro Definition Documentation

#define NUM_CAL_REGISTERS 8

Defines the number of adc calibration registers

#define COMPARE_VAL_1 0

Defines the ADC compare value 1

#define COMPARE_VAL_2 0

Defines the ADC compare value 2

Enumeration Type Documentation

enum [AdcLongTime_t](#)

Defines the Adc Long sample time.

Enumerator

X20 20 extra ADCK cycles

X12 12 extra ADCK cycles

X6 6 extra ADCK cycles

X2 2 extra ADCK cycles

enum [AdcVRefSrc_t](#)

Defines the ADC voltage reference source selections.

Enumerator

V_REF External Vref pins

V_ALT Alternate Valt pins

enum [AdcClockSrc_t](#)

Defines the ADC clock source selection.

Enumerator

BUS_CLK Bus clock

BUS_DIV2 Bus clock/2

ALT_CLK Alternate clock

ASYNC Asynchronous clock

enum [AdcMode_t](#)

This enumeration defines the conversion sequence modes.

Enumerator

SINGLE One conversion or one set of conversions if the hardware average function is enabled

CONTINUOUS Continuous conversions or sets of conversions if the hardware average function is enabled

enum [AdcAvg_t](#)

This enumeration defines the number of ADC conversions that will be averaged to create the ADC average result.

Enumerator

NO_AVG Hardware averaging is disabled

AVG_4 4 samples averaged.

AVG_8 8 samples averaged.

AVG_16 16 samples averaged.

AVG_32 32 samples averaged.

enum [AdcDiv_t](#)

This enumeration defines the divide ratios used by the ADC to generate the internal clock ADCK

Enumerator

CLKDIV_1 Divide input clock by 1

CLKDIV_2 Divide input clock by 2

CLKDIV_4 Divide input clock by 4

CLKDIV_8 Divide input clock by 8

enum [**AdcSampleTime_t**](#)

This enumeration defines the sample time configurations

Enumerator

SHORT_SAMPLE Divide input clock by 1

LONG_SAMPLE Divide input clock by 2

enum [**AdcResolution_t**](#)

This enumeration defines the Adc resolution options

Enumerator

_8_BIT_SINGLE 8 bit resolution

_12_BIT_SINGLE 12 bit resolution

_10_BIT_SINGLE 10 bit resolution

_16_BIT_SINGLE 16 bit resolution

_9_BIT_DIFF 9 bit resolution, when differential mode is selected

_13_BIT_DIFF 13 bit resolution, when differential mode is selected

_11_BIT_DIFF 11 bit resolution, when differential mode is selected

_16_BIT_DIFF 16 bit resolution, when differential mode is selected

enum [**AdcConvSpeed_t**](#)

This enumeration defines the operation speed modes

Enumerator

NORMAL_SPEED Normal conversion sequence selected.

HIGH_SPEED High-speed conversion sequence selected

enum [**AdcMux_t**](#)

This enumeration defines the ADC mux settings

Enumerator

CH_A Select the ADC A channel set

CH_B Select the ADC B channel set

enum [**AdcCompare_t**](#)

This enumeration defines the ADC compare function modes

Enumerator

GREATER_THAN_BOTH Configures greater than or equal to threshold, both CV1 and CV2 are compared

LESS_THAN_BOTH Configures less than threshold, both CV1 and CV2 are compared

GREATER_THAN_SINGLE Configures greater than or equal to threshold, only CV1 compared

LESS_THAN_SINGLE Configures less than threshold, only CV1 compared

enum [***AdcPretrigger_t***](#)

This enumeration defines the ADC pretrigger modes

Enumerator

ENABLE_A Enable Alternative pretrigger A

ENABLE_B Enable Alternative pretrigger B

enum [***AdcTrigger_t***](#)

This enumeration defines the ADC trigger sources

Enumerator

EXT_TRIG External trigger pin input (EXTRG_IN)

CMP0 CMP0 output

PIT0 PIT Trigger 0

PIT1 PIT trigger 1

TPM0 TPM0 overflow

TPM1 TPM1 overflow

TPM2 TPM2 overflow

RTC_ALARM RTC alarm

RTC_SEC RTC seconds

LPTMR0 LPTMR0 trigger

enum [***AdcSampleMode_t***](#)

Defines the Adc sample trigger modes

Enumerator

SOFTWARE Sample triggered by software

HARDWARE Sample triggered by hardware

enum [***AdcChannel_t***](#)

This enumeration defines a list of the adc channels.

Enumerator

AD0 DADP0 or DAD0

AD1 DADP1 or DAD1

AD2 DADP2 or DAD2

AD3 DADP3 or DAD3

AD4 AN4

AD5 AN5

AD6 AN6

AD7 AN7

AD8 AN4

AD9 AN5

AD10 AN6

AD11 AN7

AD12 AN4

AD13 AN5

AD14 AN6

AD15 AN7

AD16 AN4

AD17 AN5

AD18 AN6

AD19 AN7

AD20 AN4

AD21 AN5

AD22 AN6

AD23 AN7

TEMP Temperature Sensor

BANDGAP Bandgap (single-ended or differential)

V_REFSH VREFSH is selected as input

V_REFSL VREFSL is selected as input

ADC_DISABLE ADC Module is disabled.

MAX_ADC_CHANNELS Maximum ADC Channel

Function Documentation

[AdcConfig_t](#) const* const Adc_ConfigGet (void)

Description:

This function return a pointer to the Adc configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const AdcConfig_t * AdcConfig = Adc_ConfigGet();
2
3 Adc_Init(AdcConfig);
```


See also:[Adc_ConfigGet](#)[Adc_Init](#)[Adc_PowerDown](#)[Adc_StartConversion](#)[Adc_EndConversion](#)[Adc_ResultGet](#)[Adc_RegisterWrite](#)[Adc_RegisterRead](#)**- HISTORY OF CHANGES -**

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

constants.h File Reference

This file contains standard constants used in the application.

Macros

- #define [ZERO](#) (0U)
- #define [null](#) (NULL)
- #define [PI](#) (3.1415927)
- #define [TWO_PI](#) (6.2831854)
- #define [HALF_PI](#) (1.57079)
- #define [EPSILON](#) (0.0001)
- #define [DEG_TO_RAD](#) (0.01745329)
- #define [RAD_TO_DEG](#) (57.2957786)
- #define [REGBIT0](#) (1UL)
- #define [REGBIT1](#) (1UL<<1U)
- #define [REGBIT2](#) (1UL<<2U)
- #define [REGBIT3](#) (1UL<<3U)
- #define [REGBIT4](#) (1UL<<4U)
- #define [REGBIT5](#) (1UL<<5U)
- #define [REGBIT6](#) (1UL<<6U)
- #define [REGBIT7](#) (1UL<<7U)
- #define [REGBIT8](#) (1UL<<8U)
- #define [REGBIT9](#) (1UL<<9U)
- #define [REGBIT10](#) (1UL<<10U)
- #define [REGBIT11](#) (1UL<<11U)
- #define [REGBIT12](#) (1UL<<12U)
- #define [REGBIT13](#) (1UL<<13U)
- #define [REGBIT14](#) (1UL<<14U)
- #define [REGBIT15](#) (1UL<<15U)
- #define [REGBIT16](#) (1UL<<16U)
- #define [REGBIT17](#) (1UL<<17U)
- #define [REGBIT18](#) (1UL<<18U)
- #define [REGBIT19](#) (1UL<<19U)
- #define [REGBIT20](#) (1UL<<20U)
- #define [REGBIT21](#) (1UL<<21U)
- #define [REGBIT22](#) (1UL<<22U)
- #define [REGBIT23](#) (1UL<<23U)
- #define [REGBIT24](#) (1UL<<24U)
- #define [REGBIT25](#) (1UL<<25U)
- #define [REGBIT26](#) (1UL<<26U)
- #define [REGBIT27](#) (1UL<<27U)
- #define [REGBIT28](#) (1UL<<28U)
- #define [REGBIT29](#) (1UL<<29U)
- #define [REGBIT30](#) (1UL<<30U)
- #define [REGBIT31](#) (1UL<<31U)
- #define [CLOCK_1_MHZ](#) (1000000UL)
- #define [CLOCK_2_MHZ](#) (2000000UL)
- #define [CLOCK_4_MHZ](#) (4000000UL)
- #define [CLOCK_7_37_MHZ](#) (7370000UL)
- #define [CLOCK_8_MHZ](#) (8000000UL)
- #define [CLOCK_12_MHZ](#) (12000000UL)
- #define [CLOCK_16_MHZ](#) (16000000UL)
- #define [CLOCK_24_MHZ](#) (24000000UL)

- #define [CLOCK_32_MHZ](#) (32000000UL)
- #define [CLOCK_40_MHZ](#) (40000000UL)
- #define [CLOCK_48_MHZ](#) (48000000UL)
- #define [CLEAR_FLAG_POS](#) (TRUE)
- #define [CLEAR_FLAG_NUL](#) (FALSE)
- #define [Abs](#)(x) ((x)>0?(x):- (x))
- #define [Constrain](#)(amt, low, high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
- #define [Degrees](#)(rad) ((rad)*[RAD_TO_DEG](#))
- #define [Max](#)(a, b) ((a)>(b)?(a):(b))
- #define [Min](#)(a, b) ((a)<(b)?(a):(b))
- #define [Round](#)(x) ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))
- #define [Radians](#)(deg) ((deg)*[DEG_TO_RAD](#))
- #define [Sq](#)(x) ((x)*(x))
- #define [bit](#)(x) (1UL<<(x))
- #define [setBits](#)(x, y) ((x)|=(y))
- #define [clearBits](#)(x, y) ((x)&=~(y))
- #define [bitRead](#)(x, y) (((x)>>(y)) & 1)
- #define [bitWrite](#)(x, y, v) ((v) ? bitSet((x), (y)) : bitClear((x), (y)))
- #define [lowByte](#)(x) ((x) & 0x00ff)
- #define [highByte](#)(x) ((x)>>8)

Enumerations

- enum [PinLevelEnum_t](#) { [LOW](#) = 0x0U, [HIGH](#) = 0x1U, [ACTIVE_LOW](#) = 0x0U, [INACTIVE_HIGH](#) = 0x1U, [INACTIVE_LOW](#) = 0x0U, [ACTIVE_HIGH](#) = 0x1U }
- enum [LogicEnum_t](#) { [DISABLED](#) = 0U, [ENABLED](#) = 1U, [FALSE](#) = 0U, [TRUE](#) = 1U }
- enum [OnOff_t](#) { [OFF](#) = 0U, [ON](#) = 1U }
- enum [PinModeEnum_t](#) { [OUTPUT](#) = 0x0U, [INPUT](#) = 0x1U }
- enum [NumberBase_t](#) { [DEC](#) = 10U, [HEX](#) = 16U, [OCT](#) = 8U, [BIN](#) = 2U }
- enum [Byte_t](#) { [ONE_BYTES](#) = 1U, [TWO_BYTES](#) = 2U, [THREE_BYTES](#) = 3U, [FOUR_BYTES](#) = 4U, [FIVE_BYTES](#) = 5U, [SIX_BYTES](#) = 6U, [SEVEN_BYTES](#) = 7U, [EIGHT_BYTES](#) = 8U, [NINE_BYTES](#) = 9U }
- enum [ClockDivide_t](#) { [CLOCK_DIVIDEBY_1](#) = 1, [CLOCK_DIVIDEBY_2](#) = 2, [CLOCK_DIVIDEBY_4](#) = 4, [CLOCK_DIVIDEBY_8](#) = 8, [CLOCK_DIVIDEBY_16](#) = 16, [CLOCK_DIVIDEBY_32](#) = 32, [CLOCK_DIVIDEBY_64](#) = 64, [CLOCK_DIVIDEBY_128](#) = 128, [CLOCK_DIVIDEBY_256](#) = 256 }
- enum [ClockPeriod_t](#) { [CLOCK_PERIOD_NS_1_MHZ](#) = 1000UL, [CLOCK_PERIOD_NS_2_MHZ](#) = 500UL, [CLOCK_PERIOD_NS_4_MHZ](#) = 250UL, [CLOCK_PERIOD_NS_8_MHZ](#) = 125UL, [CLOCK_PERIOD_NS_16_MHZ](#) = 62UL, [CLOCK_PERIOD_NS_24_MHZ](#) = 42UL, [CLOCK_PERIOD_NS_32_MHZ](#) = 31UL, [CLOCK_PERIOD_NS_40_MHZ](#) = 25UL, [CLOCK_PERIOD_NS_48_MHZ](#) = 21UL }
- enum [BitShift_t](#) { [ZERO_BITS](#), [ONE_BIT](#), [TWO_BITS](#), [THREE_BITS](#), [FOUR_BITS](#), [FIVE_BITS](#), [SIX_BITS](#), [SEVEN_BITS](#), [EIGHT_BITS](#), [NINE_BITS](#), [TEN_BITS](#), [ELEVEN_BITS](#), [TWELVE_BITS](#), [THIRTEEN_BITS](#), [FOURTEEN_BITS](#), [FIFTEEN_BITS](#), [SIXTEEN_BITS](#), [SEVENTEEN_BITS](#), [EIGHTEEN_BITS](#), [NINETEEN_BITS](#), [TWENTY_BITS](#), [TWENTYONE_BITS](#), [TWENTYTWO_BITS](#), [TWENTYTHREE_BITS](#), [TWENTYFOUR_BITS](#), [TWENTYFIVE_BITS](#), [TWENTYSIX_BITS](#), [TWENTYSEVEN_BITS](#), [TWENTYEIGHT_BITS](#), [TWENTYNINE_BITS](#), [THIRTY_BITS](#), [THIRTYONE_BITS](#), [THIRTYTWO_BITS](#) }

Variables

- typedef [num](#)

Macro Definition Documentation

#define ZERO (0U)

Constants: Zero Used to define a constant value of zero.

#define null (NULL)

Constants: null Is used to define both null and NULL keywords.

#define PI (3.1415927)

Engineering Constant, Pi

#define TWO_PI (6.2831854)

Engineering Constant, Pi*2

#define HALF_PI (1.57079)

Engineering Constant, Pi/2

#define EPSILON (0.0001)

Engineering Constant, epsilon

#define DEG_TO_RAD (0.01745329)

Engineering Constant, converting degrees to radians

#define RAD_TO_DEG (57.2957786)

Engineering Constant, converting radians to degrees

#define REGBIT0 (1UL)

Register Bit 0 shift

#define REGBIT1 (1UL<<1U)

Register Bit 1 shift

#define REGBIT2 (1UL<<2U)

Register Bit 2 shift

#define REGBIT3 (1UL<<3U)

Register Bit 3 shift

#define REGBIT4 (1UL<<4U)

Register Bit 4 shift

#define REGBIT5 (1UL<<5U)

Register Bit 5 shift

#define REGBIT6 (1UL<<6U)

Register Bit 6 shift

#define REGBIT7 (1UL<<7U)

Register Bit 7 shift

#define REGBIT8 (1UL<<8U)

Register Bit 8 shift

#define REGBIT9 (1UL<<9U)

Register Bit 9 shift

#define REGBIT10 (1UL<<10U)

Register Bit 10 shift

#define REGBIT11 (1UL<<11U)

Register Bit 11 shift

#define REGBIT12 (1UL<<12U)

Register Bit 12 shift

#define REGBIT13 (1UL<<13U)

Register Bit 13 shift

#define REGBIT14 (1UL<<14U)

Register Bit 14 shift

#define REGBIT15 (1UL<<15U)

Register Bit 15 shift

#define REGBIT16 (1UL<<16U)

Register Bit 16 shift

#define REGBIT17 (1UL<<17U)

Register Bit 17 shift

#define REGBIT18 (1UL<<18U)

Register Bit 18 shift

#define REGBIT19 (1UL<<19U)

Register Bit 19 shift

#define REGBIT20 (1UL<<20U)

Register Bit 20 shift

#define REGBIT21 (1UL<<21U)

Register Bit 21 shift

#define REGBIT22 (1UL<<22U)

Register Bit 22 shift

#define REGBIT23 (1UL<<23U)

Register Bit 23 shift

#define REGBIT24 (1UL<<24U)

Register Bit 24 shift

#define REGBIT25 (1UL<<25U)

Register Bit 25 shift

#define REGBIT26 (1UL<<26U)

Register Bit 26 shift

#define REGBIT27 (1UL<<27U)

Register Bit 27 shift

#define REGBIT28 (1UL<<28U)

Register Bit 28 shift

#define REGBIT29 (1UL<<29U)

Register Bit 29 shift

#define REGBIT30 (1UL<<30U)

Register Bit 30 shift

#define REGBIT31 (1UL<<31U)

Register Bit 31 shift

#define CLOCK_1_MHZ (1000000UL)

Defines the unsigned long value for 1 MHz clock frequency.

#define CLOCK_2_MHZ (2000000UL)

Defines the unsigned long value for 2 MHz clock frequency.

#define CLOCK_4_MHZ (4000000UL)

Defines the unsigned long value for 4 MHz clock frequency.

#define CLOCK_7_37_MHZ (7370000UL)

Defines the unsigned long value for 7.37 MHz clock frequency.

#define CLOCK_8_MHZ (8000000UL)

Defines the unsigned long value for 8 MHz clock frequency.

#define CLOCK_12_MHZ (12000000UL)

Defines the unsigned long value for 12 MHz clock frequency.

#define CLOCK_16_MHZ (16000000UL)

Defines the unsigned long value for 16 MHz clock frequency.

#define CLOCK_24_MHZ (24000000UL)

Defines the unsigned long value for 24 MHz clock frequency.

#define CLOCK_32_MHZ (32000000UL)

Defines the unsigned long value for 32 MHz clock frequency.

#define CLOCK_40_MHZ (40000000UL)

Defines the unsigned long value for 40 MHz clock frequency.

#define CLOCK_48_MHZ (48000000UL)

Defines the unsigned long value for 48 MHz clock frequency.

#define CLEAR_FLAG_POS ([TRUE](#))

Clear a flag to 1

#define CLEAR_FLAG_NUL ([FALSE](#))

Clear a flag to 0

#define Abs(x) ((x)>0?(x):- (x))

Macro: Abs Returns the absolute value of a number.

Parameters: n - the number

Returns: n - if is greater than or equal to 0 -n - if x is less than zero

Example:

: // Take the absolute value of the number : abs = [Abs\(num\)](#);

#define Constrain(amt, low, high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))

Macro: Constrain Constrains a number to be within a range.

Parameters: amt - the number to constrain, all data types low - the lower end of the range, all data types high - the upper end of the range, all data types

Returns: amt - if amt is between a and b low - if amt is less than low high - if amt is greater than high

Example:

: // limits range of sensor values to between 10 and 150 : sensVal = [Constrain\(sensVal, 10, 150\)](#);

Related: Min, Max

#define Degrees(rad) ((rad)*[RAD_TO_DEG](#))

Macro: Degrees This macro converts radians to degrees.

#define Max(a, b) ((a)>(b)?(a):(b))

Macro: Max Returns the larger of two numbers.

Parameters: a - the first number, any data types b - the second number, any data types

Returns: The larger of the two numbers.

Example:

Related: Min, Constrain

#define Min(a, b) ((a)<(b)?(a):(b))

Macro: Min Returns the smaller of two numbers.

Parameters: a - the first number, any data types b - the second number, any data types

Returns: The smaller of the two numbers.

Example:

Related: Max, Constrain

#define Round(x) ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))

Macro: Round This macro rounds a float to a integer.

#define Radians(deg) ((deg)*[DEG_TO_RAD](#))

Macro: Radians This macro converts degrees to radians.

#define Sq(x) ((x)*(x))

Macro: Sq Returns the square a number.

#define bit(x) (1UL<<(x))

Macro: Bit Manipulation - Access specified bit

#define setBits(x, y) ((x)|=(y))

Macro: Bit Manipulation - Set the specified bit

#define clearBits(x, y) ((x)&=~(y))

Macro: Bit Manipulation - Clears the bit

#define bitRead(x, y) (((x)>>(y)) & 1)

Macro: Bit Manipulation - Read a bit

#define bitWrite(x, y, v) ((v) ? bitSet((x), (y)) : bitClear((x), (y)))

Macro: Bit Manipulation - Write a bit

#define lowByte(x) ((x) & 0x00ff)

Macro: Bit Manipulation - Access lower byte

#define highByte(x) ((x)>>8)

Macro: Bit Manipulation - Access the top byte of a 16 bit number

Enumeration Type Documentation

enum [PinLevelEnum_t](#)

Constants: I/O Pin Levels

Enumerator

LOW Reserved word representing the logical value 0 (OFF, 0 volts)

HIGH Reserved word representing the logical value 1 (ON, 5 volts)

ACTIVE_LOW Reserved word representing the logical value 0 (OFF, 0 volts)

INACTIVE_HIGH Reserved word representing the logical value 1 (ON, 5 volts)

INACTIVE_LOW Reserved word representing the logical value 0 (OFF, 0 volts)

ACTIVE_HIGH Reserved word representing the logical value 1 (ON, 5 volts)

enum [LogicEnum_t](#)

Constants: Logic Values

Enumerator

DISABLED Reserved word for representing the logical value 0 (OFF, 0 volts)

ENABLED Reserved word for representing the logical value 1 (ON, 5 volts)

FALSE Reserved word for representing the logical value 0 (OFF, 0 volts)

TRUE Reserved word for representing the logical value 1 (ON, 5 volts)

enum [OnOff_t](#)

Constants: ON/OFF States

Enumerator

OFF Reserved word for representing the logical value 0 (OFF, 0 volts)

ON Reserved word for representing the logical value 1 (ON, 5 volts)

enum [PinModeEnum_t](#)

Constants: I/O Pin Modes

Enumerator

OUTPUT Reserved word representing the mode of an I/O pin or an I/O port as an input

INPUT Reserved word representing the mode of an I/O pin or an I/O port as an output

enum [NumberBase_t](#)

Constants: Number Base

Enumerator

DEC Defines the base 10 decimal numbering system

HEX Defines the base 16 hexadecimal numbering system

OCT Defines the base 8 octal numbering system

BIN Defines the base 2 binary numbering system

enum [Byte_t](#)

Constants: Byte Numbering Used to represent number of bytes. ONE_BYTE, TWO_BYTES, ... NINE_BYTES

Enumerator

ONE_BYTES One byte

TWO_BYTES Two byte

THREE_BYTES Three byte

FOUR_BYTES Four byte

FIVE_BYTES Five byte

SIX_BYTES Six byte

SEVEN_BYTES Seven byte

EIGHT_BYTES Eight byte

NINE_BYTES Nine byte

enum [ClockDivide_t](#)

Constants: CLOCK_DIVIDEBY_X Clock Prescaler Values.

Enumerator

CLOCK_DIVIDEBY_1 Divide clock by 1

CLOCK_DIVIDEBY_2 Divide clock by 2

CLOCK_DIVIDEBY_4 Divide clock by 4

CLOCK_DIVIDEBY_8 Divide clock by 8

CLOCK_DIVIDEBY_16 Divide clock by 16

CLOCK_DIVIDEBY_32 Divide clock by 32

CLOCK_DIVIDEBY_64 Divide clock by 64

CLOCK_DIVIDEBY_128 Divide clock by 128

CLOCK_DIVIDEBY_256 Divide clock by 256

enum [ClockPeriod_t](#)

Constants: **CLOCK_PERIOD_NS_X_MHZ** Defines the unsigned long value of the clock period in nanoseconds.

Enumerator

CLOCK_PERIOD_NS_1_MHZ 1 MHz clock period

CLOCK_PERIOD_NS_2_MHZ 2 MHz clock period

CLOCK_PERIOD_NS_4_MHZ 4 MHz clock period

CLOCK_PERIOD_NS_8_MHZ 8 MHz clock period

CLOCK_PERIOD_NS_16_MHZ 16 MHz clock period

CLOCK_PERIOD_NS_24_MHZ 24 MHz clock period

CLOCK_PERIOD_NS_32_MHZ 32 MHz clock period

CLOCK_PERIOD_NS_40_MHZ 40 MHz clock period

CLOCK_PERIOD_NS_48_MHZ 48 MHz clock period

enum [BitShift_t](#)

Enum: Number of bits to shift

Enumerator

ZERO_BITS Shift 0 bytes

ONE_BIT Shift 1 bytes

TWO_BITS Shift 2 bytes

THREE_BITS Shift 3 bytes

FOUR_BITS Shift 4 bytes

FIVE_BITS Shift 5 bytes

SIX_BITS Shift 6 bytes

SEVEN_BITS Shift 7 bytes

EIGHT_BITS Shift 8 bytes

NINE_BITS Shift 9 bytes

TEN_BITS Shift 10 bytes

ELEVEN_BITS Shift 11 bytes

TWELVE_BITS Shift 12 bytes

THIRTEEN_BITS Shift 13 bytes

FOURTEEN_BITS Shift 14 bytes

FIFTEEN_BITS Shift 15 bytes

SIXTEEN_BITS Shift 16 bytes

SEVENTEEN_BITS Shift 17 bytes

EIGHTEEN_BITS Shift 18 bytes

NINETEEN_BITS Shift 19 bytes

TWENTY_BITS Shift 20 bytes

TWENTYONE_BITS Shift 21 bytes

TWENTYTWO_BITS Shift 22 bytes

TWENTYTHREE_BITS Shift 23 bytes

TWENTYFOUR_BITS Shift 24 bytes

TWENTYFIVE_BITS Shift 25 bytes

TWENTYSIX_BITS Shift 26 bytes

TWENTYSEVEN_BITS Shift 27 bytes

TWENTYEIGHT_BITS Shift 28 bytes

TWENTYNINE_BITS Shift 29 bytes

THIRTY_BITS Shift 30 bytes

THIRTYONE_BITS Shift 31 bytes

THIRTYTWO_BITS Shift 32 bytes

Variable Documentation

typedef num

```
Initial value:{  
    PERCENT_0 = 0,  
    PERCENT_100 = 100  
}Percent_t
```

Constants: Percentages General percent constants.PERCENT_0 is 0% PERCENT_100 is 100%

dio.c File Reference

The implementation for the dio.

```
#include "dio.h"
#include <xxx.h>
```

Functions

- void [Dio_Init](#) ([DioConfig_t](#) const *const *Config*)
- [DioPinState_t](#) [Dio_ChannelRead](#) ([DioChannel_t](#) const Channel)
- void [Dio_ChannelWrite](#) ([DioChannel_t](#) const Channel, [DioPinState_t](#) const State)
- void [Dio_ChannelToggle](#) ([DioChannel_t](#) const Channel)
- void [Dio_ChannelModeSet](#) ([DioChannel_t](#) const Channel, [DioMode_t](#) const Mode)
- void [Dio_ChannelDirectionSet](#) ([DioChannel_t](#) const Channel, [PinModeEnum_t](#) const Mode)
- void [Dio_RegisterWrite](#) (uint32_t const Address, TYPE const Value)
- TYPE [Dio_RegisterRead](#) (uint32_t const Address)
- void [Dio_CallbackRegister](#) (DioCallback_t const Function, TYPE(*CallbackFunction)(type))

Function Documentation

void [Dio_Init](#) ([DioConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the Dio based on the configuration table defined in dio_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: NUMBER_OF_CHANNELS_PER_PORT > 0

PRE-CONDITION: NUMBER_OF_PORTS > 0

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The DIO peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const DioConfig_t *DioConfig = Dio_ConfigGet();
2
3 Dio_Init(DioConfig);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

[DioPinState_t](#) Dio_ChannelRead ([DioChannel_t](#) const *Channel*)

Description:

This function is used to read the state of a dio channel (pin)

PRE-CONDITION: The channel is configured as INPUT

PRE-CONDITION: The channel is configured as GPIO

PRE-CONDITION: The channel is within the maximum DioChannel_t definition

POST-CONDITION: The channel state is returned

Parameters:

in	<i>Channel</i>	is the DioChannel_t that represents a pin
----	----------------	---

Returns:

The state of the channel as HIGH or LOW

Example:

```
1 uint8_t pin = Dio_ReadChannel(PORT1_0);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_ChannelWrite ([DioChannel_t](#) const *Channel*, [DioPinState_t](#) const *State*)

Description:

This function is used to write the state of a channel (pin) as either logic high or low through the use of the DioChannel_t enum to select the channel and the DioPinState_t to define the desired state.

PRE-CONDITION: The channel is configured as OUTPUT

PRE-CONDITION: The channel is configured as GPIO

PRE-CONDITION: The channel is within the maximum DioChannel_t definition

POST-CONDITION: The channel state will be State

Parameters:

in	<i>Channel</i>	is the pin to write using the DioChannel_t enum definition
in	<i>State</i>	is HIGH or LOW as defined in the DioPinState_t enum

Returns:

void

Example:

```
1 Dio_WriteChannel(PORT1_0, LOW);           // Set the PORT1_0 pin low
2 Dio_WriteChannel(PORT1_0, HIGH);         // Set the PORT1_0 pin high
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_ChannelToggle ([DioChannel_t](#) const *Channel*)

Description:

This function is used to toggle the current state of a channel (pin).

PRE-CONDITION: The channel is configured as OUTPUT

PRE-CONDITION: The channel is configured as GPIO

PRE-CONDITION: The channel is within the maximum DioChannel_t definition

POST-CONDITION:

Parameters:

in	<i>Channel</i>	is the pin from the DioChannel_t that is to be modified.
----	----------------	--

Returns:

void

Example:

```
1 Dio_ChannelToggle(PORTA_1);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_ChannelModeSet ([DioChannel_t](#) const *Channel*, [DioMode_t](#) const *Mode*)

Description:

This function is used to set the mode of an individual channel (pin). The mode is defined by the DioMode_t enum. The valid channels (pins) are defined in the DioChannel_t enum.

PRE-CONDITION: The channel is within the maximum DioChannel_t definition
PRE-CONDITION: The mode is within the maximum DioMode_t

POST-CONDITION: The channel function will be updated to Mode

Parameters:

in	<i>Channel</i>	is the pin from DioChannel_t that is to have its function changed
in	<i>Mode</i>	is the mode that the pin be multiplexed into. i.e. SPI, UART, etc

Returns:

void

Example:

```
1 Dio_ChannelModeSet(PORTA_1, GPIO);  
2 Dio_ChannelModeSet(PORTA_2, SPI_MOSI);
```

See also:

[Dio_Init](#)
[Dio_ChannelRead](#)
[Dio_ChannelWrite](#)
[Dio_ChannelToggle](#)
[Dio_ChannelModeSet](#)
[Dio_ChannelDirectionSet](#)
[Dio_RegisterWrite](#)
[Dio_RegisterRead](#)
[Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_ChannelDirectionSet ([DioChannel_t](#) const *Channel*, [PinModeEnum_t](#) const *Mode*)

Description:

This function is used to set the direction of an individual channel (pin). The direction is defined by the PinModeEnum_t enum. The valid channels (pins) are defined in the DioChannel_t enum.

PRE-CONDITION: The channel is within the maximum DioChannel_t definition
 PRE-CONDITION: The pin direction is within the maximum PinModeEnum_t

PRE-CONDITION: The pin is mode is set to GPIO

POST-CONDITION: The channel direction will be updated to PinMode

Parameters:

in	<i>Channel</i>	is the pin from DioChannel_t that is to have its function changed
in	<i>Mode</i>	is the mode that the pin be multiplexed into. i.e. SPI, UART, etc

Returns:

void

Example:

```

1 Dio_ChannelDirectionSet(PORTA_1, OUTPUT);
2 Dio_ChannelDirectionSet(PORTA_1, INPUT);

```

See also:

[Dio_Init](#)
[Dio_ChannelRead](#)
[Dio_ChannelWrite](#)
[Dio_ChannelToggle](#)
[Dio_ChannelModeSet](#)
[Dio_ChannelDirectionSet](#)
[Dio_RegisterWrite](#)
[Dio_RegisterRead](#)
[Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
10/22/2015	0.6.0	JWB	Merged into HAL
11/10/2015	1.0.0	JWB	Interface Created

void Dio_RegisterWrite (uint32_t const *Address*, TYPE const *Value*)

Description:

This function is used to directly address and modify a Dio register. The function should be used to access specialised functionality in the Dio peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Dio register addresss space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Dio peripheral map
in	<i>Value</i>	is the value to set the Dio register to

Returns:

void

Example:

```
1 Dio_RegisterWrite(0x1000, 0x15);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

TYPE Dio_RegisterRead (uint32_t const Address)

Description:

This function is used to directly address a Dio register. The function should be used to access specialised functionality in the Dio peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Dio register addresss space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	Address	is the address of the Dio register to read
----	---------	--

Returns:

The current value of the Dio register.

Example:

```
1 DioValue = Dio_RegisterRead(0x1000);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_CallbackRegister (DioCallback_t const Function, TYPE(*) (type) CallbackFunction)

Description:

This function is used to set the callback functions of the dio driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The DioCallback_t has been populated PRE-CONDITION: The callback

function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 DioCallback_t Dio_Function = DIO_SAMPLE_COMPLETE;  
2  
3 Dio_CallbackRegister(Dio_Function, Dio_SampleAverage);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

dio.h File Reference

The interface definition for the dio.

```
#include <stdint.h>
#include "dio_cfg.h"
#include "constants.h"
```

Functions

- void [Dio_Init](#) ([DioConfig_t](#) const *const *Config*)
- [DioPinState_t](#) [Dio_ChannelRead](#) ([DioChannel_t](#) const Channel)
- void [Dio_ChannelWrite](#) ([DioChannel_t](#) const Channel, [DioPinState_t](#) const State)
- void [Dio_ChannelToggle](#) ([DioChannel_t](#) const Channel)
- void [Dio_ChannelModeSet](#) ([DioChannel_t](#) const Channel, [DioMode_t](#) const Mode)
- void [Dio_ChannelDirectionSet](#) ([DioChannel_t](#) const Channel, [PinModeEnum_t](#) const Mode)
- void [Dio_RegisterWrite](#) (uint32_t const Address, TYPE const Value)
- TYPE [Dio_RegisterRead](#) (uint32_t const Address)
- void [Dio_CallbackRegister](#) (DioCallback_t const Function, TYPE(*CallbackFunction)(type))

Detailed Description

This is the header file for the definition of the interface for a digital input / output peripheral on a standard microcontroller.

Function Documentation

void Dio_Init ([DioConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the Dio based on the configuration table defined in dio_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: NUMBER_OF_CHANNELS_PER_PORT > 0

PRE-CONDITION: NUMBER_OF_PORTS > 0

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The DIO peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const DioConfig_t *DioConfig = Dio_ConfigGet();
2
3 Dio_Init(DioConfig);
```

See also:

[Dio_Init](#)

[Dio_ChannelRead](#)
[Dio_ChannelWrite](#)
[Dio_ChannelToggle](#)
[Dio_ChannelModeSet](#)
[Dio_ChannelDirectionSet](#)
[Dio_RegisterWrite](#)
[Dio_RegisterRead](#)
[Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

[DioPinState_t](#) Dio_ChannelRead ([DioChannel_t](#) const *Channel*)

Description:

This function is used to read the state of a dio channel (pin)

PRE-CONDITION: The channel is configured as INPUT

PRE-CONDITION: The channel is configured as GPIO

PRE-CONDITION: The channel is within the maximum DioChannel_t definition

POST-CONDITION: The channel state is returned

Parameters:

in	<i>Channel</i>	is the DioChannel_t that represents a pin
----	----------------	---

Returns:

The state of the channel as HIGH or LOW

Example:

```
1 uint8_t pin = Dio_ReadChannel(PORT1_0);
```

See also:

[Dio_Init](#)
[Dio_ChannelRead](#)
[Dio_ChannelWrite](#)
[Dio_ChannelToggle](#)
[Dio_ChannelModeSet](#)
[Dio_ChannelDirectionSet](#)
[Dio_RegisterWrite](#)
[Dio_RegisterRead](#)
[Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description

09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_ChannelWrite ([DioChannel_t](#) const *Channel*, [DioPinState_t](#) const *State*)

Description:

This function is used to write the state of a channel (pin) as either logic high or low through the use of the DioChannel_t enum to select the channel and the DioPinState_t to define the desired state.

PRE-CONDITION: The channel is configured as OUTPUT

PRE-CONDITION: The channel is configured as GPIO

PRE-CONDITION: The channel is within the maximum DioChannel_t definition

POST-CONDITION: The channel state will be State

Parameters:

in	<i>Channel</i>	is the pin to write using the DioChannel_t enum definition
in	<i>State</i>	is HIGH or LOW as defined in the DioPinState_t enum

Returns:

void

Example:

```
1 Dio_WriteChannel(PORT1_0, LOW);           // Set the PORT1_0 pin low
2 Dio_WriteChannel(PORT1_0, HIGH);          // Set the PORT1_0 pin high
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_ChannelToggle ([DioChannel_t](#) const *Channel*)

Description:

This function is used to toggle the current state of a channel (pin).

PRE-CONDITION: The channel is configured as OUTPUT

PRE-CONDITION: The channel is configured as GPIO

PRE-CONDITION: The channel is within the maximum DioChannel_t definition

POST-CONDITION:

Parameters:

in	<i>Channel</i>	is the pin from the DioChannel_t that is to be modified.
----	----------------	--

Returns:

void

Example:

```
1 Dio_ChannelToggle(PORTA_1);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_ChannelModeSet ([DioChannel_t](#) const *Channel*, [DioMode_t](#) const *Mode*)

Description:

This function is used to set the mode of an individual channel (pin). The mode is defined by the DioMode_t enum. The valid channels (pins) are defined in the DioChannel_t enum.

PRE-CONDITION: The channel is within the maximum DioChannel_t definition

PRE-CONDITION: The mode is within the maximum DioMode_t

POST-CONDITION: The channel function will be updated to Mode

Parameters:

in	<i>Channel</i>	is the pin from DioChannel_t that is to have its function changed
----	----------------	---

in	Mode	is the mode that the pin be multiplexed into. i.e. SPI, UART, etc
----	------	---

Returns:

void

Example:

```
1 Dio_ChannelModeSet(PORTA_1, GPIO);
2 Dio_ChannelModeSet(PORTA_2, SPI_MOSI);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_ChannelDirectionSet (DioChannel_t const Channel, PinModeEnum_t const Mode)

Description:

This function is used to set the direction of an individual channel (pin). The direction is defined by the PinModeEnum_t enum. The valid channels (pins) are defined in the DioChannel_t enum.

PRE-CONDITION: The channel is within the maximum DioChannel_t definition

PRE-CONDITION: The pin direction is within the maximum PinModeEnum_t

PRE-CONDITION: The pin is mode is set to GPIO

POST-CONDITION: The channel direction will be updated to PinMode

Parameters:

in	Channel	is the pin from DioChannel_t that is to have its function changed
in	Mode	is the mode that the pin be multiplexed into. i.e. SPI, UART, etc

Returns:

void

Example:

```
1 Dio_ChannelDirectionSet(PORTA_1, OUTPUT);
2 Dio_ChannelDirectionSet(PORTA_1, INPUT);
```

See also:

[Dio_Init](#)
[Dio_ChannelRead](#)
[Dio_ChannelWrite](#)
[Dio_ChannelToggle](#)
[Dio_ChannelModeSet](#)
[Dio_ChannelDirectionSet](#)
[Dio_RegisterWrite](#)
[Dio_RegisterRead](#)
[Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
10/22/2015	0.6.0	JWB	Merged into HAL
11/10/2015	1.0.0	JWB	Interface Created

void Dio_RegisterWrite (uint32_t const *Address*, TYPE const *Value*)

Description:

This function is used to directly address and modify a Dio register. The function should be used to access specialised functionality in the Dio peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Dio register addresss space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Dio peripheral map
in	<i>Value</i>	is the value to set the Dio register to

Returns:

void

Example:

```
1 Dio_RegisterWrite(0x1000, 0x15);
```

See also:

[Dio_Init](#)
[Dio_ChannelRead](#)
[Dio_ChannelWrite](#)
[Dio_ChannelToggle](#)
[Dio_ChannelModeSet](#)
[Dio_ChannelDirectionSet](#)
[Dio_RegisterWrite](#)
[Dio_RegisterRead](#)
[Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

TYPE Dio_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Dio register. The function should be used to access specialised functionality in the Dio peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Dio register addresss space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Dio register to read
----	----------------	--

Returns:

The current value of the Dio register.

Example:

```
1 DioValue = Dio_RegisterRead(0x1000);
```

See also:

[Dio_Init](#)

[Dio_ChannelRead](#)

[Dio_ChannelWrite](#)

[Dio_ChannelToggle](#)

[Dio_ChannelModeSet](#)

[Dio_ChannelDirectionSet](#)

[Dio_RegisterWrite](#)

[Dio_RegisterRead](#)

[Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Dio_CallbackRegister (DioCallback_t const *Function*, TYPE(*)*(type)* *CallbackFunction*)

Description:

This function is used to set the callback functions of the dio driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The DioCallback_t has been populated PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 DioCallback_t Dio_Function = DIO_SAMPLE_COMPLETE;  
2  
3 Dio_CallbackRegister(Dio_Function, Dio_SampleAverage);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_ChannelDirectionSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)
- [Dio_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

dio_cfg.c File Reference

This module contains the implementation for the digital input/output peripheral configuration.

```
#include "dio_cfg.h"
```

Functions

- [DioConfig_t](#) const *const [Dio_ConfigGet](#) (void)

Variables

- const [DioConfig_t](#) [DioConfig](#) []

Function Documentation

[DioConfig_t](#) const* const [Dio_ConfigGet](#) (void)

Description:

This function is used to initialize the Dio based on the configuration table defined in dio_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const Dio_ConfigType *DioConfig = Dio_GetConfig();
2
3 Dio_Init(DioConfig);
```

See also:

- [Dio_Init](#)
- [Dio_ChannelRead](#)
- [Dio_ChannelWrite](#)
- [Dio_ChannelToggle](#)
- [Dio_ChannelModeSet](#)
- [Dio_RegisterWrite](#)
- [Dio_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Variable Documentation

const [DioConfig_t](#) DioConfig[]

```
Initial value:=
{

{ PORT1_0,      DISABLED,      OUTPUT,      HIGH,      FCN_GPIO      },
{ PORT1\_1,      DISABLED,      OUTPUT,      HIGH,      FCN_GPIO      },
{ PORT1\_2,      DISABLED,      OUTPUT,      HIGH,      FCN_GPIO      },
{ PORT1\_3,      DISABLED,      OUTPUT,      HIGH,      FCN_GPIO      },
{ PORT1_4,      DISABLED,      OUTPUT,      HIGH,      FCN_GPIO      },
{ PORT1\_5,      DISABLED,      OUTPUT,      HIGH,      FCN_GPIO      },
{ PORT1\_6,      DISABLED,      OUTPUT,      HIGH,      FCN_GPIO      },
{ PORT1\_7,      DISABLED,      OUTPUT,      HIGH,      FCN_GPIO      },
}
```

The following array contains the configuration data for each digital input / output peripheral channel (pin). Each row represents a single pin. Each column is representing a member of the [DioConfig_t](#) structure. This table is read in by Dio_Init where each channel is then setup based on this table.

dio_cfg.h File Reference

This module contains interface definition for the Dio configuration.

Data Structures

- struct [DioConfig_t](#)

Macros

- `#define NUMBER_OF_CHANNELS_PER_PORT 8U`
- `#define NUMBER_OF_PORTS 8U`

Enumerations

- enum [DioPinState_t](#) { , [DIO_HIGH](#), [DIO_PIN_STATE_MAX](#) }
- enum [DioChannel_t](#) { [FCPU_HB](#), [PORT1_1](#), [PORT1_2](#), [PORT1_3](#), [UHF_SEL](#), [PORT1_5](#), [PORT1_6](#), [PORT1_7](#), [DIO_MAX_PIN_NUMBER](#) }
- enum [DioMode_t](#)
- enum [DioResistor_t](#)
- enum [DioSlew_t](#) { [FAST](#), [SLOW](#) }

Functions

- [DioConfig_t](#) const *const [Dio_ConfigGet](#) (void)

Detailed Description

This is the header file for the definition of the interface for retrieving the digital input/output configuration table.

Macro Definition Documentation

`#define NUMBER_OF_CHANNELS_PER_PORT 8U`

Defines the number of pins on each processor port.

`#define NUMBER_OF_PORTS 8U`

Defines the number of ports on the processor.

Enumeration Type Documentation

enum [DioPinState_t](#)

Defines the possible states for a digital output pin.

Enumerator

[DIO_HIGH](#) Defines digital state ground

[DIO_PIN_STATE_MAX](#) Defines digital state power Defines the maximum digital state

enum [DioChannel_t](#)

Defines an enumerated list of all the channels (pins) on the MCU device. The last element is used to specify the maximum number of enumerated labels.

Enumerator

FCPU_HB PORT1_0
PORT1_1 PORT1_1
PORT1_2 PORT1_2
PORT1_3 PORT1_3
UHF_SEL PORT1_4
PORT1_5 PORT1_5
PORT1_6 PORT1_6
PORT1_7 PORT1_7
DIO_MAX_PIN_NUMBER MAX CHANNELS

enum [DioMode_t](#)

Defines the possible DIO pin multiplexing values. The datasheet should be reviewed for proper muxing options.

enum [DioResistor_t](#)

Defines the possible states of the channel pull-ups

enum [DioSlew_t](#)

Defines the slew rate settings available

Enumerator

FAST Fast slew rate is configured on the corresponding pin,
SLOW Slow slew rate is configured on the corresponding pin,

Function Documentation

[DioConfig_t](#) const* const Dio_ConfigGet (void)

Description:

This function is used to initialize the Dio based on the configuration table defined in dio_cfg module.

PRE-CONDITION: Configuration table needs to be populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example

```
1 const Dio_ConfigType *DioConfig = Dio_GetConfig();  
2  
3 Dio_Init(DioConfig);
```

See also:

[Dio_Init](#)
[Dio_ChannelRead](#)

[Dio_ChannelWrite](#)
[Dio_ChannelToggle](#)
[Dio_ChannelModeSet](#)
[Dio_RegisterWrite](#)
[Dio_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

flash.c File Reference

The implementation for the flash.

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

Functions

- void [Flash_Init](#) (Flash_Config_t const *const Config)
- void [Flash_Write](#) (uint32_t const Address, uint16_t const *Data, uint8_t const Size)
- void [Flash_Read](#) (uint32_t const Address, uint16_t const *Data, uint8_t const Size)
- void [Flash_Erase](#) (uint32_t const Address)
- void [Flash_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Flash_RegisterRead](#) (uint32_t const Address)
- void [Flash_CallbackRegister](#) (FlashCallback_t const Function, TYPE(*CallbackFunction)(type))

Function Documentation

void Flash_Init (Flash_Config_t const *const *Config*)

Description:

This function initializes the flash driver

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The flash peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	- Pointer to flash configuration table.
----	---------------	---

Returns:

None.

Example:

```
1 const Flash_ConfigType *FlashConfig = Flash_GetConfig();
2 Flash_Init(FlashConfig);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_Write (uint32_t const *Address*, uint16_t const * *Data*, uint8_t const *Size*)

Description:

This function writes data to a location in flash

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The Flash_Init function as been called successfully

POST-CONDITION: The flash peripheral writes the input data to the requested location in flash.

Parameters:

in	<i>Address</i>	- uint32_t, address in flash memory.
in	<i>Data</i>	- uint32_t, pointer to data buffer to write to flash.
in	<i>Size</i>	- uint8_t, size of data to write.

Example:

```
1 uint16_t buffer[8];
2 Flash_Write(0x01234567, buffer, 8);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_Read (uint32_t const *Address*, uint16_t const * *Data*, uint8_t const *Size*)

Description:

This function reads data from flash memory.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The Flash_Init function as been called successfully

POST-CONDITION: The flash peripheral reads data from flash.

Parameters:

in	Address	- uint32_t, address in flash memory.
in	Data	- uint16_t, pointer to data buffer to read to flash.
in	Size	- uint8_t, size of data to read.

Example:

```
1 value = Flash_Read(0x01234567, Buffer, 255);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_Erase (uint32_t const Address)

Description:

This function erases a segment of flash memory.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The Flash_Init function as been called successfully

POST-CONDITION: The flash peripheral reads data from flash.

Parameters:

in	Address	- uint32_t, flash memory address to read
----	---------	--

Returns:

None.

Example:

```
1 Flash_Erase(0x01234567);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)

[Flash_Write](#)
[Flash_Erase](#)
[Flash_RegisterWrite](#)
[Flash_RegisterRead](#)
[Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a flash register. The function should be used to access specialized functionality in the register peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the flash register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the value peripheral map
in	<i>Value</i>	is the value to set the value register to

Returns:

void

Example:

```
1 Flash_RegisterWrite(0x1000, 0x15);
```

See also:

[Flash_Init](#)
[Flash_Read](#)
[Flash_Write](#)
[Flash_Erase](#)
[Flash_RegisterWrite](#)
[Flash_RegisterRead](#)
[Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

uint32_t Flash_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a flash register. The function should be used to access specialized functionality in the flash peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the flash register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the register to read
----	----------------	--

Returns:

The current value of the register.

Example:

```
1 FlashValue = Flash_RegisterRead(0x1000);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_CallbackRegister (FlashCallback_t const *Function*, TYPE(*)*(type) CallbackFunction*)

Description:

This function is used to set the callback functions of the flash driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The FlashCallback_t has been populated PRE-CONDITION: The callback

function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 FlashCallback_t Flash_Function = FLASH_SAMPLE_COMPLETE;  
2  
3 Flash_CallbackRegister(Flash_Function, Flash_SampleAverage);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

flash.h File Reference

The interface definition for the flash.

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

Functions

- void [Flash_Init](#) (Flash_Config_t const *const Config)
- void [Flash_Write](#) (uint32_t const Address, uint16_t const *Data, uint8_t const Size)
- void [Flash_Read](#) (uint32_t const Address, uint16_t const *Data, uint8_t const Size)
- void [Flash_Erase](#) (uint32_t const Address)
- void [Flash_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Flash_RegisterRead](#) (uint32_t const Address)
- void [Flash_CallbackRegister](#) (FlashCallback_t const Function, TYPE(*CallbackFunction)(type))

Detailed Description

This is the header file for the definition of the flash driver functions

Function Documentation

void Flash_Init (Flash_Config_t const *const *Config*)

Description:

This function initializes the flash driver

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The flash peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	- Pointer to flash configuration table.
----	---------------	---

Returns:

None.

Example:

```
1 const Flash_ConfigType *FlashConfig = Flash_GetConfig();
2 Flash_Init(FlashConfig);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
------	------------------	----------	-------------

09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_Write (uint32_t const *Address*, uint16_t const * *Data*, uint8_t const *Size*)

Description:

This function writes data to a location in flash

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The Flash_Init function as been called successfully

POST-CONDITION: The flash peripheral writes the input data to the requested location in flash.

Parameters:

in	<i>Address</i>	- uint32_t, address in flash memory.
in	<i>Data</i>	- uint32_t, pointer to data buffer to write to flash.
in	<i>Size</i>	- uint8_t, size of data to write.

Example:

```

1 uint16_t buffer[8];
2 Flash_Write(0x01234567, buffer, 8);

```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_Read (uint32_t const *Address*, uint16_t const * *Data*, uint8_t const *Size*)

Description:

This function reads data from flash memory.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The Flash_Init function as been called successfully

POST-CONDITION: The flash peripheral reads data from flash.

Parameters:

in	<i>Address</i>	- uint32_t, address in flash memory.
in	<i>Data</i>	- uint16_t, pointer to data buffer to read to flash.
in	<i>Size</i>	- uint8_t, size of data to read.

Example:

```
1 value = Flash_Read(0x01234567, Buffer, 255);
```

See also:

[Flash_Init](#)

[Flash_Read](#)

[Flash_Write](#)

[Flash_Erase](#)

[Flash_RegisterWrite](#)

[Flash_RegisterRead](#)

[Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_Erase (uint32_t const *Address*)

Description:

This function erases a segment of flash memory.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

PRE-CONDITION: The Flash_Init function as been called successfully

POST-CONDITION: The flash peripheral reads data from flash.

Parameters:

in	<i>Address</i>	- uint32_t, flash memory address to read
----	----------------	--

Returns:

None.

Example:

```
1 Flash_Erase(0x01234567);
```

See also:

[Flash_Init](#)
[Flash_Read](#)
[Flash_Write](#)
[Flash_Erase](#)
[Flash_RegisterWrite](#)
[Flash_RegisterRead](#)
[Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a flash register. The function should be used to access specialized functionality in the register peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the flash register address space

POST-CONDITION: The register located at Address will be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the value peripheral map
in	<i>Value</i>	is the value to set the value register to

Returns:

void

Example:

```
1 Flash_RegisterWrite(0x1000, 0x15);
```

See also:

[Flash_Init](#)
[Flash_Read](#)
[Flash_Write](#)
[Flash_Erase](#)
[Flash_RegisterWrite](#)
[Flash_RegisterRead](#)
[Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created

uint32_t Flash_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a flash register. The function should be used to access specialized functionality in the flash peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the flash register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the register to read
----	----------------	--

Returns:

The current value of the register.

Example:

```
1 FlashValue = Flash_RegisterRead(0x1000);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Created

void Flash_CallbackRegister (FlashCallback_t const *Function*, TYPE(*)*(type)* *CallbackFunction*)

Description:

This function is used to set the callback functions of the flash driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the

function will take a parameter to configure the specified callback.

PRE-CONDITION: The FlashCallback_t has been populated PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 FlashCallback_t Flash_Function = FLASH_SAMPLE_COMPLETE;  
2  
3 Flash_CallbackRegister(Flash_Function, Flash_SampleAverage);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

flash_cfg.c File Reference

This module contains the flash configuration code.

```
#include "flash_cfg.h"
```

Functions

- [FlashConfig_t](#) const *const [Flash_ConfigGet](#) (void)

Variables

- const [FlashConfig_t](#) [FlashConfig](#)

Function Documentation

[FlashConfig_t](#) const* const [Flash_ConfigGet](#) (void)

Description:

This function is used to retrieve the flash table that is used to configure the configuration table.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const FlashConfig_t *FlashConfig = Flash_ConfigGet();
2
3 Flash_Init(FlashConfig);
```

See also:

- [Flash_Init](#)
- [Flash_Read](#)
- [Flash_Write](#)
- [Flash_Erase](#)
- [Flash_RegisterWrite](#)
- [Flash_RegisterRead](#)
- [Flash_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Variable Documentation

const [FlashConfig_t](#) FlashConfig

```
Initial value:=  
{  
    M\_CLK,  
    53  
}
```

The Flash Module configuration settings. The flash clock frequency must be 257 kHz to approximately 476 kHz. The correct clock divider must be set in order to divide the selected clock source and meet the frequency requirements.

flash_cfg.h File Reference

This contains the header for the flash configuration.

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

Data Structures

- struct [FlashConfig_t](#)

Enumerations

- enum [FlashClkSrc_t](#) { [AUX_CLK](#), [M_CLK](#), [SYS_CLK](#) }

Functions

- [FlashConfig_t](#) const *const [Flash_ConfigGet](#) (void)

Enumeration Type Documentation

enum [FlashClkSrc_t](#)

The available Flash clock sources.

Enumerator

AUX_CLK Auxilary Clock

M_CLK System Clock

SYS_CLK Sub-System Master Clock

Function Documentation

[FlashConfig_t](#) const* const [Flash_ConfigGet](#) (void)

Description:

This function is used to retrieve the flash table that is used to configure the configuration table.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const FlashConfig_t *FlashConfig = Flash_ConfigGet();
2
3 Flash_Init(FlashConfig);
```

See also:

[Flash_Init](#)

[Flash_Read](#)

[Flash_Write](#)

[Flash_Erase](#)

[Flash_RegisterWrite](#)

[Flash_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

i2c.c File Reference

The implementation for the i2c.

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

Functions

- void [I2c_FrequencySet](#) (I2cChannel_t Channel, uint32_t BaudRate)
- void [I2c_Init](#) (I2cConfig_t const *const Config)
- void [I2c_DeInit](#) (I2cChannel_t const Channel)
- void [I2c_PowerModeSet](#) (I2cChannel_t const Channel, I2cPowerMode_t const PowerMode)
- void [I2c_SlaveAddressSet](#) (I2cChannel_t const Channel, uint8_t const Address)
- uint8_t [I2c_Transfer](#) (I2cTransfer_t *const Config)
- void [I2c0_ISR](#) (void)
- void [I2c1_ISR](#) (void)
- void [I2c_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [I2c_RegisterRead](#) (uint32_t const Address)
- void [I2c_CallbackRegister](#) (I2cCallback_t const Function, TYPE(*CallbackFunction)(type))

Function Documentation

```
void I2c_FrequencySet (I2cChannel_t const Channel, uint32_t const BaudRate) [inline]
```

Description:

This function is used to determine the proper I2c clock frequency dividers using the above SCL Divider lookup table. Loop through all possible Multiplier and Divider combinations to determine which provides the closest clock frequency to the desired frequency.

PRE-CONDITION: The I2c_Init function must have been called with valid configuration

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The i2c peripheral is setup with the new frequency.

Parameters:

in	<i>Channel</i>	- uint8_t, I2c channel value
in	<i>BaudRate</i>	- uint32_t desired clock frequency in Hz

Returns:

void

Example:

```
1  const I2cConfig_t *I2cConfig = I2c_ConfigGet();
2
3  I2c_Init(I2cConfig);
4  I2c_SetFreq(I2c_0, 100000);
```

See also:

- I2c_ConfigGet
- [I2c_Init](#)
- [I2c_DeInit](#)
- [I2c_Transfer](#)
- [I2c_PowerModeSet](#)
- [I2c_SlaveAddressSet](#)
- [I2c_RegisterWrite](#)

[I2c_RegisterRead](#)
[I2c_SlaveTxBufferSet](#)
[I2c_SlaveRxBufferSet](#)
[I2c_ByteCountGet](#)
[I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void I2c_Init (I2cConfig_t const *const Config)

Description:

This function is used to initialize the I2c based on the configuration table defined in i2c_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The i2c peripheral is setup with the configuration settings.

Parameters:

in	Config	is a pointer to the configuration table that contains the initialization for the peripheral.
----	--------	--

Returns:

void

Example:

```
1 const I2cConfig_t *I2cConfig = I2c_ConfigGet();
2
3 I2c_Init(I2cConfig);
```

See also:

[I2c_ConfigGet](#)
[I2c_Init](#)
[I2c_DeInit](#)
[I2c_Transfer](#)
[I2c_PowerModeSet](#)
[I2c_SlaveAddressSet](#)
[I2c_RegisterWrite](#)
[I2c_RegisterRead](#)
[I2c_SlaveTxBufferSet](#)
[I2c_SlaveRxBufferSet](#)
[I2c_ByteCountGet](#)
[I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

--	--	--	--

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void I2c_DeInit (I2cChannel_t const *Channel*)

Description:

The I2c_Init is used to deinitialize an Inter-Integrated Circuit communication peripheral. All registers are cleared to the RESET value.

PRE-CONDITION: The I2c_Init function must have been called with valid configuration
PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The i2c peripheral is setup with the configuration settings.

Parameters:

in	<i>Channel</i>	is the i2c channel to de-initialize
----	----------------	-------------------------------------

Returns:

void

Example:

```
1      const I2cConfig_t *I2cConfig = I2c_ConfigGet();
2
3      I2c_Init(I2cConfig);
4 I2c_DeInit(I2c_0);
```

See also:

- I2c_ConfigGet
- [I2c_Init](#)
- [I2c_DeInit](#)
- [I2c_Transfer](#)
- [I2c_PowerModeSet](#)
- [I2c_SlaveAddressSet](#)
- [I2c_RegisterWrite](#)
- [I2c_RegisterRead](#)
- I2c_SlaveTxBufferSet
- I2c_SlaveRxBufferSet
- I2c_ByteCountGet
- [I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void I2c_PowerModeSet (I2cChannel_t const *Channel*, I2cPowerMode_t const *PowerMode*)

Description:

The I2c_PowerMode is used to place the I2c module into operate or halt mode. In halt mode, the I2c clock generation is stopped. In operate mode, the I2c clock is generated and the module performs as configured.

PRE-CONDITION: The I2c_Init function must have been called with valid configuration
PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The i2c peripheral is setup with the power mode.

Parameters:

in	<i>Channel</i>	- uint8_t, I2c channel value
in	<i>PowerMode</i>	- I2cPowerMode_t, the mode of operation.

Returns:

void

Example:

```
1      const I2cConfig_t *I2cConfig = I2c_ConfigGet();
2
3      I2c_Init(I2cConfig);
4  I2c_SetFreq(I2c_0, 100000);
5  I2c_PowerMode(I2c_HALT, I2c_0);
```

See also:

- I2c_ConfigGet
- [I2c_Init](#)
- [I2c_DeInit](#)
- [I2c_Transfer](#)
- [I2c_PowerModeSet](#)
- [I2c_SlaveAddressSet](#)
- [I2c_RegisterWrite](#)
- [I2c_RegisterRead](#)
- I2c_SlaveTxBufferSet
- I2c_SlaveRxBufferSet
- I2c_ByteCountGet
- [I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void I2c_SlaveAddressSet (I2cChannel_t const *Channel*, uint8_t const *Address*)

Description:

This function is used to set the I2c device address of a channel.

PRE-CONDITION: The I2c_Init function must have been called with valid configuration
PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The i2c channel is setup with the target slave address

Parameters:

in	<i>Channel</i>	- uint8_t, I2c channel value
in	<i>Address</i>	- uint8_t, channel address.

Returns:

void

Example:

```
1      const I2cConfig_t *I2cConfig = I2c_ConfigGet();
2
3      I2c_Init(I2cConfig);
4  I2c_SetFreq(I2c_0, 100000);
5  I2c_PowerMode(I2c_HALT, I2c_0);
6      I2c_SetSlaveAddress(0x20, I2c_0);
```

See also:

- I2c_ConfigGet
- [I2c_Init](#)
- [I2c_DeInit](#)
- [I2c_Transfer](#)
- [I2c_PowerModeSet](#)
- [I2c_SlaveAddressSet](#)
- [I2c_RegisterWrite](#)
- [I2c_RegisterRead](#)
- I2c_SlaveTxBufferSet
- I2c_SlaveRxBufferSet
- I2c_ByteCountGet
- [I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint8_t I2c_Transfer (I2cTransfer_t *const Config)

Description:

The I2c_Transfer is used to write and read data from a slave device.

PRE-CONDITION: The I2c_Init function must have been called with valid configuration
PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The i2c transaction is carried out on the bus.

Parameters:

in,out	<i>Config</i>	- Pointer to the I2c transfer type.
--------	---------------	-------------------------------------

Returns:

uint8_t 0 - Transfer finished successfully. 1 - Transfer was aborted.

Example:

```
1      const I2cConfig_t *I2cConfig = I2c_ConfigGet();
2  const I2cTransfer_t I2c_Data_Write = {...}
3
4      I2c_Init(I2cConfig);
5  I2c_SetFreq(I2c_0, 100000);
6  I2c_PowerMode(I2c_HALT, I2c_0);
7      I2c_SetSlaveAddress(0x20, I2c_0);
8  I2c_SetSlaveTxBuffer(I2c_0, TxDataBuffer, 4);
9  I2c_SetSlaveRxBuffer(I2c_0, RxDataBuffer, 4);
10 uint8_t count = I2c_GetByteCount(I2c_1);
11
12      I2c_Transfer(&I2c_Data_Write);
```

See also:

- I2c_ConfigGet
- [I2c_Init](#)
- [I2c_DeInit](#)
- [I2c_Transfer](#)
- [I2c_PowerModeSet](#)
- [I2c_SlaveAddressSet](#)
- [I2c_RegisterWrite](#)
- [I2c_RegisterRead](#)
- I2c_SlaveTxBufferSet
- I2c_SlaveRxBufferSet
- I2c_ByteCountGet
- [I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void I2c0_ISR (void)

Description:

This function is the I2c Channel 0 Interrupt Service Routine.

PRE-CONDITION: The I2c_Init function must have been called with valid configuration PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The i2c transaction is carried out on the bus.

Returns:

None

Example:

```
1      const I2cConfig_t *I2cConfig = I2c_ConfigGet();
2 const I2cTransfer_t I2c_Data_Write = {...}
3
4      I2c_Init(I2cConfig);
5 I2c_SetFreq(I2c_0, 100000);
6 I2c_PowerMode(I2c_HALT, I2c_0);
7      I2c_SetSlaveAddress(0x20, I2c_0);
8 I2c_SetSlaveTxBuffer(I2c_0, TxDataBuffer, 4);
9 I2c_SetSlaveRxBuffer(I2c_0, RxDataBuffer, 4);
10 uint8_t count = I2c_GetByteCount(I2c_1);
11
12      I2c_Transfer(&I2c_Data_Write);
```

See also:

- I2c_ConfigGet
- [I2c_Init](#)
- [I2c_DeInit](#)
- [I2c_Transfer](#)
- [I2c_PowerModeSet](#)
- [I2c_SlaveAddressSet](#)
- [I2c_RegisterWrite](#)
- [I2c_RegisterRead](#)
- I2c_SlaveTxBufferSet
- I2c_SlaveRxBufferSet
- I2c_ByteCountGet
- [I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void I2c1_ISR (void)

Description:

This function is the I2c Channel 0 Interrupt Service Routine.

PRE-CONDITION: The I2c_Init function must have been called with valid configuration

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The i2c transaction is carried out on the bus.

Returns:

None

Example:

```
1      const I2cConfig_t *I2cConfig = I2c_ConfigGet();
2 const I2cTransfer_t I2c_Data_Write = {...}
3
4      I2c_Init(I2cConfig);
```

```

5 I2c_SetFreq(I2c_0, 100000);
6 I2c_PowerMode(I2c_HALT, I2c_0);
7     I2c_SetSlaveAddress(0x20, I2c_0);
8 I2c_SetSlaveTxBuffer(I2c_0, TxDataBuffer, 4);
9 I2c_SetSlaveRxBuffer(I2c_0, RxDataBuffer, 4);
10 uint8_t count = I2c_GetByteCount(I2c_1);
11
12     I2c_Transfer(&I2c_Data_Write);

```

See also:

[I2c_ConfigGet](#)
[I2c_Init](#)
[I2c_DeInit](#)
[I2c_Transfer](#)
[I2c_PowerModeSet](#)
[I2c_SlaveAddressSet](#)
[I2c_RegisterWrite](#)
[I2c_RegisterRead](#)
[I2c_SlaveTxBufferSet](#)
[I2c_SlaveRxBufferSet](#)
[I2c_ByteCountGet](#)
[I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void I2c_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a I2c register. The function should be used to access specialized functionality in the I2c peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the I2c register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the I2c peripheral map
in	<i>Value</i>	is the value to set the I2c register to

Returns:

void

Example:

```
1 I2c_RegisterWrite(0x1000, 0x15);
```

See also:

[I2c_ConfigGet](#)
[I2c_Init](#)
[I2c_DeInit](#)
[I2c_Transfer](#)
[I2c_PowerModeSet](#)
[I2c_SlaveAddressSet](#)
[I2c_RegisterWrite](#)
[I2c_RegisterRead](#)
[I2c_SlaveTxBufferSet](#)
[I2c_SlaveRxBufferSet](#)
[I2c_ByteCountGet](#)
[I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t I2c_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a I2c register. The function should be used to access specialized functionality in the I2c peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the I2c register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the I2c register to read
----	----------------	--

Returns:

The current value of the I2c register.

Example:

```
1 I2cValue = I2c_RegisterRead(0x1000);
```

See also:

[I2c_ConfigGet](#)
[I2c_Init](#)
[I2c_DeInit](#)
[I2c_Transfer](#)
[I2c_PowerModeSet](#)
[I2c_SlaveAddressSet](#)
[I2c_RegisterWrite](#)
[I2c_RegisterRead](#)
[I2c_SlaveTxBufferSet](#)
[I2c_SlaveRxBufferSet](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void I2c_CallbackRegister (I2cCallback_t const *Function*, TYPE*)(type) *CallbackFunction*)

Description:

This function is used to set the callback functions of the I2c driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The I2cCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 I2cCallback_t I2c_Function = I2c_SAMPLE_COMPLETE;  
2  
3 I2c_CallbackRegister(I2c_Function, I2c_SampleAverage);
```

See also:

I2c_ConfigGet
[I2c_Init](#)
[I2c_DeInit](#)
[I2c_Transfer](#)
[I2c_PowerModeSet](#)
[I2c_SlaveAddressSet](#)
[I2c_RegisterWrite](#)
[I2c_RegisterRead](#)
I2c_SlaveTxBufferSet
I2c_SlaveRxBufferSet
I2c_ByteCountGet
[I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

--	--	--	--

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

i2c.h File Reference

The interface definition for i2c.

```
#include <stdint.h>
#include "i2c_cfg.h"
```

Data Structures

- struct [I2CTransfer_t](#)

Macros

- #define [NUM_FREQ_MULT](#) 3

Enumerations

- enum [I2CTransferMode_t](#) { [I2C_TRANSMIT](#), [I2C_RECEIVE](#) }
- enum [I2CPowerMode_t](#) { [I2C_OPERATE](#), [I2C_HALT](#) }

Functions

- void [I2c_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [I2c_RegisterRead](#) (uint32_t const Address)
- void [I2c_CallbackRegister](#) (I2cCallback_t const Function, TYPE(*CallbackFunction)(type))

Detailed Description

This is the header file for the definition of the interface for the I2C bus.

Macro Definition Documentation

#define NUM_FREQ_MULT 3

Defines the number of possible multiplier values in the I2C frequency divider register

Enumeration Type Documentation

enum [I2CTransferMode_t](#)

Enumeration I2C_TransferMode Defines the two transfer modes which the I2C module has available. It can either be transmitting or receiving.

Enumerator

- I2C_TRANSMIT* I2C channel is transmitting
- I2C_RECEIVE* I2C channel is receiving

enum [I2CPowerMode_t](#)

Enumeration I2C_PowerModeType Defines the two power modes which the I2C module has available. It can either be operating or halted.

Enumerator

- I2C_OPERATE* Set the I2C channel to operate mode

Function Documentation

void I2c_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a I2c register. The function should be used to access specialized functionality in the I2c peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the I2c register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the I2c peripheral map
in	<i>Value</i>	is the value to set the I2c register to

Returns:

void

Example:

```
1 I2c_RegisterWrite(0x1000, 0x15);
```

See also:

[I2c_ConfigGet](#)

[I2c_Init](#)

[I2c_DeInit](#)

[I2c_Transfer](#)

[I2c_PowerModeSet](#)

[I2c_SlaveAddressSet](#)

[I2c_RegisterWrite](#)

[I2c_RegisterRead](#)

[I2c_SlaveTxBufferSet](#)

[I2c_SlaveRxBufferSet](#)

[I2c_ByteCountGet](#)

[I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t I2c_RegisterRead (uint32_t const *Address*)

Description:

PRE-CONDITION: Address is within the boundaries of the I2c register address space

POST-CONDITION: The value stored in the register is returned to the caller

in	<i>Address</i>	is the address of the I2c register to read
----	----------------	--

The current value of the I2c register.

```
1 I2cValue = I2c_RegisterRead(0x1000);
```

- [I2c_ConfigGet](#)
- [I2c_Init](#)
- [I2c_DeInit](#)
- [I2c_Transfer](#)
- [I2c_PowerModeSet](#)
- [I2c_SlaveAddressSet](#)
- [I2c_RegisterWrite](#)
- [I2c_RegisterRead](#)
- [I2c_SlaveTxBufferSet](#)
- [I2c_SlaveRxBufferSet](#)
- [I2c_ByteCountGet](#)
- [I2c_CallbackRegister](#)

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 I2cCallback_t I2c_Function = I2c_SAMPLE_COMPLETE;  
2  
3 I2c_CallbackRegister(I2c_Function, I2c_SampleAverage);
```

See also:

- I2c_ConfigGet
- [I2c_Init](#)
- [I2c_DeInit](#)
- [I2c_Transfer](#)
- [I2c_PowerModeSet](#)
- [I2c_SlaveAddressSet](#)
- [I2c_RegisterWrite](#)
- [I2c_RegisterRead](#)
- I2c_SlaveTxBufferSet
- I2c_SlaveRxBufferSet
- I2c_ByteCountGet
- [I2c_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

i2c_cfg.h File Reference

This module contains the configuration interface for i2c.

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

Data Structures

- struct [I2CConfig_t](#)

Macros

- #define [I2C_INTERRUPT](#)
- #define [I2C_POLLING_DELAY](#) 400

Enumerations

- enum [I2CAAddr_t](#) { [I2C_7bit](#), [I2C_10bit](#) }
- enum [I2CMode_t](#) { [I2C_SLAVE](#), [I2C_MASTER](#) }
- enum [I2CChannel_t](#) { [I2C_0](#), [I2C_1](#), [NUM_I2C_CHANNELS](#) }

Functions

- [I2CConfig_t](#) const *const [I2C_ConfigGet](#) (void)

Macro Definition Documentation

#define I2C_INTERRUPT

Select the I2C transfer method. Define either POLLING or INTERRUPT.

#define I2C_POLLING_DELAY 400

Define the delay length used by I2C_Transfer in POLLING mode. For system clock frequency of 42 MHz and I2C clock frequency of 100 kHz, a delay of 400 is sufficient.

Enumeration Type Documentation

enum [I2CAAddr_t](#)

Enumeration I2C_Addtype The I2C AddType which defines the number of bits used for addressing the I2C slaves.

Enumerator

- I2C_7bit* 7-bit I2C addressing
- I2C_10bit* 10-bit I2C addressing

enum [I2CMode_t](#)

Enumeration I2C_ModeType The I2C ModeType which defines the I2C channel modes.

Enumerator

- I2C_SLAVE* I2C Master Device
- I2C_MASTER* I2C Slave Device

enum [I2CChannel_t](#)

This enumeration defines a list of the i2c channels

Enumerator

- I2C_0* I2C Channel 0
- I2C_1* I2C Channel 1
- NUM_I2C_CHANNELS* Number of I2C channels

Function Documentation

[I2CConfig_t](#) const* const I2C_ConfigGet (void)

Description:

This function return a pointer to the I2C configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const I2cConfig_t * I2cConfig = I2c_ConfigGet();
2
3 I2c_Init(I2cConfig);
```

See also:

- I2c_ConfigGet
- [I2c_Init](#)
- [I2c_DeInit](#)
- I2C_Transfer
- [I2c_PowerModeSet](#)
- [I2c_SlaveAddressSet](#)
- [I2c_RegisterWrite](#)
- [I2c_RegisterRead](#)
- I2c_SlaveTxBufferSet
- I2c_SlaveRxBufferSet
- I2C_ByteCountGet

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

isr.c File Reference

The implementation for the interrupts.

```
#include "isr.h"
```

Functions

- void [Isr_Enable](#) ([IsrState_t](#) const *IsrIndex*)
- void [Isr_Disable](#) ([IsrState_t](#) const *IsrIndex*)
- void [Isr_GlobalEnable](#) ([IsrState_t](#) *IsrIndex*)
- void [Isr_GlobalDisable](#) (void)
- [IsrState_t](#) [Isr_GlobalStateGet](#) (void)
- void [Isr_CriticalSectionStart](#) (void)
- void [Isr_CriticalSectionEnd](#) (void)

Function Documentation

void [Isr_Enable](#) ([IsrState_t](#) const *IsrIndex*)

Description:

This function is used to enable individual interrupts

PRE-CONDITION: Desired interrupt must contain an interrupt handler

PRE-CONDITION: Desired interrupt must be registered with the vector table

POST-CONDITION: The desired interrupt will be enabled.

Parameters:

in	<i>IsrIndex</i>	The index of the ISR that will be enabled
----	-----------------	---

Returns:

void

Example:

```
1 Isr_Enable(Isr_Uart0);
```

See also:

- [Isr_Enable](#)
- [Isr_Disable](#)
- [Isr_GlobalEnable](#)
- [Isr_GlobalDisable](#)
- [Isr_GlobalStateGet](#)
- [Isr_CriticalSectionStart](#)
- [Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Isr_Disable ([IsrState_t](#) const *IsrIndex*)

Description:

This function is used to disable individual interrupts

PRE-CONDITION: Desired interrupt must contain an interrupt handler

PRE-CONDITION: Desired interrupt must be registered with the vector table

PRE-CONDITION: Desired interrupt must be enabled

POST-CONDITION: The desired interrupt will be disabled.

Parameters:

in	<i>IsrIndex</i>	The index of the ISR that will be enabled
----	-----------------	---

Returns:

void

Example:

```
1 Isr_Disable(Isr_Uart0);
```

See also:

- [Isr_Enable](#)
- [Isr_Disable](#)
- [Isr_GlobalEnable](#)
- [Isr_GlobalDisable](#)
- [Isr_GlobalStateGet](#)
- [Isr_CriticalSectionStart](#)
- [Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Isr_GlobalEnable ([IsrState_t](#) *IsrIndex*)

Description:

This function is used to enable global interrupts

PRE-CONDITION: Active interrupts require an interrupt handler

PRE-CONDITION: Desired interrupt must be registered with the vector table

POST-CONDITION: Global interrupts will be enabled.

Returns:

void

Example:

```
1 Isr_GlobalEnable();
```

See also:

- [Isr_Enable](#)
- [Isr_Disable](#)
- [Isr_GlobalEnable](#)
- [Isr_GlobalDisable](#)
- [Isr_GlobalStateGet](#)
- [Isr_CriticalSectionStart](#)
- [Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Isr_GlobalDisable (void)

Description:

This function is used to disable global interrupts

PRE-CONDITION: None

POST-CONDITION: All interrupts will be disabled.

Returns:

void

Example:

```
1 Isr_GlobalDisable();
```

See also:

- [Isr_Enable](#)
- [Isr_Disable](#)
- [Isr_GlobalEnable](#)
- [Isr_GlobalDisable](#)
- [Isr_GlobalStateGet](#)
- [Isr_CriticalSectionStart](#)
- [Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[IsrState_t](#) Isr_GlobalStateGet (void)

Description:

This function is used to retrieve the state of the global interrupts.

PRE-CONDITION: None

POST-CONDITION: Current state of global variables will be returned

Returns:

IsrState_t The state of the global interrupts

Example:

```
1 IsrState_t IsrState;  
2  
3 IsrState = Isr_GlobalStateGet();
```

See also:

- [Isr_Enable](#)
- [Isr_Disable](#)
- [Isr_GlobalEnable](#)
- [Isr_GlobalDisable](#)
- [Isr_GlobalStateGet](#)
- [Isr_CriticalSectionStart](#)
- [Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Isr_CriticalSectionStart (void)

Description:

This function is used to start a critical section of code. It disables the global interrupts.

PRE-CONDITION: Global interrupts are enabled

POST-CONDITION: All interrupts will be disabled.

Returns:

IsrState_t The state of the global interrupts

Example:

```

1 Isr_CriticalSectionStart();
2
3 Tick = Timer_TickGet();
4
5 Isr_CriticalSectionEnd();

```

See also:

[Isr_Enable](#)
[Isr_Disable](#)
[Isr_GlobalEnable](#)
[Isr_GlobalDisable](#)
[Isr_GlobalStateGet](#)
[Isr_CriticalSectionStart](#)
[Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Isr_CriticalSectionEnd (void)

Description:

This function is used to end a critical section of code. It checks the previous state of the global interrupts and restores that state.

PRE-CONDITION: Global interrupts are enabled PRE-CONDITION: [Isr_CriticalSectionStart\(\)](#) has been called

POST-CONDITION: Critical section will be ended with interrupts enabled

Returns:

None

Example:

```

1 Isr_CriticalSectionStart();
2
3 Tick = Timer_TickGet();
4
5 Isr_CriticalSectionEnd();

```

See also:

[Isr_Enable](#)
[Isr_Disable](#)
[Isr_GlobalEnable](#)
[Isr_GlobalDisable](#)
[Isr_GlobalStateGet](#)
[Isr_CriticalSectionStart](#)
[Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

isr.h File Reference

The interface definition for interrupts.

Enumerations

- enum [IsrState_t](#) { [ISR_DISABLED](#), [ISR_ENABLED](#) }

Functions

- void [Isr_Enable](#) ([IsrState_t](#) const IsrIndex)
- void [Isr_Disable](#) ([IsrState_t](#) const IsrIndex)
- void [Isr_GlobalDisable](#) (void)
- [IsrState_t](#) [Isr_GlobalStateGet](#) (void)
- void [Isr_CriticalSectionStart](#) (void)
- void [Isr_CriticalSectionEnd](#) (void)

Detailed Description

This is the header file for the definition of the isr driver functions

Enumeration Type Documentation

enum [IsrState_t](#)

This enumeration is a list of test types

Enumerator

ISR_DISABLED Global Interrupts Disabled

ISR_ENABLED Global Interrupts Enabled

Function Documentation

void [Isr_Enable](#) ([IsrState_t](#) const *IsrIndex*)

Description:

This function is used to enable individual interrupts

PRE-CONDITION: Desired interrupt must contain an interrupt handler

PRE-CONDITION: Desired interrupt must be registered with the vector table

POST-CONDITION: The desired interrupt will be enabled.

Parameters:

in	<i>IsrIndex</i>	The index of the ISR that will be enabled
----	-----------------	---

Returns:

void

Example:

```
1 Isr_Enable(Isr_Uart0);
```

See also:

[Isr_Enable](#)

[Isr_Disable](#)
[Isr_GlobalEnable](#)
[Isr_GlobalDisable](#)
[Isr_GlobalStateGet](#)
[Isr_CriticalSectionStart](#)
[Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Isr_Disable ([IsrState_t](#) const *IsrIndex*)

Description:

This function is used to disable individual interrupts

PRE-CONDITION: Desired interrupt must contain an interrupt handler

PRE-CONDITION: Desired interrupt must be registered with the vector table

PRE-CONDITION: Desired interrupt must be enabled

POST-CONDITION: The desired interrupt will be disabled.

Parameters:

in	<i>IsrIndex</i>	The index of the ISR that will be enabled
----	-----------------	---

Returns:

void

Example:

```
1 Isr_Disable(Isr_Uart0);
```

See also:

[Isr_Enable](#)
[Isr_Disable](#)
[Isr_GlobalEnable](#)
[Isr_GlobalDisable](#)
[Isr_GlobalStateGet](#)
[Isr_CriticalSectionStart](#)
[Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Isr_GlobalDisable (void)

Description:

This function is used to disable global interrupts

PRE-CONDITION: None

POST-CONDITION: All interrupts will be disabled.

Returns:

void

Example:

```
1 Isr_GlobalDisable();
```

See also:

- [Isr_Enable](#)
- [Isr_Disable](#)
- [Isr_GlobalEnable](#)
- [Isr_GlobalDisable](#)
- [Isr_GlobalStateGet](#)
- [Isr_CriticalSectionStart](#)
- [Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

[IsrState_t](#) Isr_GlobalStateGet (void)

Description:

This function is used to retrieve the state of the global interrupts.

PRE-CONDITION: None

POST-CONDITION: Current state of global variables will be returned

Returns:

IsrState_t The state of the global interrupts

Example:

```
1 IsrState_t IsrState;  
2  
3 IsrState = Isr_GlobalStateGet();
```

See also:

[Isr_Enable](#)
[Isr_Disable](#)
[Isr_GlobalEnable](#)
[Isr_GlobalDisable](#)
[Isr_GlobalStateGet](#)
[Isr_CriticalSectionStart](#)
[Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Isr_CriticalSectionStart (void)

Description:

This function is used to start a critical section of code. It disables the global interrupts.

PRE-CONDITION: Global interrupts are enabled

POST-CONDITION: All interrupts will be disabled.

Returns:

IsrState_t The state of the global interrupts

Example:

```
1 Isr_CriticalSectionStart();  
2  
3 Tick = Timer_TickGet();  
4  
5 Isr_CriticalSectionEnd();
```

See also:

[Isr_Enable](#)
[Isr_Disable](#)
[Isr_GlobalEnable](#)
[Isr_GlobalDisable](#)
[Isr_GlobalStateGet](#)
[Isr_CriticalSectionStart](#)
[Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Isr_CriticalSectionEnd (void)

Description:

This function is used to end a critical section of code. It checks the previous state of the global interrupts and restores that state.

PRE-CONDITION: Global interrupts are enabled PRE-CONDITION: [Isr_CriticalSectionStart\(\)](#) has been called

POST-CONDITION: Critical section will be ended with interrupts enabled

Returns:

None

Example:

```
1 Isr_CriticalSectionStart();
2
3 Tick = Timer_TickGet();
4
5 Isr_CriticalSectionEnd();
```

See also:

- [Isr_Enable](#)
- [Isr_Disable](#)
- [Isr_GlobalEnable](#)
- [Isr_GlobalDisable](#)
- [Isr_GlobalStateGet](#)
- [Isr_CriticalSectionStart](#)
- [Isr_CriticalSectionEnd](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

mcu.c File Reference

The implementation for the mcu.

```
#include "mcu.h"
#include "mcu_cfg.h"
#include "constants.h"
```

Functions

- void [Mcu_Init](#) ([McuConfig_t](#) const *const Config)
- void [Mcu_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Mcu_RegisterRead](#) (uint32_t const Address)
- void [Mcu_CallbackRegister](#) (McuCallback_t const Function, TYPE(*CallbackFunction)(type))

Function Documentation

void [Mcu_Init](#) ([McuConfig_t](#) const *const *Config*)

Description:

This function initializes the mcu clock.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The Mcu peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const McuConfig_t *McuConfig = Mcu_ConfigGet();
2
3 Mcu_Init(McuConfig);
```

See also:

- [Mcu_ConfigGet](#)
- [Mcu_Init](#)
- [Mcu_RegisterWrite](#)
- [Mcu_RegisterRead](#)
- [Mcu_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Mcu_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Mcu register. The function should be used to access specialized functionality in the Mcu peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Mcu register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Mcu peripheral map
in	<i>Value</i>	is the value to set the Mcu register to

Returns:

void

Example:

```
1 Mcu_RegisterWrite(0x1000, 0x15);
```

See also:

- [Mcu_ConfigGet](#)
- [Mcu_Init](#)
- [Mcu_RegisterWrite](#)
- [Mcu_RegisterRead](#)
- [Mcu_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Mcu_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Mcu register. The function should be used to access specialized functionality in the Mcu peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Mcu register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Mcu register to read
----	----------------	--

Returns:

The current value of the Mcu register.

Example:

```
1 McuValue = Mcu_RegisterRead(0x1000);
```

See also:

[Mcu_ConfigGet](#)

[Mcu_Init](#)

[Mcu_RegisterWrite](#)

[Mcu_RegisterRead](#)

[Mcu_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Mcu_CallbackRegister (McuCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the mcu driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The McuCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 McuCallback_t Mcu_Function = MCU_SAMPLE_COMPLETE;  
2  
3 Mcu_CallbackRegister(Mcu_Function, Mcu_SampleAverage);
```

See also:

[Mcu_ConfigGet](#)

[Mcu_Init](#)

[Mcu_RegisterWrite](#)

[Mcu_RegisterRead](#)

[Mcu_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

mcu.h File Reference

The interface definition for the mcu.

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

Macros

- #define [CANNED_OSC](#) 0
- #define [CRYSTAL](#) 1

Functions

- void [Mcu_Init](#) ([McuConfig_t](#) const *const Config)
- void [Mcu_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Mcu_RegisterRead](#) (uint32_t const Address)
- void [Mcu_CallbackRegister](#) (McuCallback_t const Function, TYPE(*CallbackFunction)(type))

Detailed Description

This is the header file for the definition of the interface for the watchdog timer.

Macro Definition Documentation

#define CANNED_OSC 0

Constant used by pll_init to select an external clock

#define CRYSTAL 1

Constant used by pll_init to select a crystal oscillator

Function Documentation

void Mcu_Init ([McuConfig_t](#) const *const *Config*)

Description:

This function initializes the mcu clock.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The Mcu peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const McuConfig_t *McuConfig = Mcu_ConfigGet();
2
3 Mcu_Init(McuConfig);
```

See also:[Mcu_ConfigGet](#)[Mcu_Init](#)[Mcu_RegisterWrite](#)[Mcu_RegisterRead](#)[Mcu_CallbackRegister](#)**- HISTORY OF CHANGES -**

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Mcu_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Mcu register. The function should be used to access specialized functionality in the Mcu peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Mcu register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Mcu peripheral map
in	<i>Value</i>	is the value to set the Mcu register to

Returns:

void

Example:

```
1 Mcu_RegisterWrite(0x1000, 0x15);
```

See also:[Mcu_ConfigGet](#)[Mcu_Init](#)[Mcu_RegisterWrite](#)[Mcu_RegisterRead](#)[Mcu_CallbackRegister](#)**- HISTORY OF CHANGES -**

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Mcu_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Mcu register. The function should be used to access specialized functionality in the Mcu peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Mcu register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Mcu register to read
----	----------------	--

Returns:

The current value of the Mcu register.

Example:

```
1 McuValue = Mcu_RegisterRead(0x1000);
```

See also:

- [Mcu_ConfigGet](#)
- [Mcu_Init](#)
- [Mcu_RegisterWrite](#)
- [Mcu_RegisterRead](#)
- [Mcu_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Mcu_CallbackRegister (McuCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the mcu driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The McuCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 McuCallback_t Mcu_Function = MCU_SAMPLE_COMPLETE;  
2  
3 Mcu_CallbackRegister(Mcu_Function, Mcu_SampleAverage);
```

See also:

- [Mcu_ConfigGet](#)
- [Mcu_Init](#)
- [Mcu_RegisterWrite](#)
- [Mcu_RegisterRead](#)
- [Mcu_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

mcu_cfg.c File Reference

This module contains the configuration for the mcu module.

```
#include "mcu_cfg.h"
#include "constants.h"
```

Functions

- [McuConfig_t](#) const *const [Mcu_ConfigGet](#) (void)

Variables

- const [McuConfig_t](#) [McuConfig](#)

Function Documentation

[McuConfig_t](#) const* const [Mcu_ConfigGet](#) (void)

Description:

This function return a pointer to the MCU configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const McuConfig_t * McuConfig = Mcu_ConfigGet();
2
3 Mcu_Init(McuConfig);
```

See also:

- [Mcu_ConfigGet](#)
- [Mcu_Init](#)
- Mcu_FEEInit
- Mcu_FBIInit
- Mcu_FBEInit
- Mcu_PBEInit
- Mcu_PEEInit
- Mcu_BLPIInit
- Mcu_BLPEInit
- Mcu_TimeoutStart
- Mcu_TimeoutCheck
- Mcu_PllInit

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Variable Documentation

const [McuConfig_t](#) McuConfig

```
Initial value:=
{
    FEI\_MODE,
    DIV\_256,
    MHZ\_40\_50,
    OSC,
    DISABLED,
    2,
    DISABLED,
    24,
    DISABLED,
    DISABLED,
    FAST\_INT,
    DIV\_1,
    LOW\_POWER,
    DISABLED,
    DISABLED,
    DISABLED,
    DISABLED,
    1,
    2,
}
```

The Mcu configuration settings to initialize the clock registers.

mcu_cfg.h File Reference

This module contains the configuration interface for mcu.

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

Data Structures

- struct [McuConfig_t](#)

Macros

- #define [GetSystemClock\(\)](#) 48000000UL
- #define [GetInstructionClock\(\)](#) ([GetSystemClock\(\)](#) / 2)
- #define [GetOscillatorFrequency\(\)](#) 8000000UL

Enumerations

- enum [McuMode_t](#) { [FEI_MODE](#) }
- enum [McuFreqMode_t](#) { [LOW_POWER](#), [HIGH_GAIN](#) }
- enum [McuFreqRange_t](#) { [LOW_RANGE](#), [HIGH_RANGE](#), [VERY_HIGH_RANGE](#) }
- enum [McuDiv_t](#) { [DIV_1](#) = 0U, [DIV_2](#) = 1U, [DIV_4](#) = 2U, [DIV_8](#) = 3U, [DIV_16](#) = 4U, [DIV_32](#) = 5U, [DIV_64](#) = 6U, [DIV_128](#) = 7U, [DIV_32_HIGH](#) = 0U, [DIV_64_HIGH](#) = 1U, [DIV_128_HIGH](#) = 2U, [DIV_256](#) = 3U, [DIV_512](#) = 4U, [DIV_1024](#) = 5U, [DIV_1280](#) = 6U, [DIV_1536](#) = 7U, [DIV_32_LOW](#) = 5U, [DIV_64_LOW](#) = 6U, [DIV_128_LOW](#) = 7U }
- enum [McuFLLFreq_t](#) { [MHZ_20_25](#), [MHZ_24](#), [MHZ_40_50](#), [MHZ_48](#), [MHZ_60_75](#), [MHZ_72](#), [MHZ_80_100](#), [MHZ_96](#) }
- enum [McuExtClkSrc_t](#) { [EXT_CLK](#), [OSC](#) }
- enum [McuIntRefSrc_t](#) { [SLOW_INT](#), [FAST_INT](#) }

Functions

- [McuConfig_t](#) const *const [Mcu_ConfigGet](#) (void)

Macro Definition Documentation

#define GetSystemClock() 48000000UL

The speed of the system clock in Hz

#define GetInstructionClock() ([GetSystemClock\(\)](#) / 2)

Instruction clock speed in Hz, system clock divided by 2

#define GetOscillatorFrequency() 8000000UL

The External Oscillator frequency in Hz

Enumeration Type Documentation

enum [McuMode_t](#)

Defines the oscillator frequency modes.

Enumerator

FEI_MODE FLL engaged internal

enum [McuFreqMode_t](#)

Defines the oscillator frequency modes.

Enumerator

LOW_POWER Oscillator is in low-power mode

HIGH_GAIN Oscillator is in high-frequency mode

enum [McuFreqRange_t](#)

Defines the oscillator frequency range selections.

Enumerator

LOW_RANGE Oscillator has low frequency range (<8 MHz)

HIGH_RANGE Oscillator has high frequency range

VERY_HIGH_RANGE Oscillator has very high frequency range

enum [McuDiv_t](#)

Defines the possible clock dividers.

Enumerator

DIV_1 Divide clock by 1

DIV_2 Divide clock by 2

DIV_4 Divide clock by 4

DIV_8 Divide clock by 8

DIV_16 Divide clock by 16

DIV_32 Divide Factor is 32

DIV_64 Divide Factor is 64

DIV_128 Divide Factor is 128

DIV_32_HIGH If Frequency range is set to high, Divide FLL clock by 32

DIV_64_HIGH If Frequency range is set to high, Divide FLL clock by 64

DIV_128_HIGH If Frequency range is set to high, Divide FLL clock by 128

DIV_256 If Frequency range is set to high, Divide FLL clock by 256

DIV_512 If Frequency range is set to high, Divide FLL clock by 512

DIV_1024 If Frequency range is set to high, Divide FLL clock by 1024

DIV_1280 If Frequency range is set to high, Divide FLL clock by 1280

DIV_1536 If Frequency range is set to high, Divide FLL clock by 1536

DIV_32_LOW If Frequency range is set to low, Divide FLL clock by 32

DIV_64_LOW If Frequency range is set to low, Divide FLL clock by 64

DIV_128_LOW If Frequency range is set to low, Divide FLL clock by 128

enum [McuFLLFreq_t](#)

Defines the DCO Frequency range selections.

Enumerator

MHZ_20_25 20-25 MHz (21 for internal clk src)

MHZ_24 24 MHz

MHZ_40_50 40-50 MHz (42 for internal clk src)

MHZ_48 48 MHz

MHZ_60_75 60-75 MHz (63 for internal clk src)

MHZ_72 72 MHz

MHZ_80_100 80-100 MHz (84 for internal clk src)

MHZ_96 96 MHz

enum [McuExtClkSrc_t](#)

Defines the external clock source selections.

Enumerator

EXT_CLK External reference clock requested.

OSC Oscillator requested.

enum [McuIntRefSrc_t](#)

Defines the Internal reference clock selections.

Enumerator

SLOW_INT Slow internal reference clock is selected.

FAST_INT Fast internal reference clock is selected.

Function Documentation

[McuConfig_t](#) const* const **Mcu_ConfigGet** (void)

Description:

This function return a pointer to the MCU configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const McuConfig_t * McuConfig = Mcu_ConfigGet();
2
3 Mcu_Init(McuConfig);
```

See also:

[Mcu_ConfigGet](#)

[Mcu_Init](#)

Mcu_FEEInit

Mcu_FBIInit

Mcu_FBEInit

Mcu_PBEInit

Mcu_PEEInit

Mcu_BLPInit

Mcu_BLPEInit

Mcu_TimeoutStart

Mcu_TimeoutCheck
Mcu_PllInit

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

pwm.c File Reference

The implementation for the pwm.

```
#include "pwm.h"
#include "constants.h"
```

Functions

- void [Pwm_Init](#) ([PwmConfig_t](#) const *const Config)
- void [Pwm_DutyCycleSet](#) ([PwmChannel_t](#) const Channel, uint16_t const DutyCycle)
- void [Pwm_FrequencySet](#) ([PwmChannel_t](#) const Channel, uint16_t const Frequency)
- void [Pwm_Enable](#) ([PwmChannel_t](#) const Channel)
- void [Pwm_Disable](#) ([PwmChannel_t](#) const Channel)
- void [Pwm_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Pwm_RegisterRead](#) (uint32_t const Address)
- void [Pwm_CallbackRegister](#) (PwmCallback_t const Function, TYPE(*CallbackFunction)(type))

Function Documentation

void Pwm_Init ([PwmConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the pwm based on the configuration table defined in pwm_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The Pwm peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3 Pwm_Init(PwmConfig);
```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)
- [Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

	Software		
--	----------	--	--

Date	Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

```
void Pwm_DutyCycleSet (PwmChannel\_t const Channel, uint16_t const DutyCycle)
```

Description:

This function is used to set the duty cycle of the pwm signal.

PRE-CONDITION: Pwm_Init must be called with valid configuration data

POST-CONDITION: The Pwm peripheral channels duty cycle is set

Parameters:

in	<i>Channel</i>	is the PWM channel to configure
in	<i>DutyCycle</i>	is the duty in a scale of 0 to 1000. 50.2% is 502.

Returns:

void

Example:

```

1      const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3      Pwm_Init(PwmConfig);
4  Pwm_DutyCycleSet(PWM_0, 502);

```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)
- [Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_FrequencySet ([PwmChannel_t](#) const *Channel*, uint16_t const *Frequency*)

Description:

This function is used to set the frequency of the pwm signal.

PRE-CONDITION: Pwm_Init must be called with valid configuration data

POST-CONDITION: The Pwm peripheral channels frequency is set

Parameters:

in	<i>Channel</i>	is the PWM channel to configure
in	<i>Frequency</i>	is the desired frequency in Hz

Returns:

void

Example:

```
1      const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3      Pwm_Init(PwmConfig);
4  Pwm_DutyCycleSet(PWM_0, 502);
5  Pwm_FrequencySet(100);
```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)
- [Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_Enable ([PwmChannel_t](#) const *Channel*)

Description:

This function is used to set enable the PWM channel

PRE-CONDITION: Pwm_Init must be called with valid configuration data

PRE-CONDITION: The Pwm peripheral channels frequency is set

PRE-CONDITION: The Pwm peripheral channels duty cycle is set

POST-CONDITION: The Pwm peripheral channel is enabled

Parameters:

in	<i>Channel</i>	is the PWM channel to configure
----	----------------	---------------------------------

Returns:

void

Example:

```
1      const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3      Pwm_Init(PwmConfig);
4  Pwm_DutyCycleSet(PWM_0, 502);
5  Pwm_FrequencySet(100);
6  Pwm_Enable(PWM_0);
```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)
- [Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_Disable ([PwmChannel_t](#) const *Channel*)

Description:

This function is used to set disable a PWM channel

PRE-CONDITION: Pwm_Init must be called with valid configuration data

POST-CONDITION: The Pwm peripheral channel is disabled

Parameters:

in	<i>Channel</i>	is the PWM channel to configure
----	----------------	---------------------------------

Returns:

void

Example:

```
1      const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3      Pwm_Init(PwmConfig);
4 Pwm_DutyCycleSet(PWM_0, 502);
5 Pwm_FrequencySet(100);
6 Pwm_Enable(PWM_0);
7 ...
8 Pwm_Disable(PWM_0);
```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)
- [Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Pwm register. The function should be used to access specialized functionality in the Pwm peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Pwm register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Pwm peripheral map
in	<i>Value</i>	is the value to set the Pwm register to

Returns:

void

Example:

```
1 Pwm_RegisterWrite(0x1000, 0x15);
```

See also:

- [Pwm_ConfigGet](#)

[Pwm_Init](#)
[Pwm_DutyCycleSet](#)
[Pwm_FrequencySet](#)
[Pwm_Enable](#)
[Pwm_Disable](#)
[Pwm_RegisterWrite](#)
[Pwm_RegisterRead](#)
[Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Pwm_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Pwm register. The function should be used to access specialized functionality in the Pwm peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Pwm register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Pwm register to read
----	----------------	--

Returns:

The current value of the Pwm register.

Example:

```
1 PwmValue = Pwm_RegisterRead(0x1000);
```

See also:

[Pwm_ConfigGet](#)
[Pwm_Init](#)
[Pwm_DutyCycleSet](#)
[Pwm_FrequencySet](#)
[Pwm_Enable](#)
[Pwm_Disable](#)
[Pwm_RegisterWrite](#)
[Pwm_RegisterRead](#)
[Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
------	------------------	----------	-------------

09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_CallbackRegister (PwmCallback_t const *Function*, TYPE*)(type) *CallbackFunction*)

Description:

This function is used to set the callback functions of the adc driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The AdcCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 AdcCallback_t Adc_Function = ADC_SAMPLE_COMPLETE;  
2  
3 Adc_CallbackRegister(Adc_Function, Adc_SampleAverage);
```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)
- [Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

pwm.h File Reference

The interface definition for the pwm.

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

Functions

- void [Pwm_Init](#) ([PwmConfig_t](#) const *const Config)
- void [Pwm_DutyCycleSet](#) ([PwmChannel_t](#) const Channel, uint16_t const DutyCycle)
- void [Pwm_FrequencySet](#) ([PwmChannel_t](#) const Channel, uint16_t const Frequency)
- void [Pwm_Enable](#) ([PwmChannel_t](#) const Channel)
- void [Pwm_Disable](#) ([PwmChannel_t](#) const Channel)
- void [Pwm_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Pwm_RegisterRead](#) (uint32_t const Address)
- void [Pwm_CallbackRegister](#) (PwmCallback_t const Function, TYPE(*CallbackFunction)(type))

Detailed Description

This is the header file for the definition of the interface for the watchdog timer.

Function Documentation

void [Pwm_Init](#) ([PwmConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the pwm based on the configuration table defined in pwm_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The Pwm peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3 Pwm_Init(PwmConfig);
```

See also:

[Pwm_ConfigGet](#)

[Pwm_Init](#)

[Pwm_DutyCycleSet](#)

[Pwm_FrequencySet](#)

[Pwm_Enable](#)

[Pwm_Disable](#)

[Pwm_RegisterWrite](#)

[Pwm_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_DutyCycleSet ([PwmChannel_t](#) const *Channel*, uint16_t const *DutyCycle*)

Description:

This function is used to set the duty cycle of the pwm signal.

PRE-CONDITION: Pwm_Init must be called with valid configuration data

POST-CONDITION: The Pwm peripheral channels duty cycle is set

Parameters:

in	<i>Channel</i>	is the PWM channel to configure
in	<i>DutyCycle</i>	is the duty in a scale of 0 to 1000. 50.2% is 502.

Returns:

void

Example:

```
1      const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3      Pwm_Init(PwmConfig);
4  Pwm_DutyCycleSet(PWM_0, 502);
```

See also:

[Pwm_ConfigGet](#)

[Pwm_Init](#)

[Pwm_DutyCycleSet](#)

[Pwm_FrequencySet](#)

[Pwm_Enable](#)

[Pwm_Disable](#)

[Pwm_RegisterWrite](#)

[Pwm_RegisterRead](#)

[Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_FrequencySet ([PwmChannel_t](#) const *Channel*, uint16_t const *Frequency*)

Description:

This function is used to set the frequency of the pwm signal.

PRE-CONDITION: Pwm_Init must be called with valid configuration data

POST-CONDITION: The Pwm peripheral channels frequency is set

Parameters:

in	<i>Channel</i>	is the PWM channel to configure
in	<i>Frequency</i>	is the desired frequency in Hz

Returns:

void

Example:

```
1      const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3      Pwm_Init(PwmConfig);
4  Pwm_DutyCycleSet(PWM_0, 502);
5  Pwm_FrequencySet(100);
```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)
- [Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_Enable ([PwmChannel_t](#) const *Channel*)

Description:

This function is used to set enable the PWM channel

PRE-CONDITION: Pwm_Init must be called with valid configuration data

PRE-CONDITION: The Pwm peripheral channels frequency is set

PRE-CONDITION: The Pwm peripheral channels duty cycle is set

POST-CONDITION: The Pwm peripheral channel is enabled

Parameters:

in	<i>Channel</i>	is the PWM channel to configure
----	----------------	---------------------------------

Returns:

void

Example:

```
1      const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3      Pwm_Init(PwmConfig);
4  Pwm_DutyCycleSet(PWM_0, 502);
5  Pwm_FrequencySet(100);
6  Pwm_Enable(PWM_0);
```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)
- [Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_Disable ([PwmChannel_t](#) const *Channel*)

Description:

This function is used to set disable a PWM channel

PRE-CONDITION: Pwm_Init must be called with valid configuration data

POST-CONDITION: The Pwm peripheral channel is disabled

Parameters:

in	<i>Channel</i>	is the PWM channel to configure
----	----------------	---------------------------------

Returns:

void

Example:

```
1      const PwmConfig_t *PwmConfig = Pwm_ConfigGet();
2
3      Pwm_Init(PwmConfig);
```

```

4 Pwm_DutyCycleSet(PWM_0, 502);
5 Pwm_FrequencySet(100);
6 Pwm_Enable(PWM_0);
7 ...
8 Pwm_Disable(PWM_0);

```

See also:

[Pwm_ConfigGet](#)
[Pwm_Init](#)
[Pwm_DutyCycleSet](#)
[Pwm_FrequencySet](#)
[Pwm_Enable](#)
[Pwm_Disable](#)
[Pwm_RegisterWrite](#)
[Pwm_RegisterRead](#)
[Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Pwm register. The function should be used to access specialized functionality in the Pwm peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Pwm register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Pwm peripheral map
in	<i>Value</i>	is the value to set the Pwm register to

Returns:

void

Example:

```

1 Pwm_RegisterWrite(0x1000, 0x15);

```

See also:

[Pwm_ConfigGet](#)
[Pwm_Init](#)
[Pwm_DutyCycleSet](#)
[Pwm_FrequencySet](#)
[Pwm_Enable](#)

[Pwm_Disable](#)
[Pwm_RegisterWrite](#)
[Pwm_RegisterRead](#)
[Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Pwm_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Pwm register. The function should be used to access specialized functionality in the Pwm peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Pwm register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Pwm register to read
----	----------------	--

Returns:

The current value of the Pwm register.

Example:

```
1 PwmValue = Pwm_RegisterRead(0x1000);
```

See also:

[Pwm_ConfigGet](#)
[Pwm_Init](#)
[Pwm_DutyCycleSet](#)
[Pwm_FrequencySet](#)
[Pwm_Enable](#)
[Pwm_Disable](#)
[Pwm_RegisterWrite](#)
[Pwm_RegisterRead](#)
[Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Pwm_CallbackRegister (PwmCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the adc driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The AdcCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 AdcCallback_t Adc_Function = ADC_SAMPLE_COMPLETE;  
2  
3 Adc_CallbackRegister(Adc_Function, Adc_SampleAverage);
```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)
- [Pwm_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

pwm_cfg.c File Reference

This module contains the configuration for the pwm module.

```
#include "pwm_cfg.h"
#include "constants.h"
```

Functions

- [PwmConfig_t](#) const *const [Pwm_ConfigGet](#) (void)

Variables

- const [PwmConfig_t](#) [PwmConfig](#) []

Function Documentation

[PwmConfig_t](#) const* const [Pwm_ConfigGet](#) (void)

Description:

This function return a pointer to the PWM configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const PwmConfig_t * PwmConfig = Pwm_ConfigGet();
2
3 Pwm_Init(PwmConfig);
```

See also:

- [Pwm_ConfigGet](#)
- [Pwm_Init](#)
- [Pwm_DutyCycleSet](#)
- [Pwm_FrequencySet](#)
- [Pwm_Enable](#)
- [Pwm_Disable](#)
- [Pwm_RegisterWrite](#)
- [Pwm_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Variable Documentation

const [PwmConfig_t](#) PwmConfig[]

```
Initial value:=
{

    { PWM0\_0,      DISABLED,      DISABLED,      500      },
    { PWM0\_1,      DISABLED,      DISABLED,      500      },
    { PWM0\_2,      DISABLED,      DISABLED,      500      },
    { PWM0\_3,      DISABLED,      DISABLED,      500      },
    { PWM0\_4,      DISABLED,      DISABLED,      500      },
    { PWM0\_5,      DISABLED,      DISABLED,      500      },
    { PWM1\_0,      DISABLED,      DISABLED,      500      },
    { PWM1\_1,      DISABLED,      DISABLED,      500      },
    { PWM2\_0,      ENABLED,      DISABLED,      500      },
    { PWM2\_1,      ENABLED,      DISABLED,      500      },
}
```

This configuration table is used to configure the behavior and function of the PWM. The channels are defined in [pwm_cfg.h](#). The configuration consists of

- o PWM Channel - Specify the name of the pwm. This label must be defined in the Pwm_ChannelType enumeration.
- o Pwm Mode - The mode of this PWM channel. Edge aligned or center aligned is determined by the counting mode of the timer channel. UP_COUNT = Edge aligned PWM. UP_DOWN = center aligned PWM. DISABLED - PWM signal is disabled. ENABLED - PWM signal is enabled.
- o Interrupt Enabled - This sets whether the capture/compare interrupt is enabled. DISABLED - Sets the interrupt enable bit low ENABLED - Sets the interrupt enable bit high
- o Duty Cycle - This sets the duty cycle percentage of the pwm channel. Percentage is out of 1000, so 1000 == 100%, 500 = 50%

pwm_cfg.h File Reference

This module contains the configuration interface for pwm.

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

Data Structures

- struct [PwmConfig_t](#)

Enumerations

- enum [PwmChannel_t](#) { [PWM0_0](#), [PWM0_1](#), [PWM0_2](#), [PWM0_3](#), [PWM0_4](#), [PWM0_5](#), [PWM1_0](#), [PWM1_1](#), [PWM2_0](#), [PWM2_1](#), [NUM_PWM_CHANNELS](#) }

Functions

- [PwmConfig_t](#) const *const [Pwm_ConfigGet](#) (void)

Enumeration Type Documentation

enum [PwmChannel_t](#)

Defines the Pwm module channel names

Enumerator

PWM0_0 TB1 Signal

PWM0_1 TB1 Signal

PWM0_2 TB2 Signal

PWM0_3 TB3 Signal

PWM0_4 TB4 Signal

PWM0_5 TB5 Signal

PWM1_0 TB1 Signal

PWM1_1 TB1 Signal

PWM2_0 TB1 Signal

PWM2_1 TB1 Signal

NUM_PWM_CHANNELS Number of output compare channels on this microcontroller

Function Documentation

[PwmConfig_t](#) const* const [Pwm_ConfigGet](#) (void)

Description:

This function return a pointer to the PWM configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const PwmConfig_t * PwmConfig = Pwm_ConfigGet();  
2  
3 Pwm_Init(PwmConfig);
```

See also:

[Pwm_ConfigGet](#)

[Pwm_Init](#)

[Pwm_DutyCycleSet](#)

[Pwm_FrequencySet](#)

[Pwm_Enable](#)

[Pwm_Disable](#)

[Pwm_RegisterWrite](#)

[Pwm_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

spi.c File Reference

The implementation for the spi.

```
#include "spi.h"
#include "dio.h"
```

Functions

- void [Spi_BaudRateSet](#) (const [SpiConfig_t](#) Config)
 - void [Spi_SSSet](#) (const [SpiConfig_t](#) Config)
 - void [Spi_CalcDelay](#) (const [SpiConfig_t](#) Config)
 - void [Spi_Init](#) ([SpiConfig_t](#) const *const Config)
 - void [Spi_DeInit](#) (SpiChannel_t Channel)
 - void [Spi_Transfer](#) ([SpiTransfer_t](#) const *const Config)
 - void [Spi_Setup](#) ([SpiTransfer_t](#) const *const Config)
 - void [Spi_ChipSelectSet](#) ([SpiTransfer_t](#) const *const Config)
 - void [Spi_ChipSelectClear](#) ([SpiTransfer_t](#) const *const Config)
 - void [Spi_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
 - uint32_t [Spi_RegisterRead](#) (uint32_t const Address)
 - void [Spi_CallbackRegister](#) (SpiCallback_t const Function, TYPE(*CallbackFunction)(type))
-

Function Documentation

void [Spi_BaudRateSet](#) ([SpiConfig_t](#) const *Config*) [*inline*]

Description:

This function is used to set the baud rate divider and prescaler register values in order to get a SPI clock frequency as close as possible to the desired speed. This is an inline function and if not inlined should be declared as static.

PRE-CONDITION: [Spi_Init](#) must be called with valid configuration data

PRE-CONDITION: [SpiTransfer_t](#) must be configured for the specified device

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Slave device will be de-selected via gpio line.

Parameters:

in	<i>Config</i>	is a configure structure describing the data transfer that occur.
----	---------------	---

Returns:

void

Example:

```
1 Spi_BaudRateSet (Config);
```

See also:

[Spi_ConfigGet](#)

[Spi_Init](#)

[Spi_DeInit](#)

[Spi_Transfer](#)

[Spi_Setup](#)

[Spi_ChipSelectSet](#)

[Spi_ChipSelectClear](#)

[Spi_BaudRateSet](#)

[Spi_SSSet](#)
[Spi_CalcDelay](#)
[Spi_RegisterWrite](#)
[Spi_RegisterRead](#)
[Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

```
void Spi_SSSet (SpiConfig\_t const Config) [inline]
```

Description:

This function is used to set the slave select setting for a SPI channel. This is an inline function and if not inlined should be declared as static.

PRE-CONDITION: Spi_Init must be called with valid configuration data

PRE-CONDITION: [SpiTransfer_t](#) must be configured for the specified device

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Slave device will be de-selected via gpio line.

Parameters:

in	<i>Config</i>	is a configure structure describing the data transfer that occur.
----	---------------	---

Returns:

void

Example:

```
1 Spi_SSSet (Config);
```

See also:

[Spi_ConfigGet](#)
[Spi_Init](#)
[Spi_DeInit](#)
[Spi_Transfer](#)
[Spi_Setup](#)
[Spi_ChipSelectSet](#)
[Spi_ChipSelectClear](#)
[Spi_BaudRateSet](#)
[Spi_SSSet](#)
[Spi_CalcDelay](#)
[Spi_RegisterWrite](#)
[Spi_RegisterRead](#)
[Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

```
void Spi_CalcDelay (SpiConfig\_t const Config) [inline]
```

Description:

This function is used to calculate the SPI transfer delay for a channel. This is an inline function and if not inlined should be declared as static.

PRE-CONDITION: Spi_Init must be called with valid configuration data

PRE-CONDITION: [SpiTransfer_t](#) must be configured for the specified device

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Slave device will be de-selected via gpio line.

Parameters:

in	<i>Config</i>	is a configure structure describing the data transfer that occur.
----	---------------	---

Returns:

void

Example:

```
1 Spi_CalcDelay(Config);
```

See also:

[Spi_ConfigGet](#)

[Spi_Init](#)

[Spi_DeInit](#)

[Spi_Transfer](#)

[Spi_Setup](#)

[Spi_ChipSelectSet](#)

[Spi_ChipSelectClear](#)

[Spi_BaudRateSet](#)

[Spi_SSSet](#)

[Spi_CalcDelay](#)

[Spi_RegisterWrite](#)

[Spi_RegisterRead](#)

[Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

	Software		
--	----------	--	--

Date	Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_Init ([SpiConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the Spi based on the configuration table defined in spi_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```

1 const SpiConfig_t *SpiConfig = Spi_ConfigGet();
2
3 Spi_Init(SpiConfig);

```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_DeInit](#)
- [Spi_Transfer](#)
- [Spi_Setup](#)
- [Spi_ChipSelectSet](#)
- [Spi_ChipSelectClear](#)
- [Spi_BaudRateSet](#)
- [Spi_SSSet](#)
- [Spi_CalcDelay](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
			Interface

void Spi_DeInit (SpiChannel_t Channel)

Description:

This function is used to deinitialize the SPI channel. All registers are cleared to the RESET values.

PRE-CONDITION: Spi_Init has been called previously

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The SPI peripheral is no longer intialized

Parameters:

in	<i>Channel</i>	is the SPI channel i.e. 0, 1, etc that will be deinitialized.
----	----------------	---

Returns:

void

Example:

```
1      const SpiConfig_t *SpiConfig = Spi_ConfigGet();
2
3      Spi_Init(SpiConfig);
4 Spi_DeInit(SPI0);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_DeInit](#)
- [Spi_Transfer](#)
- [Spi_Setup](#)
- [Spi_ChipSelectSet](#)
- [Spi_ChipSelectClear](#)
- [Spi_BaudRateSet](#)
- [Spi_SSSet](#)
- [Spi_CalcDelay](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_Transfer ([SpiTransfer_t](#) const *const *Config*)

Description:

This function is used to initialize a data transfer on the SPI bus.

PRE-CONDITION: Spi_Init must be called with valid configuration data

PRE-CONDITION: [SpiTransfer_t](#) must be configured for the specified device

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Communication will occur based on the configuration structure.

Parameters:

in	<i>Config</i>	is a configure structure describing the data transfer that occur.
----	---------------	---

Returns:

void

Example:

```
1      const SpiConfig_t *SpiConfig = Spi_ConfigGet();
2
3      Spi_Init(SpiConfig);
4  Spi_Transfer(AccelerometerConfig);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_DeInit](#)
- [Spi_Transfer](#)
- [Spi_Setup](#)
- [Spi_ChipSelectSet](#)
- [Spi_ChipSelectClear](#)
- [Spi_BaudRateSet](#)
- [Spi_SSSet](#)
- [Spi_CalcDelay](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_Setup ([SpiTransfer_t](#) const *const *Config*) [inline]

Description:

This function is used to setup the polarity, phase and endianness of a data transfer. This is an inline function and if not inlined should be declared as static.

PRE-CONDITION: Spi_Init must be called with valid configuration data

PRE-CONDITION: [SpiTransfer_t](#) must be configured for the specified device

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Peripheral registers will be setup for the transfer.

Parameters:

in	<i>Config</i>	is a configure structure describing the data transfer that occur.
----	---------------	---

Returns:

void

Example:

```
1 Spi_Setup(Config);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_DeInit](#)
- [Spi_Transfer](#)
- [Spi_Setup](#)
- [Spi_ChipSelectSet](#)
- [Spi_ChipSelectClear](#)
- [Spi_BaudRateSet](#)
- [Spi_SSSet](#)
- [Spi_CalcDelay](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_ChipSelectSet ([SpiTransfer_t](#) const *const *Config*) [inline]

Description:

This function is used to select a slave device. It toggles an I/O line into the active state of the slave device. This is an inline function and if not inlined should be declared as static.

PRE-CONDITION: Spi_Init must be called with valid configuration data

PRE-CONDITION: [SpiTransfer_t](#) must be configured for the specified device

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Slave device will be selected via gpio line.

Parameters:

in	<i>Config</i>	is a configure structure describing the data transfer that occur.
----	---------------	---

Returns:

void

Example:

```
1 Spi_ChipSelectSet(Config);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_DeInit](#)
- [Spi_Transfer](#)
- [Spi_Setup](#)
- [Spi_ChipSelectSet](#)
- [Spi_ChipSelectClear](#)
- [Spi_BaudRateSet](#)
- [Spi_SSSet](#)
- [Spi_CalcDelay](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_ChipSelectClear ([SpiTransfer_t](#) const *const *Config*)[inline]

Description:

This function is used to de-select a slave device. It toggles an I/O line into the inactive state. This is an inline function and if not inlined should be declared as static.

PRE-CONDITION: Spi_Init must be called with valid configuration data

PRE-CONDITION: [SpiTransfer_t](#) must be configured for the specified device

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Slave device will be de-selected via gpio line.

Parameters:

in	<i>Config</i>	is a configure structure describing the data transfer that occur.
----	---------------	---

Returns:

void

Example:

```
1 Spi_ChipSelectClear(Config);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_DeInit](#)
- [Spi_Transfer](#)
- [Spi_Setup](#)
- [Spi_ChipSelectSet](#)
- [Spi_ChipSelectClear](#)
- [Spi_BaudRateSet](#)
- [Spi_SSSet](#)
- [Spi_CalcDelay](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Spi register. The function should be used to access specialized functionality in the Spi peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Spi register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Spi peripheral map
in	<i>Value</i>	is the value to set the Spi register to

Returns:

void

Example:

```
1 Spi_RegisterWrite(0x1000, 0x15);
```

See also:

[Spi_ConfigGet](#)
[Spi_Init](#)
[Spi_DeInit](#)
[Spi_Transfer](#)
[Spi_Setup](#)
[Spi_ChipSelectSet](#)
[Spi_ChipSelectClear](#)
[Spi_BaudRateSet](#)
[Spi_SSSet](#)
[Spi_CalcDelay](#)
[Spi_RegisterWrite](#)
[Spi_RegisterRead](#)
[Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Spi_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Spi register. The function should be used to access specialized functionality in the Spi peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Spi register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Spi register to read
----	----------------	--

Returns:

The current value of the Spi register.

Example:

```
1 SpiValue = Spi_RegisterRead(0x1000);
```

See also:

[Spi_ConfigGet](#)
[Spi_Init](#)
[Spi_ChannelRead](#)
[Spi_ChannelWrite](#)
[Spi_ChannelToggle](#)
[Spi_ChannelModeSet](#)
[Spi_ChannelDirectionSet](#)
[Spi_RegisterWrite](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_CallbackRegister (SpiCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the spi driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The SpiCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 SpiCallback_t Spi_Function = SPI_SAMPLE_COMPLETE;  
2  
3 Spi_CallbackRegister(Spi_Function, Spi_SampleAverage);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_ChannelRead](#)
- [Spi_ChannelWrite](#)
- [Spi_ChannelToggle](#)
- [Spi_ChannelModeSet](#)
- [Spi_ChannelDirectionSet](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created

11/10/2015	1.0.0	JWB	Interface Released
------------	-------	-----	--------------------

spi.h File Reference

The interface definition for spi.

```
#include <stdint.h>
#include "spi_cfg.h"
#include "dio.h"
```

Data Structures

- struct [SpiTransfer_t](#)

Enumerations

- enum [SpiPolarity_t](#) { [POLARITY_LOW](#), [POLARITY_HIGH](#) }
- enum [SpiPhase_t](#) { [PHASE_HIGH](#), [PHASE_LOW](#) }
- enum [SpiBitOrder_t](#) { [LSB_FIRST](#), [MSB_FIRST](#) }
- enum [SpiChipSelect_t](#) { [CS_ACTIVE_LOW](#), [CS_ACTIVE_HIGH](#) }

Functions

- void [Spi_Init](#) ([SpiConfig_t](#) const *const Config)
- void [Spi_DeInit](#) (SpiChannel_t const Channel)
- void [Spi_Transfer](#) ([SpiTransfer_t](#) const *const Config)
- void [Spi_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Spi_RegisterRead](#) (uint32_t const Address)
- void [Spi_CallbackRegister](#) (SpiCallback_t const Function, TYPE(*CallbackFunction)(type))

Detailed Description

This is the header file for the definition of the interface for the SPI peripheral.

Enumeration Type Documentation

enum [SpiPolarity_t](#)

Defines the clock polarity options.

Enumerator

- POLARITY_LOW*** Idle clock state is low
- POLARITY_HIGH*** Idle clock state is high

enum [SpiPhase_t](#)

Defines the clock phase.

Enumerator

- PHASE_HIGH*** Sample from leading clock edge
- PHASE_LOW*** Sample from trailing clock edge

enum [SpiBitOrder_t](#)

Defines the bit ordering for a Spi Transfer.

Enumerator

LSB_FIRST Least significant bit sent first

MSB_FIRST Most significant bit sent first

enum [SpiChipSelect_t](#)

Defines the state of the CS when data is being transferred to the device

Enumerator

CS_ACTIVE_LOW Chip select active low

CS_ACTIVE_HIGH Chip select active high

Function Documentation

void [Spi_Init](#) ([SpiConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the Spi based on the configuration table defined in spi_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const SpiConfig_t *SpiConfig = Spi_ConfigGet();
2
3 Spi_Init(SpiConfig);
```

See also:

[Spi_ConfigGet](#)

[Spi_Init](#)

[Spi_DeInit](#)

[Spi_Transfer](#)

[Spi_Setup](#)

[Spi_ChipSelectSet](#)

[Spi_ChipSelectClear](#)

[Spi_BaudRateSet](#)

[Spi_SSSet](#)

[Spi_CalcDelay](#)

[Spi_RegisterWrite](#)

[Spi_RegisterRead](#)

[Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_DeInit (SpiChannel_t Channel)

Description:

This function is used to deinitialize the SPI channel. All registers are cleared to the RESET values.

PRE-CONDITION: Spi_Init has been called previously

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The SPI peripheral is no longer intialized

Parameters:

in	Channel	is the SPI channel i.e. 0, 1, etc that will be deinitialized.
----	---------	---

Returns:

void

Example:

```
1      const SpiConfig_t *SpiConfig = Spi_ConfigGet();
2
3      Spi_Init(SpiConfig);
4 Spi_DeInit(SPI0);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_DeInit](#)
- [Spi_Transfer](#)
- [Spi_Setup](#)
- [Spi_ChipSelectSet](#)
- [Spi_ChipSelectClear](#)
- [Spi_BaudRateSet](#)
- [Spi_SSSet](#)
- [Spi_CalcDelay](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_Transfer (SpiTransfer_t const *const Config)

Description:

This function is used to initialize a data transfer on the SPI bus.

PRE-CONDITION: Spi_Init must be called with valid configuration data

PRE-CONDITION: SpiTransfer_t must be configured for the specified device

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Communication will occur based on the configuration structure.

Parameters:

in	Config	is a configure structure describing the data transfer that occur.
----	--------	---

Returns:

void

Example:

```
1      const SpiConfig_t *SpiConfig = Spi_ConfigGet();
2
3      Spi_Init(SpiConfig);
4  Spi_Transfer(AccelerometerConfig);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_DeInit](#)
- [Spi_Transfer](#)
- [Spi_Setup](#)
- [Spi_ChipSelectSet](#)
- [Spi_ChipSelectClear](#)
- [Spi_BaudRateSet](#)
- [Spi_SSSet](#)
- [Spi_CalcDelay](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_RegisterWrite (uint32_t const Address, uint32_t const Value)

Description:

This function is used to directly address and modify a Spi register. The function should be used to access specialized functionality in the Spi peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Spi register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Spi peripheral map
in	<i>Value</i>	is the value to set the Spi register to

Returns:

void

Example:

```
1 Spi_RegisterWrite(0x1000, 0x15);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_DeInit](#)
- [Spi_Transfer](#)
- [Spi_Setup](#)
- [Spi_ChipSelectSet](#)
- [Spi_ChipSelectClear](#)
- [Spi_BaudRateSet](#)
- [Spi_SSSet](#)
- [Spi_CalcDelay](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Spi_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Spi register. The function should be used to access specialized functionality in the Spi peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Spi register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Spi register to read
----	----------------	--

Returns:

The current value of the Spi register.

Example:

```
1 SpiValue = Spi_RegisterRead(0x1000);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_ChannelRead](#)
- [Spi_ChannelWrite](#)
- [Spi_ChannelToggle](#)
- [Spi_ChannelModeSet](#)
- [Spi_ChannelDirectionSet](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)
- [Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Spi_CallbackRegister (SpiCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the spi driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The SpiCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 SpiCallback_t Spi_Function = SPI_SAMPLE_COMPLETE;
2
3 Spi_CallbackRegister(Spi_Function, Spi_SampleAverage);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_ChannelRead](#)

Spi_ChannelWrite
Spi_ChannelToggle
Spi_ChannelModeSet
Spi_ChannelDirectionSet
[Spi_RegisterWrite](#)
[Spi_RegisterRead](#)
[Spi_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

spi_cfg.c File Reference

This module contains the configuration for the spi module.

```
#include "spi_cfg.h"
#include "constants.h"
```

Functions

- [SpiConfig_t](#) const *const [Spi_ConfigGet](#) (void)

Variables

- const [SpiConfig_t](#) [SpiConfig](#) []

Detailed Description

This module contains the configuration interface for spi.

Function Documentation

[SpiConfig_t](#) const* const [Spi_ConfigGet](#) (void)

Description:

This function return a pointer to the Spi configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const SpiConfig_t * SpiConfig = Spi_ConfigGet();
2
3 Spi_Init(SpiConfig);
```

See also:

- [Spi_ConfigGet](#)
- [Spi_Init](#)
- [Spi_ChannelRead](#)
- [Spi_ChannelWrite](#)
- [Spi_ChannelToggle](#)
- [Spi_ChannelModeSet](#)
- [Spi_ChannelDirectionSet](#)
- [Spi_RegisterWrite](#)
- [Spi_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Variable Documentation

const [SpiConfig_t](#) SpiConfig[]

```
Initial value:=
{
    {SPI_0,      ENABLED,      MASTER,      GPIO,      DISABLED,
ENABLED,      250000UL    },
    {SPI_1,      DISABLED,      MASTER,      GPIO,      DISABLED,
ENABLED,      250000UL    },
}
```

This configuration table is used to configure the behavior and function of the spi channels. The channels are defined in [spi_cfg.h](#). The configuration consists of

- o SPI Channel - Specify the name of the spi channel. This label must be defined in the Spi_ChannelType enumeration.
- o Channel Enable - Specify whether the Spi channel is enabled.
- o Master Mode - Set the Spi channel to master or slave mode
- o Slave Select Pin Mode - Select the SS Pin function. GPIO - SS pin is general purpose I/O, not spi. SS_INPUT - Slave select is an input. SS_OUTPUT - SS pin is an output.
- o Bidirectional Mode - Enable or disable bidirectional mode. A single pin is used for both input and output. MOSI for master, MISO for slave.
- o Wait Mode - Enable or disable spi clock operation when CPU is in wait mode.
- o Clock Frequency - Determines the SPI clock frequency (in Hz).

tmr.c File Reference

The implementation for the timer.

```
#include "tmr_cfg.h"
#include "constants.h"
#include "mcu_cfg.h"
```

Macros

- `#define TMR_PERIOD_DIV 1000000UL`

Functions

- void [Tmr_ClkDivSet](#) (uint8_t Prescaler, uint8_t Channel)
- void [Tmr_Init](#) ([TmrConfig_t](#) const *const Config)
- void [Tmr_Enable](#) ([TmrRegister_t](#) const Channel, [TmrClockMode_t](#) const Mode)
- void [Tmr_Disable](#) ([TmrRegister_t](#) const Channel)
- void [Tmr_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Tmr_RegisterRead](#) (uint32_t const Address)
- void [Tmr_CallbackRegister](#) (TmrCallback_t const Function, TYPE(*CallbackFunction)(type))

Macro Definition Documentation

```
#define TMR_PERIOD_DIV 1000000UL
    Divider value used in timer period register calculation
```

Function Documentation

```
void Tmr_ClkDivSet (uint8_t const Prescaler, uint8_t const Channel)[inline]
```

Description:

This function is used to set the clock divider for the timer module.

PRE-CONDITION: Tmer_Init must be called

PRE-CONDITION: MCU clocks initialized

PRE-CONDITION: Timer clock enabled

POST-CONDITION: The TMR peripheral is setup with the clock settings.

Parameters:

in	<i>Prescaler</i>	- the prescaler to set the clock to
in	<i>Channel</i>	- the timer channel to configure

Returns:

void

Example:

```
1 // Set the chosen timer clock input divider
2 Tmr_ClkDivSet(Config[i].ClkPrescaler , i);
```

See also:

- [Tmr_ConfigGet](#)
- [Tmr_Init](#)

[Tmr_Enable](#)
[Tmr_Disable](#)
[Tmr_RegisterWrite](#)
[Tmr_RegisterRead](#)
[Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Tmr_Init ([TmrConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the Tmr based on the configuration table defined in tmr_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: MCU clocks initialized

PRE-CONDITION: Timer clock enabled

POST-CONDITION: The TMR peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const TmrConfig_t *TmrConfig = Tmr_ConfigGet();
2
3 Tmr_Init(TmrConfig);
```

See also:

[Tmr_ConfigGet](#)
[Tmr_Init](#)
[Tmr_Enable](#)
[Tmr_Disable](#)
[Tmr_RegisterWrite](#)
[Tmr_RegisterRead](#)
[Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created

void Tmr_Enable ([TmrRegister_t](#) const *Channel*, [TmrClockMode_t](#) const *Mode*)

Description:

This function enables and starts the specified timer channel.

PRE-CONDITION: Tmer_Init must be called

PRE-CONDITION: MCU clocks initialized

PRE-CONDITION: Timer clock enabled

POST-CONDITION: The timer channel is enabled in the specified mode

Parameters:

in	<i>Channel</i>	- TmrRegister_t, The timer channel to set.
in	<i>Mode</i>	- TmrRegister_t, The timer channel to set.

Returns:

void

Example:

```
1      const TmrConfig_t *TmrConfig = Tmr_ConfigGet();
2
3      Tmr_Init(TmrConfig);
4  Tmr_Enable(TIMER_0, MODE_COUNT_UP);
```

See also:

- [Tmr_ConfigGet](#)
- [Tmr_Init](#)
- [Tmr_Enable](#)
- [Tmr_Disable](#)
- [Tmr_RegisterWrite](#)
- [Tmr_RegisterRead](#)
- [Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Tmr_Disable ([TmrRegister_t](#) const *Channel*)

Description:

This function disables the specified timer channel.

PRE-CONDITION: Tmer_Init must be called

PRE-CONDITION: MCU clocks initialized

PRE-CONDITION: Timer clock enabled

POST-CONDITION: The timer channel is disabled.

Parameters:

in	<i>Channel</i>	- TmrRegister_t, The timer channel to set.
----	----------------	--

Returns:

void

Example:

```
1      const TmrConfig_t *TmrConfig = Tmr_ConfigGet();
2
3      Tmr_Init(TmrConfig);
4  Tmr_Enable(TIMER_0, MODE_COUNT_UP);
5  ...
6  Tmr_Disable(TIMER_0);
```

See also:

- [Tmr_ConfigGet](#)
- [Tmr_Init](#)
- [Tmr_Enable](#)
- [Tmr_Disable](#)
- [Tmr_RegisterWrite](#)
- [Tmr_RegisterRead](#)
- [Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Tmr_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Tmr register. The function should be used to access specialized functionality in the Tmr peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Tmr register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Tmr peripheral map
in	<i>Value</i>	is the value to set the Tmr register to

Returns:

void

Example:

```
1 Tmr_RegisterWrite(0x1000, 0x15);
```

See also:

[Tmr_ConfigGet](#)

[Tmr_Init](#)

[Tmr_Enable](#)

[Tmr_Disable](#)

[Tmr_RegisterWrite](#)

[Tmr_RegisterRead](#)

[Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Tmr_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Tmr register. The function should be used to access specialized functionality in the Tmr peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Tmr register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Tmr register to read
----	----------------	--

Returns:

The current value of the Tmr register.

Example:

```
1 TmrValue = Tmr_RegisterRead(0x1000);
```

See also:

[Tmr_ConfigGet](#)

[Tmr_Init](#)

[Tmr_Enable](#)

[Tmr_Disable](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Tmr_CallbackRegister (TmrCallback_t const *Function*, TYPE(*)*(type)* *CallbackFunction*)

Description:

This function is used to set the callback functions of the tmr driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The TmrCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```

1 TmrCallback_t Tmr_Function = TMR_SAMPLE_COMPLETE;
2
3 Tmr_CallbackRegister(Tmr_Function, Tmr_SampleAverage);

```

See also:

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

tmr.h File Reference

The interface definition for the timer.

```
#include "tmr_cfg.h"
```

Functions

- void [Tmr_Init](#) ([TmrConfig_t](#) const *const Config)
- void [Tmr_Enable](#) ([TmrRegister_t](#) const Channel, [TmrClockMode_t](#) const Mode)
- void [Tmr_Disable](#) ([TmrRegister_t](#) const Channel)
- void [Tmr_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Tmr_RegisterRead](#) (uint32_t const Address)
- void [Tmr_CallbackRegister](#) (TmrCallback_t const Function, TYPE(*CallbackFunction)(type))

Detailed Description

This is the header file for the definition of the interface for the timer.

Function Documentation

void Tmr_Init ([TmrConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the Tmr based on the configuration table defined in tmr_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: MCU clocks initialized

PRE-CONDITION: Timer clock enabled

POST-CONDITION: The TMR peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const TmrConfig_t *TmrConfig = Tmr_ConfigGet();
2
3 Tmr_Init(TmrConfig);
```

See also:

- [Tmr_ConfigGet](#)
- [Tmr_Init](#)
- [Tmr_Enable](#)
- [Tmr_Disable](#)
- [Tmr_RegisterWrite](#)
- [Tmr_RegisterRead](#)
- [Tmr_CallbackRegister](#)

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Tmr_Enable ([TmrRegister_t](#) const *Channel*, [TmrClockMode_t](#) const *Mode*)

Description:

This function enables and starts the specified timer channel.

PRE-CONDITION: Tmer_Init must be called

PRE-CONDITION: MCU clocks initialized

PRE-CONDITION: Timer clock enabled

POST-CONDITION: The timer channel is enabled in the specified mode

Parameters:

in	<i>Channel</i>	- TmrRegister_t, The timer channel to set.
in	<i>Mode</i>	- TmrRegister_t, The timer channel to set.

Returns:

void

Example:

```
1      const TmrConfig_t *TmrConfig = Tmr_ConfigGet();
2
3      Tmr_Init(TmrConfig);
4  Tmr_Enable(TIMER_0, MODE_COUNT_UP);
```

See also:

- [Tmr_ConfigGet](#)
- [Tmr_Init](#)
- [Tmr_Enable](#)
- [Tmr_Disable](#)
- [Tmr_RegisterWrite](#)
- [Tmr_RegisterRead](#)
- [Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Tmr_Disable ([TmrRegister_t](#) const *Channel*)

Description:

This function disables the specified timer channel.

PRE-CONDITION: Tmer_Init must be called

PRE-CONDITION: MCU clocks initialized

PRE-CONDITION: Timer clock enabled

POST-CONDITION: The timer channel is disabled.

Parameters:

in	<i>Channel</i>	- TmrRegister_t, The timer channel to set.
----	----------------	--

Returns:

void

Example:

```
1      const TmrConfig_t *TmrConfig = Tmr_ConfigGet();
2
3      Tmr_Init(TmrConfig);
4  Tmr_Enable(TIMER_0, MODE_COUNT_UP);
5  ...
6  Tmr_Disable(TIMER_0);
```

See also:

- [Tmr_ConfigGet](#)
- [Tmr_Init](#)
- [Tmr_Enable](#)
- [Tmr_Disable](#)
- [Tmr_RegisterWrite](#)
- [Tmr_RegisterRead](#)
- [Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Tmr_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Tmr register. The function should be used to access specialized functionality in the Tmr peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Tmr register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Tmr peripheral map
in	<i>Value</i>	is the value to set the Tmr register to

Returns:

void

Example:

```
1 Tmr_RegisterWrite(0x1000, 0x15);
```

See also:

[Tmr_ConfigGet](#)

[Tmr_Init](#)

[Tmr_Enable](#)

[Tmr_Disable](#)

[Tmr_RegisterWrite](#)

[Tmr_RegisterRead](#)

[Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Tmr_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Tmr register. The function should be used to access specialized functionality in the Tmr peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Tmr register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Tmr register to read
----	----------------	--

Returns:

The current value of the Tmr register.

Example:

```
1 TmrValue = Tmr_RegisterRead(0x1000);
```

See also:

[Tmr_ConfigGet](#)

[Tmr_Init](#)
[Tmr_Enable](#)
[Tmr_Disable](#)
[Tmr_RegisterWrite](#)
[Tmr_RegisterRead](#)
[Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Tmr_CallbackRegister (TmrCallback_t const *Function*, TYPE*)(type) *CallbackFunction*)

Description:

This function is used to set the callback functions of the tmr driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The TmrCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```

1 TmrCallback_t Tmr_Function = TMR_SAMPLE_COMPLETE;
2
3 Tmr_CallbackRegister(Tmr_Function, Tmr_SampleAverage);

```

See also:

[Tmr_ConfigGet](#)
[Tmr_Init](#)
[Tmr_Enable](#)
[Tmr_Disable](#)
[Tmr_RegisterWrite](#)
[Tmr_RegisterRead](#)
[Tmr_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description

09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

tmr_cfg.c File Reference

This module contains the configuration for the tmr module.

```
#include "tmr_cfg.h"
#include "constants.h"
```

Functions

- [TmrConfig_t](#) const *const [Tmr_ConfigGet](#) (void)

Variables

- const [TmrConfig_t](#) [TmrConfig](#) []

Function Documentation

[TmrConfig_t](#) const* const [Tmr_ConfigGet](#) (void)

Description:

This function return a pointer to the Tmr configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const TmrConfig_t * TmrConfig = Tmr_ConfigGet();
2
3 Tmr_Init(TmrConfig);
```

See also:

- [Tmr_ConfigGet](#)
- [Tmr_Init](#)
- [Tmr_Enable](#)
- [Tmr_Disable](#)
- [Tmr_RegisterWrite](#)
- [Tmr_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Variable Documentation

const [TmrConfig_t](#) TmrConfig[]

```
Initial value:=
{
    {TMR0,      ENABLED,      UP_COUNT,      FLL_PLL,      MODULE_CLK,
TMR_DIV_1,    DISABLED,      3,              100,              },
    {TMR1,      DISABLED,      UP_COUNT,      NOT_APPLICABLE,  STOP,
TMR_DIV_1,    DISABLED,      0,              0,                  },
    {TMR2,      ENABLED,      UP_COUNT,      FLL_PLL,      MODULE_CLK,
TMR_DIV_1,    DISABLED,      3,              100,              },
}
```

This configuration table is used to configure the behavior and function of the timers. The channels are defined in [tmr_cfg.h](#). The configuration consists of

- o Timer Name - Specify the name of the timer. This label must be defined in the Tmr_RegisterType enumeration.
- o Timer Enable - Enable or disable the specified timer channel.
- o Timer Mode - Specify the mode of the timer. Timer modes are defined in the Tmr_CounterEnum enumeration.
- o Clock Source - Specify the source of the timer clock. All channels must use the same clock source, so only TMR0 clock source is used.
- o Clock Mode Selection - Specify the clock mode. Clock modes are defined in the Tmr_ClkModeType enumeration.
- o Clock Prescaler - Specify the clock input divider.
- o Interrupt Enable - This sets whether the interrupt for this timer is enabled. DISABLED - Sets the interrupt enable bit low ENABLED - Sets the interrupt enable bit high
- o Timer Interval - Specify the period of the timer in microseconds.

tmr_cfg.h File Reference

This module contains the configuration interface for timer.

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

Data Structures

- struct [TmrConfig_t](#)

Enumerations

- enum [TmrPrescale_t](#) { [TMR_DIV_1](#) = 1, [TMR_DIV_2](#) = 2, [TMR_DIV_4](#) = 4, [TMR_DIV_8](#) = 8, [TMR_DIV_16](#) = 16, [TMR_DIV_32](#) = 32, [TMR_DIV_64](#) = 64, [TMR_DIV_128](#) = 128 }
- enum [TmrCounter_t](#) { [UP_COUNT](#), [UP_DOWN](#) }
- enum [TmrClkSrc_t](#) { [FLL_PLL](#) = 1, [OSCERCLK](#), [INT_CLK](#) }
- enum [TmrRegister_t](#) { [TMR0](#) = 0U, [TMR1](#) = 1U, [TMR2](#) = 2U, [NUM_TIMERS](#) = 3U }
- enum [TmrClockMode_t](#) { [STOP](#), [MODULE_CLK](#), [EXTERNAL_CLK](#) }

Functions

- [TmrConfig_t](#) const *const [Tmr_ConfigGet](#) (void)

Enumeration Type Documentation

enum [TmrPrescale_t](#)

Defines the timer clock input dividers

Enumerator

- TMR_DIV_1*** Timer prescalar of 1:1
- TMR_DIV_2*** Timer prescalar of 1:2
- TMR_DIV_4*** Timer prescalar of 1:4
- TMR_DIV_8*** Timer prescalar of 1:8
- TMR_DIV_16*** Timer prescalar of 1:16
- TMR_DIV_32*** Timer prescalar of 1:32
- TMR_DIV_64*** Timer prescalar of 1:64
- TMR_DIV_128*** Timer prescalar of 1:128

enum [TmrCounter_t](#)

Defines the counter modes for the timer.

Enumerator

- UP_COUNT*** LPTPM counter operates in up counting mode.
- UP_DOWN*** LPTPM counter operates in up-down counting mode.

enum [TmrClkSrc_t](#)

Defines the clock sources for the timer.

Enumerator

- FLL_PLL*** Use the FLL or PLL clock

OSCERCLK Oscillator clock frequency

INT_CLK Internal clock

enum [TmrRegister_t](#)

This enumeration is a list of the timer channels

Enumerator

TMR0 Timer 0

TMR1 Timer 1

TMR2 Timer 2

NUM_TIMERS Number of timers on the microcontroller

enum [TmrClockMode_t](#)

Defines the available clock modes for the timer

Enumerator

STOP LPTPM counter is disabled

MODULE_CLK LPTPM counter increments on every LPTPM counter clock

EXTERNAL_CLK LPTPM counter increments on rising edge of LPTPM_EXTCLK
synchronized to the LPTPM counter clock

Function Documentation

[TmrConfig_t](#) const* const Tmr_ConfigGet (void)

Description:

This function return a pointer to the Tmr configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const TmrConfig_t * TmrConfig = Tmr_ConfigGet();
2
3 Tmr_Init(TmrConfig);
```

See also:

[Tmr_ConfigGet](#)

[Tmr_Init](#)

[Tmr_Enable](#)

[Tmr_Disable](#)

[Tmr_RegisterWrite](#)

[Tmr_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
------	------------------	----------	-------------

09/01/2015	1.0.0	JWB	Interface Created
------------	-------	-----	-------------------

uart.c File Reference

The implementation for the uart.

```
#include "uart.h"
```

```
#include "constants.h"
```

Functions

- void [Uart_Init](#) ([UartConfig_t](#) const *const Config)
- void [Uart_ParitySet](#) ([UartChannel_t](#) const Channel, [UartConfig_t](#) const *const Config)
- void [Uart_IsrModeSet](#) ([UartChannel_t](#) const Channel, [UartConfig_t](#) const *const Config)
- void [Uart_BaudRateSet](#) ([UartChannel_t](#) const Channel, [UartConfig_t](#) const *const Config)
- uint8_t [Uart_CharGet](#) ([UartChannel_t](#) const Channel)
- uint8_t [Uart_IsDataPresent](#) ([UartChannel_t](#) const Channel)
- void [Uart_CharPut](#) ([UartChannel_t](#) const Channel, char const Ch)
- void [Uart_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Uart_RegisterRead](#) (uint32_t const Address)
- void [Uart_CallbackRegister](#) (UartCallback_t const Function, TYPE(*CallbackFunction)(type))

Variables

- const unsigned char [CharacterArray](#) [] ={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'}
- const [UartBaud_t](#) [UartBaudTable](#) []

Function Documentation

void [Uart_Init](#) ([UartConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the Uart based on the configuration table defined in `uart_cfg` module.

PRE-CONDITION: Configuration table needs to be populated (`sizeof > 0`)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The uart peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const UartConfig_t *UartConfig = Uart_ConfigGet();
2
3 Uart_Init(UartConfig);
```

See also:

[Uart_Init](#)

[Uart_ParitySet](#)

[Uart_IsrModeSet](#)

[Uart_BaudRateSet](#)

[Uart_CharGet](#)

[Uart_IsDataPresent](#)

[Uart_CharPut](#)

[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_ParitySet ([UartChannel_t](#) const *Channel*, [UartConfig_t](#) const *const *Config*)

Description:

This function is used to set the Uart transmission parity.

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The uart channel is configured with the provided parity.

Parameters:

in	<i>Config</i>	- Pointer to UartConfig_t .
in	<i>Channel</i>	- uint8_t, Uart channel number.

Returns:

void

Example:

```
1 const UartConfig_t *UartConfig = Uart_GetConfig();  
2  
3 Uart_Init(UartConfig);
```

See also:

[Uart_Init](#)
[Uart_ParitySet](#)
[Uart_IsrModeSet](#)
[Uart_BaudRateSet](#)
[Uart_CharGet](#)
[Uart_IsDataPresent](#)
[Uart_CharPut](#)
[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_IsrModeSet ([UartChannel_t](#) const *Channel*, [UartConfig_t](#) const *const *Config*)

Description:

This function is used to set the ISR mode.

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The uart channel interrupt is configured

Parameters:

in	<i>Channel</i>	- uint8_t, Uart channel number.
in	<i>Config</i>	- Pointer to UartConfig_t .

Returns:

void

Example:

```
1 const UartConfig_t *UartConfig = Uart_GetConfig();
2
3 Uart_Init(UartConfig);
```

See also:

- [Uart_Init](#)
- [Uart_ParitySet](#)
- [Uart_IsrModeSet](#)
- [Uart_BaudRateSet](#)
- [Uart_CharGet](#)
- [Uart_IsDataPresent](#)
- [Uart_CharPut](#)
- [Uart_RegisterWrite](#)
- [Uart_RegisterWrite](#)
- [Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_BaudRateSet ([UartChannel_t](#) const *Channel*, [UartConfig_t](#) const *const *Config*)

Description:

This function is used to set channel baud rate

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The uart channel data rate is set.

Parameters:

in	<i>Channel</i>	- uint8_t, Uart channel number.
in	<i>Config</i>	- Pointer to UartConfig_t .

Returns:

void

Example:

```
1 Uart_Init(UartConfig_t UartConfig)
2 {
3     Uart_SetBaud(UartConfig);
4 }
```

See also:

[Uart_Init](#)

[Uart_ParitySet](#)

[Uart_IsrModeSet](#)

[Uart_BaudRateSet](#)

[Uart_CharGet](#)

[Uart_IsDataPresent](#)

[Uart_CharPut](#)

[Uart_RegisterWrite](#)

[Uart_RegisterWrite](#)

[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint8_t Uart_CharGet ([UartChannel_t](#) const *Channel*)

Description:

This function is used to get a character from the uart.

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The uart channel data is received.

Parameters:

in	<i>Channel</i>	- uint8_t, Uart channel number.
----	----------------	---------------------------------

Returns:

uint8_t - the data byte

Example:

```
1 uint8_t uartByte = Uart_GetChar(UART_0);
```

See also:

[Uart_Init](#)
[Uart_ParitySet](#)
[Uart_IsrModeSet](#)
[Uart_BaudRateSet](#)
[Uart_CharGet](#)
[Uart_IsDataPresent](#)
[Uart_CharPut](#)
[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint8_t Uart_IsDataPresent ([UartChannel_t](#) const *Channel*)

Description:

This routine checks to see if there is a new byte in UART reception buffer. Uart_Init must be called prior to calling this routine.

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: State of the receive buffer is updated.

Parameters:

in	<i>Channel</i>	- uint8_t, Uart channel number.
----	----------------	---------------------------------

Returns:

0, No new data received. 1, Data is in the receive buffer.

Example:

```
1 uint8_t DataPresent = Uart_IsDataPresent(UART_0);
```

See also:

[Uart_Init](#)
[Uart_ParitySet](#)
[Uart_IsrModeSet](#)
[Uart_BaudRateSet](#)
[Uart_CharGet](#)
[Uart_IsDataPresent](#)

[Uart_CharPut](#)
[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_CharPut ([UartChannel_t](#) const *Channel*, char const *Ch*)

Description:

This routine writes a character to the transmit FIFO, and then waits for the transmit FIFO to be empty. Uart_Init must be called prior to calling this routine.

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Data put out on the uart channel

Parameters:

in	<i>Channel</i>	- UartChannelType, Uart channel to use.
in	<i>Ch</i>	- Byte to be sent.

Returns:

None

Example:

```
1      char test = 'c';  
2  
3  Uart_PutChar(test, UART_0);
```

See also:

[Uart_Init](#)
[Uart_ParitySet](#)
[Uart_IsrModeSet](#)
[Uart_BaudRateSet](#)
[Uart_CharGet](#)
[Uart_IsDataPresent](#)
[Uart_CharPut](#)
[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description

09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Uart register. The function should be used to access specialized functionality in the Uart peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Uart register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Uart peripheral map
in	<i>Value</i>	is the value to set the Uart register to

Returns:

void

Example:

```
1 Uart_RegisterWrite(0x1000, 0x15);
```

See also:

- [Uart_Init](#)
- [Uart_ParitySet](#)
- [Uart_IsrModeSet](#)
- [Uart_BaudRateSet](#)
- [Uart_CharGet](#)
- [Uart_IsDataPresent](#)
- [Uart_CharPut](#)
- [Uart_RegisterWrite](#)
- [Uart_RegisterWrite](#)
- [Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Uart_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Uart register. The function should be used to access specialized functionality in the Uart peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Uart register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Uart register to read
----	----------------	---

Returns:

The current value of the Uart register.

Example:

```
1 UartValue = Uart_RegisterRead(0x1000);
```

See also:

- [Uart_Init](#)
- [Uart_ParitySet](#)
- [Uart_IsrModeSet](#)
- [Uart_BaudRateSet](#)
- [Uart_CharGet](#)
- [Uart_IsDataPresent](#)
- [Uart_CharPut](#)
- [Uart_RegisterWrite](#)
- [Uart_RegisterWrite](#)
- [Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_CallbackRegister (UartCallback_t const *Function*, TYPE(*)*(type)* CallbackFunction)

Description:

This function is used to set the callback functions of the Uart driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The UartCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

--	--	--

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 UartCallback_t Uart_Function = Uart_SAMPLE_COMPLETE;
2
3 Uart_CallbackRegister(Uart_Function, Uart_SampleAverage);
```

See also:

- [Uart_Init](#)
- [Uart_ParitySet](#)
- [Uart_IsrModeSet](#)
- [Uart_BaudRateSet](#)
- [Uart_CharGet](#)
- [Uart_IsDataPresent](#)
- [Uart_CharPut](#)
- [Uart_RegisterWrite](#)
- [Uart_RegisterWrite](#)
- [Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

Variable Documentation

const unsigned char CharacterArray[] ={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'}

Character array used to print hex values

const [UartBaud_t](#) UartBaudTable[]

This table defines the uart register values for common system clock frequencies and UART baud rates.

uart.h File Reference

The interface definition for the uart.

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

Data Structures

- struct [UartBaud_t](#)

Macros

- #define [NUM_BAUD_TABLE](#) (sizeof([UartBaudTable](#))/sizeof([UartBaud_t](#)))

Functions

- void [Uart_Init](#) ([UartConfig_t](#) const *const Config)
- void [Uart_BaudRateSet](#) ([UartChannel_t](#) const Channel, [UartConfig_t](#) const *const Config)
- uint8_t [Uart_CharGet](#) ([UartChannel_t](#) const Channel)
- void [Uart_CharPut](#) ([UartChannel_t](#) const Channel, char const Ch)
- uint8_t [Uart_IsDataPresent](#) ([UartChannel_t](#) const Channel)
- void [Uart_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Uart_RegisterRead](#) (uint32_t const Address)
- void [Uart_CallbackRegister](#) (UartCallback_t const Function, TYPE(*CallbackFunction)(type))

Macro Definition Documentation

```
#define NUM_BAUD_TABLE (sizeof(UartBaudTable)/sizeof(UartBaud\_t))
```

Defines the number of entries in the UartBaudTable array

Function Documentation

```
void Uart_Init (UartConfig\_t const *const Config)
```

Description:

This function is used to initialize the Uart based on the configuration table defined in `uart_cfg` module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The uart peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const UartConfig_t *UartConfig = Uart_ConfigGet();
2
3 Uart_Init(UartConfig);
```

See also:

[Uart_Init](#)
[Uart_ParitySet](#)
[Uart_IsrModeSet](#)
[Uart_BaudRateSet](#)
[Uart_CharGet](#)
[Uart_IsDataPresent](#)
[Uart_CharPut](#)
[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_BaudRateSet ([UartChannel_t](#) const *Channel*, [UartConfig_t](#) const *const *Config*)

Description:

This function is used to set channel baud rate

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The uart channel data rate is set.

Parameters:

in	<i>Channel</i>	- uint8_t, Uart channel number.
in	<i>Config</i>	- Pointer to UartConfig_t .

Returns:

void

Example:

```

1 Uart_Init(UartConfig_t UartConfig)
2 {
3     Uart_SetBaud(UartConfig);
4 }
```

See also:

[Uart_Init](#)
[Uart_ParitySet](#)
[Uart_IsrModeSet](#)
[Uart_BaudRateSet](#)
[Uart_CharGet](#)
[Uart_IsDataPresent](#)
[Uart_CharPut](#)
[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint8_t Uart_CharGet ([UartChannel_t](#) const *Channel*)

Description:

This function is used to get a character from the uart.

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: The uart channel data is received.

Parameters:

in	<i>Channel</i>	- uint8_t, Uart channel number.
----	----------------	---------------------------------

Returns:

uint8_t - the data byte

Example:

```
1 uint8_t uartByte = Uart_GetChar(UART_0);
```

See also:

- [Uart_Init](#)
- [Uart_ParitySet](#)
- [Uart_IsrModeSet](#)
- [Uart_BaudRateSet](#)
- [Uart_CharGet](#)
- [Uart_IsDataPresent](#)
- [Uart_CharPut](#)
- [Uart_RegisterWrite](#)
- [Uart_RegisterWrite](#)
- [Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_CharPut ([UartChannel_t](#) const *Channel*, char const *Ch*)

Description:

This routine writes a character to the transmit FIFO, and then waits for the transmit FIFO to be empty. Uart_Init must be called prior to calling this routine.

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: Data put out on the uart channel

Parameters:

in	<i>Channel</i>	- UartChannelType, Uart channel to use.
in	<i>Ch</i>	- Byte to be sent.

Returns:

None

Example:

```
1      char test = 'c';
2
3  Uart_PutChar(test, UART_0);
```

See also:

- [Uart_Init](#)
- [Uart_ParitySet](#)
- [Uart_IsrModeSet](#)
- [Uart_BaudRateSet](#)
- [Uart_CharGet](#)
- [Uart_IsDataPresent](#)
- [Uart_CharPut](#)
- [Uart_RegisterWrite](#)
- [Uart_RegisterWrite](#)
- [Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint8_t Uart_IsDataPresent ([UartChannel_t](#) const *Channel*)

Description:

This routine checks to see if there is a new byte in UART reception buffer. Uart_Init must be called prior to calling this routine.

PRE-CONDITION: The Uart_Init function is called with valid configuration.

PRE-CONDITION: The MCU clocks must be configured and enabled.

POST-CONDITION: State of the receive buffer is updated.

Parameters:

in	<i>Channel</i>	- uint8_t, Uart channel number.
----	----------------	---------------------------------

Returns:

0, No new data received. 1, Data is in the receive buffer.

Example:

```
1 uint8_t DataPresent = Uart_IsDataPresent(UART_0);
```

See also:

- [Uart_Init](#)
- [Uart_ParitySet](#)
- [Uart_IsrModeSet](#)
- [Uart_BaudRateSet](#)
- [Uart_CharGet](#)
- [Uart_IsDataPresent](#)
- [Uart_CharPut](#)
- [Uart_RegisterWrite](#)
- [Uart_RegisterWrite](#)
- [Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Uart register. The function should be used to access specialized functionality in the Uart peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Uart register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Uart peripheral map
in	<i>Value</i>	is the value to set the Uart register to

Returns:

void

Example:

```
1 Uart_RegisterWrite(0x1000, 0x15);
```

See also:

[Uart_Init](#)
[Uart_ParitySet](#)
[Uart_IsrModeSet](#)
[Uart_BaudRateSet](#)
[Uart_CharGet](#)
[Uart_IsDataPresent](#)
[Uart_CharPut](#)
[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Uart_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a Uart register. The function should be used to access specialized functionality in the Uart peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Uart register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the Uart register to read
----	----------------	---

Returns:

The current value of the Uart register.

Example:

```
1 UartValue = Uart_RegisterRead(0x1000);
```

See also:

[Uart_Init](#)
[Uart_ParitySet](#)
[Uart_IsrModeSet](#)
[Uart_BaudRateSet](#)
[Uart_CharGet](#)
[Uart_IsDataPresent](#)
[Uart_CharPut](#)
[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Uart_CallbackRegister (UartCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the Uart driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The UartCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 UartCallback_t Uart_Function = Uart_SAMPLE_COMPLETE;  
2  
3 Uart_CallbackRegister(Uart_Function, Uart_SampleAverage);
```

See also:

[Uart_Init](#)
[Uart_ParitySet](#)
[Uart_IsrModeSet](#)
[Uart_BaudRateSet](#)
[Uart_CharGet](#)
[Uart_IsDataPresent](#)
[Uart_CharPut](#)
[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uart_cfg.c File Reference

This module contains the uart configuration code.

```
#include "uart_cfg.h"
#include "constants.h"
```

Functions

- [UartConfig_t](#) const *const [Uart_ConfigGet](#) (void)

Variables

- const [UartConfig_t](#) [UartConfig](#) []

Function Documentation

[UartConfig_t](#) const* const [Uart_ConfigGet](#) (void)

Description:

This function is used to initialize the uart based on the configuration table defined in this module.
PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const UartConfig_t *UartConfig = Uart_ConfigGet();
2
3 Uart_Init(UartConfig);
```

See also:

- [Uart_Init](#)
- [Uart_ParitySet](#)
- [Uart_IsrModeSet](#)
- [Uart_BaudRateSet](#)
- [Uart_CharGet](#)
- [Uart_IsDataPresent](#)
- [Uart_CharPut](#)
- [Uart_RegisterWrite](#)
- [Uart_RegisterWrite](#)
- [Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Variable Documentation

const [UartConfig_t](#) UartConfig[]

```
Initial value:=
{
    {UART\_0, ENABLED, UART, SUBMCLK, 115200, DISABLED, UART\_LSB\_FIRST,
BITS\_EIGHT, 1, DISABLED, 1, DISABLED },
    {UART\_1, DISABLED, UART, SUBMCLK, 115200, DISABLED, UART\_LSB\_FIRST,
BITS\_EIGHT, 1, DISABLED, 1, DISABLED },
    {UART\_2, DISABLED, UART, SUBMCLK, 115200, DISABLED, UART\_LSB\_FIRST,
BITS\_EIGHT, 1, DISABLED, 1, DISABLED },
}
```

This configuration table is used to configure the behavior and function of the timers. The channels are defined in [tmr_cfg.h](#). The configuration consists of

- o Uart Name - Specify the name of the uart. This label must be defined in the Uart_RegisterType enumeration.
- o Uart Enable - Specify whether the uart channel is enabled.
- o Uart Mode - Specify the mode of the UCSI channel. UART - Standard UART mode IDLE_LINE - Idle-line multiprocessor mode ADDR_BIT - Address-bit multiprocessor mode UART_AUTO - UART mode with automatic baud rate detection
- o Clock Source - Choose the UART clock source.
- o Baud Rate - Specify the desired baud rate for the uart channel.
- o Loopback - Enable or disable loopback mode
- o Bit Direction - Specify the bit ordering for uart receive and transmit shift registers. LEAST_FIRST - Least significant bit first MOST_FIRST - Most significant bit first
- o Data Length - Specify the character length, 7- or 8-bits.
- o Data Length - Specify the number of stop bits, 1 or 2.
- o Parity Type - Specify whether the uart parity is ODD, EVEN, or DISABLED.
- o AutoBaud Enable - Enable or Disable automatic baud rate detection.
- o Delimiter - Specify the break/synch delimiter length for auto baud detection.
- o Interrupt Enable - This sets whether the receive interrupt for this uart is enabled. DISABLED - Disable the UART interrupts RX_ONLY - Enable receive interrupt, transmit interrupt disabled TX_ONLY - Enable transmit interrupt, receive interrupt disabled RX_TX - Enable both receive and transmit interrupts

UART_0 and UART_1 channels are shared with the SPIA_0 and SPIA_1 channel. Only one of them can be enabled on a channel at a time.

uart_cfg.h File Reference

This file contains the header definitions for the uart configuration.

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

Data Structures

- struct [UartConfig_t](#)

Enumerations

- enum [UartClkSrc_t](#) { [UCLK](#), [ACLOCK](#), [SUBMCLK](#) }
- enum [UartParity_t](#) { [ODD](#) = 0x01, [EVEN](#) = 0x02 }
- enum [UartBitOrder_t](#) { [UART_LSB_FIRST](#), [UART_MSB_FIRST](#) }
- enum [UartComm_t](#) { [BITS_EIGHT](#), [BITS_NINE](#) }
- enum [UartMode_t](#) { [UART](#) }
- enum [UartInt_t](#) { [RX_ONLY](#) = 1U, [TX_ONLY](#) = 2U, [RX_TX](#) = 3U }
- enum [UartChannel_t](#) { [UART_0](#), [UART_1](#), [UART_2](#), [NUM_UART_CHANNELS](#) }

Functions

- [UartConfig_t](#) const *const [Uart_ConfigGet](#) (void)

Enumeration Type Documentation

enum [UartClkSrc_t](#)

Defines the available BRCLK source selections.

Enumerator

- UCLK*** U Clock
- ACLOCK*** Auxilary Clock
- SUBMCLK*** Sub-System Master Clock

enum [UartParity_t](#)

Defines the uart parity selections.

Enumerator

- ODD*** Odd Parity
- EVEN*** Even Parity

enum [UartBitOrder_t](#)

Defines the bit ordering for uart receive and transmit shift registers.

Enumerator

- UART_LSB_FIRST*** Least significant bit sent first
- UART_MSB_FIRST*** Most significant bit sent first

enum [UartComm_t](#)

Defines the size of data sent to and from peripheral

Enumerator

- BITS_EIGHT*** 8 bits sent at a time

BITS_NINE 9 bits sent at a time

enum [UartMode_t](#)

Defines the possible USCI modes

Enumerator

UART Standard UART mode

enum [UartInt_t](#)

Defines the possible interrupt modes

Enumerator

RX_ONLY Enable Receive Interrupt

TX_ONLY Enable Transmit Interrupt

RX_TX Enable both Rx and Tx Interrupts

enum [UartChannel_t](#)

This enumeration is a list of the uart channels

Enumerator

UART_0 UART0

UART_1 UART1

UART_2 UART2

NUM_UART_CHANNELS Number of UART channels

Function Documentation

[UartConfig_t](#) const* const Uart_ConfigGet (void)

Description:

This function is used to initialize the uart based on the configuration table defined in this module.

PRE-CONDITION: Configuration table needs to be populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const UartConfig_t *UartConfig = Uart_ConfigGet();
2
3 Uart_Init(UartConfig);
```

See also:

[Uart_Init](#)

[Uart_ParitySet](#)

[Uart_IsrModeSet](#)

[Uart_BaudRateSet](#)

[Uart_CharGet](#)

[Uart_IsDataPresent](#)

[Uart_CharPut](#)

[Uart_RegisterWrite](#)
[Uart_RegisterWrite](#)
[Uart_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

wdt.c File Reference

The implementation for the watchdog.

```
#include "wdt.h"
#include "constants.h"
```

Macros

- `#define WDT_SOFTWARE_RESET 0xFF`

Functions

- void [Wdt_Init](#) ([WdtConfig_t](#) const *const Config)
- void [Wdt_Enable](#) (void)
- void [Wdt_Disable](#) (void)
- void [Wdt_Clear](#) (void)
- void [Wdt_Reset](#) (void)
- void [Wdt_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Wdt_RegisterRead](#) (uint32_t const Address)
- void [Wdt_CallbackRegister](#) (WdtCallback_t const Function, TYPE(*CallbackFunction)(type))

Variables

- TYPE volatile *const [wdtcon](#) = (TYPE*)®ISTER
- TYPE volatile *const [wdtsvr](#) = (TYPE*)®ISTER

Macro Definition Documentation

`#define WDT_SOFTWARE_RESET 0xFF`

Used to force a soft reset of the processor through the watchdog.

Function Documentation

void Wdt_Init ([WdtConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the watchdog based on the configuration table defined in `wdt_cfg` module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: System Clock Initialized

POST-CONDITION: The WDT peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3 Wdt_Init(WdtConfig);
```

See also:

[Wdt_ConfigGet](#)
[Wdt_Init](#)
[Wdt_Enable](#)
[Wdt_Disable](#)
[Wdt_Clear](#)
[Wdt_Reset](#)
[Wdt_RegisterWrite](#)
[Wdt_RegisterRead](#)
[Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_Enable (void)

Description:

This function is used to enable the watchdog. In most MCU's once enabled the watchdog cannot be disabled! (Thankfully).

PRE-CONDITION: Wdt_Init must be called with valid configuration data.

POST-CONDITION: The WDT peripheral is enabled and now must be cleared to prevent a reset.

Returns:

void

Example:

```
1    const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3    Wdt_Init(WdtConfig);
4    Wdt_Enable();
```

See also:

[Wdt_ConfigGet](#)
[Wdt_Init](#)
[Wdt_Enable](#)
[Wdt_Disable](#)
[Wdt_Clear](#)
[Wdt_Reset](#)
[Wdt_RegisterWrite](#)
[Wdt_RegisterRead](#)
[Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
------	------------------	----------	-------------

09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_Disable (void)

Description:

This function is used to disable the watchdog (not recommended or possible on many microcontrollers once it is enabled)

PRE-CONDITION: Wdt_Init must be called with valid configuration data

PRE-CONDITION: Wdt_Enable must have been called to enable the watchdog

POST-CONDITION: The WDT peripheral is disabled (if possible)

Returns:

void

Example:

```
1      const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3      Wdt_Init(WdtConfig);
4 Wdt_Enable();
5 Wdt_Disable();
```

See also:

- [Wdt_ConfigGet](#)
- [Wdt_Init](#)
- [Wdt_Enable](#)
- [Wdt_Disable](#)
- [Wdt_Clear](#)
- [Wdt_Reset](#)
- [Wdt_RegisterWrite](#)
- [Wdt_RegisterRead](#)
- [Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_Clear (void)

Description:

This function is used to clear the watchdog register to prevent a watchdog reset of the system.

PRE-CONDITION: Wdt_Init must be called with valid configuration data

PRE-CONDITION: Wdt_Enable must have been called to enable the watchdog.

POST-CONDITION: The WDT peripheral counter register is cleared.

Returns:

void

Example:

```
1      const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3      Wdt_Init(WdtConfig);
4  Wdt_Enable();
5
6  // application code
7  // System health checks
8  if(SystemHealthy == TRUE)
9  {
10         Wdt_Clear();
11 }
```

See also:

[Wdt_ConfigGet](#)

[Wdt_Init](#)

[Wdt_Enable](#)

[Wdt_Disable](#)

[Wdt_Clear](#)

[Wdt_Reset](#)

[Wdt_RegisterWrite](#)

[Wdt_RegisterRead](#)

[Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_Reset (void)

Description:

This function is used to trigger a watchdog reset of the processor. Function may have no effect if the Wdt register has already been written to.

PRE-CONDITION: Wdt_Init must be called with valid configuration data

PRE-CONDITION: Wdt_Enable must have been called to enable the watchdog.

POST-CONDITION: The WDT forces a reset of the MCU.

Returns:

void

Example:

```
1      const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3      Wdt_Init(WdtConfig);
4  Wdt_Enable();
5
6  // application code
7  // System health checks
8  if(SystemHealthy == TRUE)
9  {
10         Wdt_Clear();
11 }
12 else
13 {
14         Wdt_Reset();
15 }
```

See also:

- [Wdt_ConfigGet](#)
- [Wdt_Init](#)
- [Wdt_Enable](#)
- [Wdt_Disable](#)
- [Wdt_Clear](#)
- [Wdt_Reset](#)
- [Wdt_RegisterWrite](#)
- [Wdt_RegisterRead](#)
- [Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Dio register. The function should be used to access specialized functionality in the Dio peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Dio register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Dio peripheral map
in	<i>Value</i>	is the value to set the Dio register to

Returns:

void

Example:

```
1 Wdt_RegisterWrite(0x1000, 0x15);
```

See also:

[Wdt_ConfigGet](#)

[Wdt_Init](#)

[Wdt_Enable](#)

[Wdt_Disable](#)

[Wdt_Clear](#)

[Wdt_Reset](#)

[Wdt_RegisterWrite](#)

[Wdt_RegisterRead](#)

[Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Wdt_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a wdt register. The function should be used to access specialized functionality in the wdt peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the wdt register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the wdt register to read
----	----------------	--

Returns:

The current value of the wdt register.

Example:

```
1 WdtValue = Wdt_RegisterRead(0x1000);
```

See also:

[Wdt_ConfigGet](#)

[Wdt_Init](#)

[Wdt_Enable](#)

[Wdt_Disable](#)

[Wdt_Clear](#)

[Wdt_Reset](#)

[Wdt_RegisterWrite](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_CallbackRegister (WdtCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the Wdt driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The WdtCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 WdtCallback_t Wdt_Function = Wdt_SAMPLE_COMPLETE;  
2  
3 Wdt_CallbackRegister(Wdt_Function, Wdt_SampleAverage);
```

See also:

[Wdt_ConfigGet](#)
[Wdt_Init](#)
[Wdt_Enable](#)
[Wdt_Disable](#)
[Wdt_Clear](#)
[Wdt_Reset](#)
[Wdt_RegisterWrite](#)
[Wdt_RegisterRead](#)
[Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

Variable Documentation

TYPE volatile* const wdtcon = (TYPE*)®ISTER

Defines a pointer to the wdt control register on the microcontroller.

TYPE volatile* const wdtsvr = (TYPE*)®ISTER

Defines a pointer to the wdt service register on the microcontroller.

wdt.h File Reference

The interface definition for the watchdog.

```
#include "wdt_cfg.h"
```

Functions

- void [Wdt_Init](#) ([WdtConfig_t](#) const *const Config)
- void [Wdt_Enable](#) (void)
- void [Wdt_Disable](#) (void)
- void [Wdt_Reset](#) (void)
- void [Wdt_Clear](#) (void)
- void [Wdt_RegisterWrite](#) (uint32_t const Address, uint32_t const Value)
- uint32_t [Wdt_RegisterRead](#) (uint32_t const Address)
- void [Wdt_CallbackRegister](#) (WdtCallback_t const Function, TYPE(*CallbackFunction)(type))

Detailed Description

This is the header file for the definition of the interface for the watchdog timer.

Function Documentation

void Wdt_Init ([WdtConfig_t](#) const *const *Config*)

Description:

This function is used to initialize the watchdog based on the configuration table defined in wdt_cfg module.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

PRE-CONDITION: System Clock Initialized

POST-CONDITION: The WDT peripheral is setup with the configuration settings.

Parameters:

in	<i>Config</i>	is a pointer to the configuration table that contains the initialization for the peripheral.
----	---------------	--

Returns:

void

Example:

```
1 const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3 Wdt_Init(WdtConfig);
```

See also:

- [Wdt_ConfigGet](#)
- [Wdt_Init](#)
- [Wdt_Enable](#)
- [Wdt_Disable](#)
- [Wdt_Clear](#)
- [Wdt_Reset](#)
- [Wdt_RegisterWrite](#)
- [Wdt_RegisterRead](#)
- [Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_Enable (void)

Description:

This function is used to enable the watchdog. In most MCU's once enabled the watchdog cannot be disabled! (Thankfully).

PRE-CONDITION: Wdt_Init must be called with valid configuration data.

POST-CONDITION: The WDT peripheral is enabled and now must be cleared to prevent a reset.

Returns:

void

Example:

```
1      const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3      Wdt_Init(WdtConfig);
4  Wdt_Enable();
```

See also:

- [Wdt_ConfigGet](#)
- [Wdt_Init](#)
- [Wdt_Enable](#)
- [Wdt_Disable](#)
- [Wdt_Clear](#)
- [Wdt_Reset](#)
- [Wdt_RegisterWrite](#)
- [Wdt_RegisterRead](#)
- [Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_Disable (void)

Description:

This function is used to disable the watchdog (not recommended or possible on many microcontrollers once it is enabled)

PRE-CONDITION: Wdt_Init must be called with valid configuration data

PRE-CONDITION: Wdt_Enable must have been called to enable the watchdog

POST-CONDITION: The WDT peripheral is disabled (if possible)

Returns:

void

Example:

```
1      const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3      Wdt_Init(WdtConfig);
4  Wdt_Enable();
5  Wdt_Disable();
```

See also:

- [Wdt_ConfigGet](#)
- [Wdt_Init](#)
- [Wdt_Enable](#)
- [Wdt_Disable](#)
- [Wdt_Clear](#)
- [Wdt_Reset](#)
- [Wdt_RegisterWrite](#)
- [Wdt_RegisterRead](#)
- [Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_Reset (void)

Description:

This function is used to trigger a watchdog reset of the processor. Function may have no effect if the Wdt register has already been written to.

PRE-CONDITION: Wdt_Init must be called with valid configuration data

PRE-CONDITION: Wdt_Enable must have been called to enable the watchdog.

POST-CONDITION: The WDT forces a reset of the MCU.

Returns:

void

Example:

```
1      const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3      Wdt_Init(WdtConfig);
4  Wdt_Enable();
5
6  // application code
7  // System health checks
8  if(SystemHealthy == TRUE)
9  {
10     Wdt_Clear();
11 }
12 else
13 {
14     Wdt_Reset();
15 }
```

See also:

- [Wdt_ConfigGet](#)
- [Wdt_Init](#)
- [Wdt_Enable](#)
- [Wdt_Disable](#)
- [Wdt_Clear](#)
- [Wdt_Reset](#)
- [Wdt_RegisterWrite](#)
- [Wdt_RegisterRead](#)
- [Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_Clear (void)

Description:

This function is used to clear the watchdog register to prevent a watchdog reset of the system.

PRE-CONDITION: Wdt_Init must be called with valid configuration data

PRE-CONDITION: Wdt_Enable must have been called to enable the watchdog.

POST-CONDITION: The WDT peripheral counter register is cleared.

Returns:

void

Example:

```
1      const WdtConfig_t *WdtConfig = Wdt_ConfigGet();
2
3      Wdt_Init(WdtConfig);
4  Wdt_Enable();
5
6  // application code
```

```

7 // System health checks
8 if(SystemHealthy == TRUE)
9 {
10     Wdt_Clear();
11 }

```

See also:

[Wdt_ConfigGet](#)
[Wdt_Init](#)
[Wdt_Enable](#)
[Wdt_Disable](#)
[Wdt_Clear](#)
[Wdt_Reset](#)
[Wdt_RegisterWrite](#)
[Wdt_RegisterRead](#)
[Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_RegisterWrite (uint32_t const *Address*, uint32_t const *Value*)

Description:

This function is used to directly address and modify a Dio register. The function should be used to access specialized functionality in the Dio peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the Dio register address space

POST-CONDITION: The register located at Address with be updated with Value

Parameters:

in	<i>Address</i>	is a register address within the Dio peripheral map
in	<i>Value</i>	is the value to set the Dio register to

Returns:

void

Example:

```

1 Wdt_RegisterWrite(0x1000, 0x15);

```

See also:

[Wdt_ConfigGet](#)
[Wdt_Init](#)
[Wdt_Enable](#)
[Wdt_Disable](#)
[Wdt_Clear](#)

[Wdt_Reset](#)
[Wdt_RegisterWrite](#)
[Wdt_RegisterRead](#)
[Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

uint32_t Wdt_RegisterRead (uint32_t const *Address*)

Description:

This function is used to directly address a wdt register. The function should be used to access specialized functionality in the wdt peripheral that is not exposed by any other function of the interface.

PRE-CONDITION: Address is within the boundaries of the wdt register address space

POST-CONDITION: The value stored in the register is returned to the caller

Parameters:

in	<i>Address</i>	is the address of the wdt register to read
----	----------------	--

Returns:

The current value of the wdt register.

Example:

```
1 WdtValue = Wdt_RegisterRead(0x1000);
```

See also:

[Wdt_ConfigGet](#)
[Wdt_Init](#)
[Wdt_Enable](#)
[Wdt_Disable](#)
[Wdt_Clear](#)
[Wdt_Reset](#)
[Wdt_RegisterWrite](#)
[Wdt_RegisterRead](#)
[Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

void Wdt_CallbackRegister (WdtCallback_t const *Function*, TYPE(*)(*type*) *CallbackFunction*)

Description:

This function is used to set the callback functions of the Wdt driver. By default, the callbacks are initialized to a NULL pointer. The driver may contain more than one possible callback, so the function will take a parameter to configure the specified callback.

PRE-CONDITION: The WdtCallback_t has been populated

PRE-CONDITION: The callback function exists within memory.

POST-CONDITION: The specified callback function will be registered with the driver.

Parameters:

in	<i>Function</i>	is the callback function that will be registered
in	<i>CallbackFunction</i>	is a function pointer to the desired function

Returns:

None.

Example:

```
1 WdtCallback_t Wdt_Function = Wdt_SAMPLE_COMPLETE;  
2  
3 Wdt_CallbackRegister(Wdt_Function, Wdt_SampleAverage);
```

See also:

- [Wdt_ConfigGet](#)
- [Wdt_Init](#)
- [Wdt_Enable](#)
- [Wdt_Disable](#)
- [Wdt_Clear](#)
- [Wdt_Reset](#)
- [Wdt_RegisterWrite](#)
- [Wdt_RegisterRead](#)
- [Wdt_CallbackRegister](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	0.5.0	JWB	Interface Created
11/10/2015	1.0.0	JWB	Interface Released

wdt_cfg.c File Reference

This module contains the configuration for the wdt module.

```
#include "wdt_cfg.h"
#include "constants.h"
```

Functions

- [WdtConfig_t](#) const *const [Wdt_ConfigGet](#) (void)

Variables

- const [WdtConfig_t](#) [WdtConfig](#)

Function Documentation

[WdtConfig_t](#) const* const [Wdt_ConfigGet](#) (void)

Description:

This function return a pointer to the Wdt configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const WdtConfig_t * WdtConfig = Wdt_ConfigGet();
2
3 Wdt_Init(WdtConfig);
```

See also:

- [Wdt_ConfigGet](#)
- [Wdt_Init](#)
- [Wdt_Enable](#)
- [Wdt_Disable](#)
- [Wdt_Clear](#)
- [Wdt_Reset](#)
- [Wdt_RegisterWrite](#)
- [Wdt_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Variable Documentation

const [WdtConfig_t](#) WdtConfig

```
Initial value:=  
{  
    NORMAL,  
    ONE\_KHZ,  
    INT\_256,  
}
```

The Wdt configuration settings to initialize the wdt register.

wdt_cfg.h File Reference

This module contains the configuration interface for wdt.

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

Data Structures

- struct [WdtConfig_t](#)

Enumerations

- enum [WdtMode_t](#) { **NORMAL** = 1, **WINDOWED** = 2, [NORMAL](#) = 1, [WINDOWED](#) = 2 }
- enum [WdtClkSrc_t](#) { **ONE_KHZ**, **BUS**, [ONE_KHZ](#), [BUS](#) }
- enum [WdtInterval_t](#) { **NONE**, **INT_32**, **INT_256**, **INT_1024**, **INT_8192** = 0x01, **INT_65536**, **INT_262144**, [NONE](#), [INT_32](#), [INT_256](#), [INT_1024](#), [INT_8192](#) = 0x01, [INT_65536](#), [INT_262144](#) }

Functions

- [WdtConfig_t](#) const *const [Wdt_ConfigGet](#) (void)

Enumeration Type Documentation

enum WdtMode_t

Defines the available modes of the Watchdog Timer

Enumerator

NORMAL Normal watchdog operation

WINDOWED Window is opened three quarters through the timeout period. Only supported when the Wdt is running from the Bus clock

enum WdtClkSrc_t

Defines the available clock sources for the Watchdog Timer

Enumerator

ONE_KHZ Internal 1 kHz clock

BUS Bus clock

enum WdtInterval_t

Defines the available Watchdog Timer intervals

Enumerator

NONE Watchdog timer is disabled

INT_32 1 kHz internal clk source * 32, or 32 ms

INT_256 1 kHz internal clk source * 256, or 256 ms

INT_1024 1 kHz internal clk source * 1024, or 1.024 s

INT_8192 For Bus clock, Watchdog clock source * 8192

INT_65536 For Bus clock, Watchdog clock source * 65536

INT_262144 For Bus clock, Watchdog clock source * 262144

Function Documentation

[WdtConfig_t](#) const* const Wdt_ConfigGet (void)

Description:

This function return a pointer to the Wdt configuration structure.

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)

POST-CONDITION: A constant pointer to the first member of the configuration table will be returned.

Returns:

A pointer to the configuration table.

Example Example:

```
1 const WdtConfig_t * WdtConfig = Wdt_ConfigGet();
2
3 Wdt_Init(WdtConfig);
```

See also:

- [Wdt_ConfigGet](#)
- [Wdt_Init](#)
- [Wdt_Enable](#)
- [Wdt_Disable](#)
- [Wdt_Clear](#)
- [Wdt_Reset](#)
- [Wdt_RegisterWrite](#)
- [Wdt_RegisterRead](#)

- HISTORY OF CHANGES -

Date	Software Version	Initials	Description
09/01/2015	1.0.0	JWB	Interface Created

Index

`_10_BIT_SINGLE`
 [adc_cfg.h](#), 48

`_11_BITT_DIFF`
 [adc_cfg.h](#), 48

`_12_BIT_SINGLE`
 [adc_cfg.h](#), 48

`_13_BITT_DIFF`
 [adc_cfg.h](#), 48

`_16_BIT_DIFF`
 [adc_cfg.h](#), 48

`_16_BIT_SINGLE`
 [adc_cfg.h](#), 48

`_8_BIT_SINGLE`
 [adc_cfg.h](#), 48

`_9_BIT_DIFF`
 [adc_cfg.h](#), 48

`Abs`
 [constants.h](#), 57

`ACLOCK`
 [uart_cfg.h](#), 219

`ACTIVE_HIGH`
 [constants.h](#), 59

`ACTIVE_LOW`
 [constants.h](#), 59

`AD0`
 [adc_cfg.h](#), 50

`AD1`
 [adc_cfg.h](#), 50

`AD10`
 [adc_cfg.h](#), 50

`AD11`
 [adc_cfg.h](#), 50

`AD12`
 [adc_cfg.h](#), 50

`AD13`
 [adc_cfg.h](#), 50

`AD14`
 [adc_cfg.h](#), 50

`AD15`
 [adc_cfg.h](#), 50

`AD16`
 [adc_cfg.h](#), 50

`AD17`
 [adc_cfg.h](#), 50

`AD18`
 [adc_cfg.h](#), 50

`AD19`
 [adc_cfg.h](#), 50

`AD2`
 [adc_cfg.h](#), 50

`AD20`
 [adc_cfg.h](#), 50

`AD21`

- adc_cfg.h, 50
- AD22
 - adc_cfg.h, 50
- AD23
 - adc_cfg.h, 50
- AD3
 - adc_cfg.h, 50
- AD4
 - adc_cfg.h, 50
- AD5
 - adc_cfg.h, 50
- AD6
 - adc_cfg.h, 50
- AD7
 - adc_cfg.h, 50
- AD8
 - adc_cfg.h, 50
- AD9
 - adc_cfg.h, 50
- adc.c, 26
 - Adc_AverageSet, 28
 - Adc_Calibrate, 26
 - Adc_CallbackRegister, 35
 - Adc_CompareSet, 27
 - Adc_EndConversion, 32
 - Adc_Init, 29
 - Adc_PowerDown, 30
 - Adc_PretriggerSet, 29
 - Adc_RegisterRead, 34
 - Adc_RegisterWrite, 33
 - Adc_ResultGet, 33
 - Adc_StartConversion, 31
- adc.h, 37
 - Adc_CallbackRegister, 42
 - Adc_EndConversion, 39
 - Adc_GetAdcDoneFlag, 37
 - Adc_PowerDown, 37
 - Adc_RegisterRead, 41
 - Adc_RegisterWrite, 41
 - Adc_ResultGet, 40
 - Adc_StartConversion, 38
 - AdcError_t, 37
- Adc_AverageSet
 - adc.c, 28
- Adc_Calibrate
 - adc.c, 26
- Adc_CallbackRegister
 - adc.c, 35
 - adc.h, 42
- adc_cfg.c, 44
 - Adc_ConfigGet, 44
 - AdcConfig, 45
- adc_cfg.h, 46
 - _10_BIT_SINGLE, 48
 - _11_BITT_DIFF, 48
 - _12_BIT_SINGLE, 48
 - _13_BITT_DIFF, 48
 - _16_BIT_DIFF, 48

_16_BIT_SINGLE, 48
_8_BIT_SINGLE, 48
_9_BIT_DIFF, 48
AD0, 50
AD1, 50
AD10, 50
AD11, 50
AD12, 50
AD13, 50
AD14, 50
AD15, 50
AD16, 50
AD17, 50
AD18, 50
AD19, 50
AD2, 50
AD20, 50
AD21, 50
AD22, 50
AD23, 50
AD3, 50
AD4, 50
AD5, 50
AD6, 50
AD7, 50
AD8, 50
AD9, 50
Adc_ConfigGet, 50
ADC_DISABLE, 50
AdcAvg_t, 47
AdcChannel_t, 49
AdcClockSrc_t, 47
AdcCompare_t, 49
AdcConvSpeed_t, 48
AdcDiv_t, 48
AdcLongTime_t, 47
AdcMode_t, 47
AdcMux_t, 48
AdcPretrigger_t, 49
AdcResolution_t, 48
AdcSampleMode_t, 49
AdcSampleTime_t, 48
AdcTrigger_t, 49
AdcVRefSrc_t, 47
ALT_CLK, 47
ASYNC, 47
AVG_16, 48
AVG_32, 48
AVG_4, 47
AVG_8, 48
BANDGAP, 50
BUS_CLK, 47
BUS_DIV2, 47
CH_A, 48
CH_B, 49
CLKDIV_1, 48
CLKDIV_2, 48
CLKDIV_4, 48

CLKDIV_8, 48
CMP0, 49
COMPARE_VAL_1, 46
COMPARE_VAL_2, 47
CONTINUOUS, 47
ENABLE_A, 49
ENABLE_B, 49
EXT_TRIG, 49
GREATER_THAN_BOTH, 49
GREATER_THAN_SINGLE, 49
HARDWARE, 49
HIGH_SPEED, 48
LESS_THAN_BOTH, 49
LESS_THAN_SINGLE, 49
LONG_SAMPLE, 48
LPTMR0, 49
MAX_ADC_CHANNELS, 50
NO_AVG, 47
NORMAL_SPEED, 48
NUM_CAL_REGISTERS, 46
PIT0, 49
PIT1, 49
RTC_ALARM, 49
RTC_SEC, 49
SHORT_SAMPLE, 48
SINGLE, 47
SOFTWARE, 49
TEMP, 50
TPM0, 49
TPM1, 49
TPM2, 49
V_ALT, 47
V_REF, 47
V_REFSH, 50
V_REFSL, 50
X12, 47
X2, 47
X20, 47
X6, 47

Adc_CompareSet

adc.c, 27

Adc_ConfigGet

adc_cfg.c, 44

adc_cfg.h, 50

ADC_DISABLE

adc_cfg.h, 50

Adc_EndConversion

adc.c, 32

adc.h, 39

Adc_GetAdcDoneFlag

adc.h, 37

Adc_Init

adc.c, 29

Adc_PowerDown

adc.c, 30

adc.h, 37

Adc_PretriggerSet

adc.c, 29

Adc_RegisterRead
 adc.c, 34
 adc.h, 41
Adc_RegisterWrite
 adc.c, 33
 adc.h, 41
Adc_ResultGet
 adc.c, 33
 adc.h, 40
Adc_StartConversion
 adc.c, 31
 adc.h, 38
AdcAvg
 AdcConfig_t, 5
AdcAvg_t
 adc_cfg.h, 47
AdcChannel_t
 adc_cfg.h, 49
AdcClockSrc_t
 adc_cfg.h, 47
AdcCompare_t
 adc_cfg.h, 49
AdcConfig
 adc_cfg.c, 45
AdcConfig_t, 5
 AdcAvg, 5
 AdcEnable, 5
 ClkDiv, 6
 ClkSrc, 6
 CompareMode, 6
 ConvMode, 5
 ConvTrigger, 6
 DiffMode, 6
 IntEnable, 6
 LngSampleTime, 6
 MuxSelect, 6
 OperationSpeed, 6
 PowerMode, 6
 Pretrigger, 6
 Resolution, 5
 SampleTime, 6
 TriggerSrc, 6
 VRefSrc, 6
AdcConvSpeed_t
 adc_cfg.h, 48
AdcDiv_t
 adc_cfg.h, 48
AdcEnable
 AdcConfig_t, 5
AdcError_t
 adc.h, 37
AdcLongTime_t
 adc_cfg.h, 47
AdcMode_t
 adc_cfg.h, 47
AdcMux_t
 adc_cfg.h, 48
AdcPretrigger_t

- adc_cfg.h, 49
- AdcResolution_t
 - adc_cfg.h, 48
- AdcSampleMode_t
 - adc_cfg.h, 49
- AdcSampleTime_t
 - adc_cfg.h, 48
- AdcTrigger_t
 - adc_cfg.h, 49
- AdcVRefSrc_t
 - adc_cfg.h, 47
- AddrType
 - I2CConfig_t, 10
- ALT_CLK
 - adc_cfg.h, 47
- ASYNC
 - adc_cfg.h, 47
- AUX_CLK
 - flash_cfg.h, 98
- AVG_16
 - adc_cfg.h, 48
- AVG_32
 - adc_cfg.h, 48
- AVG_4
 - adc_cfg.h, 47
- AVG_8
 - adc_cfg.h, 48
- BANDGAP
 - adc_cfg.h, 50
- BaudRate
 - I2CConfig_t, 10
 - SpiConfig_t, 16
 - UartBaud_t, 22
 - UartConfig_t, 23
- Bidirection
 - SpiConfig_t, 16
- BIN
 - constants.h, 60
- bit
 - constants.h, 58
- BitDirection
 - UartConfig_t, 23
- bitRead
 - constants.h, 58
- BITS_EIGHT
 - uart_cfg.h, 220
- BITS_NINE
 - uart_cfg.h, 220
- BitShift_t
 - constants.h, 61
- bitWrite
 - constants.h, 58
- BUS
 - wdt_cfg.h, 239
- BUS_CLK
 - adc_cfg.h, 47
- BUS_DIV2
 - adc_cfg.h, 47

- BusClkDiv
 - McuConfig_t, 14
- Byte_t
 - constants.h, 60
- CANNED_OSC
 - mcu.h, 133
- CH_A
 - adc_cfg.h, 48
- CH_B
 - adc_cfg.h, 49
- Channel
 - DioConfig_t, 8
 - I2CTransfer_t, 11
- ChannelEnable
 - I2CConfig_t, 10
- ChannelName
 - PwmConfig_t, 15
 - SpiConfig_t, 16
- CharacterArray
 - uart.c, 208
- ChipSelect
 - SpiTransfer_t, 18
- CLEAR_FLAG_NUL
 - constants.h, 57
- CLEAR_FLAG_POS
 - constants.h, 57
- clearBits
 - constants.h, 58
- ClkDiv
 - AdcConfig_t, 6
 - FlashConfig_t, 9
- CLKDIV_1
 - adc_cfg.h, 48
- CLKDIV_2
 - adc_cfg.h, 48
- CLKDIV_4
 - adc_cfg.h, 48
- CLKDIV_8
 - adc_cfg.h, 48
- ClkFreq
 - UartBaud_t, 22
- ClkMode
 - TmrConfig_t, 20
- ClkPrescaler
 - TmrConfig_t, 20
- ClkSrc
 - AdcConfig_t, 6
 - FlashConfig_t, 9
 - UartConfig_t, 23
- CLOCK_1_MHZ
 - constants.h, 56
- CLOCK_12_MHZ
 - constants.h, 57
- CLOCK_16_MHZ
 - constants.h, 57
- CLOCK_2_MHZ
 - constants.h, 56
- CLOCK_24_MHZ

constants.h, 57
CLOCK_32_MHZ
constants.h, 57
CLOCK_4_MHZ
constants.h, 56
CLOCK_40_MHZ
constants.h, 57
CLOCK_48_MHZ
constants.h, 57
CLOCK_7_37_MHZ
constants.h, 56
CLOCK_8_MHZ
constants.h, 57
CLOCK_DIVIDEBY_1
constants.h, 60
CLOCK_DIVIDEBY_128
constants.h, 60
CLOCK_DIVIDEBY_16
constants.h, 60
CLOCK_DIVIDEBY_2
constants.h, 60
CLOCK_DIVIDEBY_256
constants.h, 60
CLOCK_DIVIDEBY_32
constants.h, 60
CLOCK_DIVIDEBY_4
constants.h, 60
CLOCK_DIVIDEBY_64
constants.h, 60
CLOCK_DIVIDEBY_8
constants.h, 60
CLOCK_PERIOD_NS_1_MHZ
constants.h, 60
CLOCK_PERIOD_NS_16_MHZ
constants.h, 60
CLOCK_PERIOD_NS_2_MHZ
constants.h, 60
CLOCK_PERIOD_NS_24_MHZ
constants.h, 60
CLOCK_PERIOD_NS_32_MHZ
constants.h, 61
CLOCK_PERIOD_NS_4_MHZ
constants.h, 60
CLOCK_PERIOD_NS_40_MHZ
constants.h, 61
CLOCK_PERIOD_NS_48_MHZ
constants.h, 61
CLOCK_PERIOD_NS_8_MHZ
constants.h, 60
ClockDivide_t
constants.h, 60
ClockMode
McuConfig_t, 12
ClockPeriod_t
constants.h, 60
ClockSource
TmrConfig_t, 20
WdtConfig_t, 25

- CMPO
 - adc_cfg.h, 49
- COMPARE_VAL_1
 - adc_cfg.h, 46
- COMPARE_VAL_2
 - adc_cfg.h, 47
- CompareMode
 - AdcConfig_t, 6
- constants.h, 52
 - Abs, 57
 - ACTIVE_HIGH, 59
 - ACTIVE_LOW, 59
 - BIN, 60
 - bit, 58
 - bitRead, 58
 - BitShift_t, 61
 - bitWrite, 58
 - Byte_t, 60
 - CLEAR_FLAG_NUL, 57
 - CLEAR_FLAG_POS, 57
 - clearBits, 58
 - CLOCK_1_MHZ, 56
 - CLOCK_12_MHZ, 57
 - CLOCK_16_MHZ, 57
 - CLOCK_2_MHZ, 56
 - CLOCK_24_MHZ, 57
 - CLOCK_32_MHZ, 57
 - CLOCK_4_MHZ, 56
 - CLOCK_40_MHZ, 57
 - CLOCK_48_MHZ, 57
 - CLOCK_7_37_MHZ, 56
 - CLOCK_8_MHZ, 57
 - CLOCK_DIVIDEBY_1, 60
 - CLOCK_DIVIDEBY_128, 60
 - CLOCK_DIVIDEBY_16, 60
 - CLOCK_DIVIDEBY_2, 60
 - CLOCK_DIVIDEBY_256, 60
 - CLOCK_DIVIDEBY_32, 60
 - CLOCK_DIVIDEBY_4, 60
 - CLOCK_DIVIDEBY_64, 60
 - CLOCK_DIVIDEBY_8, 60
 - CLOCK_PERIOD_NS_1_MHZ, 60
 - CLOCK_PERIOD_NS_16_MHZ, 60
 - CLOCK_PERIOD_NS_2_MHZ, 60
 - CLOCK_PERIOD_NS_24_MHZ, 60
 - CLOCK_PERIOD_NS_32_MHZ, 61
 - CLOCK_PERIOD_NS_4_MHZ, 60
 - CLOCK_PERIOD_NS_40_MHZ, 61
 - CLOCK_PERIOD_NS_48_MHZ, 61
 - CLOCK_PERIOD_NS_8_MHZ, 60
 - ClockDivide_t, 60
 - ClockPeriod_t, 60
 - Constrain, 57
 - DEC, 59
 - DEG_TO_RAD, 54
 - Degrees, 58
 - DISABLED, 59
 - EIGHT_BITS, 61

EIGHT_BYTES, 60
EIGHTEEN_BITS, 61
ELEVEN_BITS, 61
ENABLED, 59
EPSILON, 54
FALSE, 59
FIFTEEN_BITS, 61
FIVE_BITS, 61
FIVE_BYTES, 60
FOUR_BITS, 61
FOUR_BYTES, 60
FOURTEEN_BITS, 61
HALF_PI, 54
HEX, 59
HIGH, 59
highByte, 59
INACTIVE_HIGH, 59
INACTIVE_LOW, 59
INPUT, 59
LogicEnum_t, 59
LOW, 59
lowByte, 58
Max, 58
Min, 58
NINE_BITS, 61
NINE_BYTES, 60
NINETEEN_BITS, 61
null, 54
num, 62
NumberBase_t, 59
OCT, 60
OFF, 59
ON, 59
ONE_BIT, 61
ONE_BYTES, 60
OnOff_t, 59
OUTPUT, 59
PI, 54
PinLevelEnum_t, 59
PinModeEnum_t, 59
RAD_TO_DEG, 54
Radians, 58
REGBIT0, 54
REGBIT1, 54
REGBIT10, 55
REGBIT11, 55
REGBIT12, 55
REGBIT13, 55
REGBIT14, 55
REGBIT15, 55
REGBIT16, 55
REGBIT17, 55
REGBIT18, 55
REGBIT19, 55
REGBIT2, 54
REGBIT20, 55
REGBIT21, 56
REGBIT22, 56

REGBIT23, 56
REGBIT24, 56
REGBIT25, 56
REGBIT26, 56
REGBIT27, 56
REGBIT28, 56
REGBIT29, 56
REGBIT3, 54
REGBIT30, 56
REGBIT31, 56
REGBIT4, 54
REGBIT5, 54
REGBIT6, 55
REGBIT7, 55
REGBIT8, 55
REGBIT9, 55
Round, 58
setBits, 58
SEVEN_BITS, 61
SEVEN_BYTES, 60
SEVENTEEN_BITS, 61
SIX_BITS, 61
SIX_BYTES, 60
SIXTEEN_BITS, 61
Sq, 58
TEN_BITS, 61
THIRTEEN_BITS, 61
THIRTY_BITS, 61
THIRTYONE_BITS, 62
THIRTYTWO_BITS, 62
THREE_BITS, 61
THREE_BYTES, 60
TRUE, 59
TWELVE_BITS, 61
TWENTY_BITS, 61
TWENTYEIGHT_BITS, 61
TWENTYFIVE_BITS, 61
TWENTYFOUR_BITS, 61
TWENTYNINE_BITS, 61
TWENTYONE_BITS, 61
TWENTYSEVEN_BITS, 61
TWENTYSIX_BITS, 61
TWENTYTHREE_BITS, 61
TWENTYTWO_BITS, 61
TWO_BITS, 61
TWO_BYTES, 60
TWO_PI, 54
ZERO, 54
ZERO_BITS, 61
Constrain
 constants.h, 57
CONTINUOUS
 adc_cfg.h, 47
ConvMode
 AdcConfig_t, 5
ConvTrigger
 AdcConfig_t, 6
CRYSTAL

- mcu.h, 133
- CS_ACTIVE_HIGH
 - spi.h, 174
- CS_ACTIVE_LOW
 - spi.h, 174
- CsPolarity
 - SpiTransfer_t, 18
- Data
 - DioConfig_t, 8
- DataBuf
 - I2CTransfer_t, 11
- DataLength
 - UartConfig_t, 24
- DEC
 - constants.h, 59
- DEG_TO_RAD
 - constants.h, 54
- Degrees
 - constants.h, 58
- Delimiter
 - UartConfig_t, 24
- DiffMode
 - AdcConfig_t, 6
- dio.c, 63
 - Dio_CallbackRegister, 70
 - Dio_ChannelDirectionSet, 67
 - Dio_ChannelModeSet, 66
 - Dio_ChannelRead, 64
 - Dio_ChannelToggle, 66
 - Dio_ChannelWrite, 65
 - Dio_Init, 63
 - Dio_RegisterRead, 69
 - Dio_RegisterWrite, 68
- dio.h, 71
 - Dio_CallbackRegister, 78
 - Dio_ChannelDirectionSet, 75
 - Dio_ChannelModeSet, 74
 - Dio_ChannelRead, 72
 - Dio_ChannelToggle, 74
 - Dio_ChannelWrite, 73
 - Dio_Init, 71
 - Dio_RegisterRead, 77
 - Dio_RegisterWrite, 76
- Dio_CallbackRegister
 - dio.c, 70
 - dio.h, 78
- dio_cfg.c, 79
 - Dio_ConfigGet, 79
 - DioConfig, 80
- dio_cfg.h, 81
 - Dio_ConfigGet, 82
 - DIO_HIGH, 81
 - DIO_MAX_PIN_NUMBER, 82
 - DIO_PIN_STATE_MAX, 81
 - DioChannel_t, 82
 - DioMode_t, 82
 - DioPinState_t, 81
 - DioResistor_t, 82

- DioSlew_t, 82
- FAST, 82
- FCPU_HB, 82
- NUMBER_OF_CHANNELS_PER_PORT, 81
- NUMBER_OF_PORTS, 81
- PORT1_1, 82
- PORT1_2, 82
- PORT1_3, 82
- PORT1_5, 82
- PORT1_6, 82
- PORT1_7, 82
- SLOW, 82
- UHF_SEL, 82
- Dio_ChannelDirectionSet
 - dio.c, 67
 - dio.h, 75
- Dio_ChannelModeSet
 - dio.c, 66
 - dio.h, 74
- Dio_ChannelRead
 - dio.c, 64
 - dio.h, 72
- Dio_ChannelToggle
 - dio.c, 66
 - dio.h, 74
- Dio_ChannelWrite
 - dio.c, 65
 - dio.h, 73
- Dio_ConfigGet
 - dio_cfg.c, 79
 - dio_cfg.h, 82
- DIO_HIGH
 - dio_cfg.h, 81
- Dio_Init
 - dio.c, 63
 - dio.h, 71
- DIO_MAX_PIN_NUMBER
 - dio_cfg.h, 82
- DIO_PIN_STATE_MAX
 - dio_cfg.h, 81
- Dio_RegisterRead
 - dio.c, 69
 - dio.h, 77
- Dio_RegisterWrite
 - dio.c, 68
 - dio.h, 76
- DioChannel_t
 - dio_cfg.h, 82
- DioConfig
 - dio_cfg.c, 80
- DioConfig_t, 8
 - Channel, 8
 - Data, 8
 - Direction, 8
 - Function, 8
 - Resistor, 8
- DioMode_t
 - dio_cfg.h, 82

DioPinState_t
dio_cfg.h, 81

DioResistor_t
dio_cfg.h, 82

DioSlew_t
dio_cfg.h, 82

Direction
DioConfig_t, 8
SpiTransfer_t, 18

DISABLED
constants.h, 59

DIV_1
mcu_cfg.h, 140

DIV_1024
mcu_cfg.h, 140

DIV_128
mcu_cfg.h, 140

DIV_128_HIGH
mcu_cfg.h, 140

DIV_128_LOW
mcu_cfg.h, 140

DIV_1280
mcu_cfg.h, 140

DIV_1536
mcu_cfg.h, 140

DIV_16
mcu_cfg.h, 140

DIV_2
mcu_cfg.h, 140

DIV_256
mcu_cfg.h, 140

DIV_32
mcu_cfg.h, 140

DIV_32_HIGH
mcu_cfg.h, 140

DIV_32_LOW
mcu_cfg.h, 140

DIV_4
mcu_cfg.h, 140

DIV_512
mcu_cfg.h, 140

DIV_64
mcu_cfg.h, 140

DIV_64_HIGH
mcu_cfg.h, 140

DIV_64_LOW
mcu_cfg.h, 140

DIV_8
mcu_cfg.h, 140

DutyCycle
PwmConfig_t, 15

EIGHT_BITS
constants.h, 61

EIGHT_BYTES
constants.h, 60

EIGHTEEN_BITS
constants.h, 61

ELEVEN_BITS

- constants.h, 61
- ENABLE_A
 - adc_cfg.h, 49
- ENABLE_B
 - adc_cfg.h, 49
- ENABLED
 - constants.h, 59
- EPSILON
 - constants.h, 54
- EVEN
 - uart_cfg.h, 219
- ExRefSrc
 - McuConfig_t, 12
- ExRefStopMode
 - McuConfig_t, 13
- EXT_CLK
 - mcu_cfg.h, 141
- EXT_TRIG
 - adc_cfg.h, 49
- EXTERNAL_CLK
 - tmr_cfg.h, 198
- FALSE
 - constants.h, 59
- FAST
 - dio_cfg.h, 82
- FAST_INT
 - mcu_cfg.h, 141
- FCPU_HB
 - dio_cfg.h, 82
- FEI_MODE
 - mcu_cfg.h, 140
- FIFTEEN_BITS
 - constants.h, 61
- FIVE_BITS
 - constants.h, 61
- FIVE_BYTES
 - constants.h, 60
- flash.c, 84
 - Flash_CallbackRegister, 88
 - Flash_Erase, 86
 - Flash_Init, 84
 - Flash_Read, 85
 - Flash_RegisterRead, 88
 - Flash_RegisterWrite, 87
 - Flash_Write, 85
- flash.h, 90
 - Flash_CallbackRegister, 94
 - Flash_Erase, 92
 - Flash_Init, 90
 - Flash_Read, 91
 - Flash_RegisterRead, 94
 - Flash_RegisterWrite, 93
 - Flash_Write, 91
- Flash_CallbackRegister
 - flash.c, 88
 - flash.h, 94
- flash_cfg.c, 96
 - Flash_ConfigGet, 96

- FlashConfig, 97
- flash_cfg.h, 98
 - AUX_CLK, 98
 - Flash_ConfigGet, 98
 - FlashClkSrc_t, 98
 - M_CLK, 98
 - SYS_CLK, 98
- Flash_ConfigGet
 - flash_cfg.c, 96
 - flash_cfg.h, 98
- Flash_Erase
 - flash.c, 86
 - flash.h, 92
- Flash_Init
 - flash.c, 84
 - flash.h, 90
- Flash_Read
 - flash.c, 85
 - flash.h, 91
- Flash_RegisterRead
 - flash.c, 88
 - flash.h, 94
- Flash_RegisterWrite
 - flash.c, 87
 - flash.h, 93
- Flash_Write
 - flash.c, 85
 - flash.h, 91
- FlashClkSrc_t
 - flash_cfg.h, 98
- FlashConfig
 - flash_cfg.c, 97
- FlashConfig_t, 9
 - ClkDiv, 9
 - ClkSrc, 9
- FLL_PLL
 - tmr_cfg.h, 198
- FLLDiv
 - McuConfig_t, 12
- FLLFrequency
 - McuConfig_t, 12
- FOUR_BITS
 - constants.h, 61
- FOUR_BYTES
 - constants.h, 60
- FOURTEEN_BITS
 - constants.h, 61
- Function
 - DioConfig_t, 8
- GetInstructionClock
 - mcu_cfg.h, 139
- GetOscillatorFrequency
 - mcu_cfg.h, 139
- GetSystemClock
 - mcu_cfg.h, 139
- GREATER_THAN_BOTH
 - adc_cfg.h, 49
- GREATER_THAN_SINGLE

- adc_cfg.h, 49
- HALF_PI
 - constants.h, 54
- HARDWARE
 - adc_cfg.h, 49
- HEX
 - constants.h, 59
- HIGH
 - constants.h, 59
- HIGH_GAIN
 - mcu_cfg.h, 140
- HIGH_RANGE
 - mcu_cfg.h, 140
- HIGH_SPEED
 - adc_cfg.h, 48
- highByte
 - constants.h, 59
- i2c.c, 100
 - I2c_CallbackRegister, 109
 - I2c_DeInit, 102
 - I2c_FrequencySet, 100
 - I2c_Init, 101
 - I2c_PowerModeSet, 103
 - I2c_RegisterRead, 108
 - I2c_RegisterWrite, 107
 - I2c_SlaveAddressSet, 104
 - I2c_Transfer, 104
 - I2c0_ISR, 105
 - I2c1_ISR, 106
- i2c.h, 111
 - I2c_CallbackRegister, 113
 - I2C_HALT, 112
 - I2C_OPERATE, 112
 - I2C_RECEIVE, 111
 - I2c_RegisterRead, 113
 - I2c_RegisterWrite, 112
 - I2C_TRANSMIT, 111
 - I2CPowerMode_t, 111
 - I2CTransferMode_t, 111
 - NUM_FREQ_MULT, 111
- I2C_0
 - i2c_cfg.h, 116
- I2C_1
 - i2c_cfg.h, 116
- I2C_10bit
 - i2c_cfg.h, 115
- I2C_7bit
 - i2c_cfg.h, 115
- I2c_CallbackRegister
 - i2c.c, 109
 - i2c.h, 113
- i2c_cfg.h, 115
 - I2C_0, 116
 - I2C_1, 116
 - I2C_10bit, 115
 - I2C_7bit, 115
 - I2C_ConfigGet, 116
 - I2C_INTERRUPT, 115

- I2C_MASTER, 116
- I2C_POLLING_DELAY, 115
- I2C_SLAVE, 115
- I2CAddr_t, 115
- I2CChannel_t, 116
- I2CMode_t, 115
- NUM_I2C_CHANNELS, 116
- I2C_Channel
 - I2CConfig_t, 10
- I2C_ConfigGet
 - i2c_cfg.h, 116
- I2c_DeInit
 - i2c.c, 102
- I2c_FrequencySet
 - i2c.c, 100
- I2C_HALT
 - i2c.h, 112
- I2c_Init
 - i2c.c, 101
- I2C_INTERRUPT
 - i2c_cfg.h, 115
- I2C_MASTER
 - i2c_cfg.h, 116
- I2C_OPERATE
 - i2c.h, 112
- I2C_POLLING_DELAY
 - i2c_cfg.h, 115
- I2c_PowerModeSet
 - i2c.c, 103
- I2C_RECEIVE
 - i2c.h, 111
- I2c_RegisterRead
 - i2c.c, 108
 - i2c.h, 113
- I2c_RegisterWrite
 - i2c.c, 107
 - i2c.h, 112
- I2C_SLAVE
 - i2c_cfg.h, 115
- I2c_SlaveAddressSet
 - i2c.c, 104
- I2c_Transfer
 - i2c.c, 104
- I2C_TRANSMIT
 - i2c.h, 111
- I2c0_ISR
 - i2c.c, 105
- I2c1_ISR
 - i2c.c, 106
- I2CAddr_t
 - i2c_cfg.h, 115
- I2CChannel_t
 - i2c_cfg.h, 116
- I2CConfig_t, 10
 - AddrType, 10
 - BaudRate, 10
 - ChannelEnable, 10
 - I2C_Channel, 10

- I2CId, 10
- Mode, 10
- I2CId
 - I2CConfig_t, 10
- I2CMode_t
 - i2c_cfg.h, 115
- I2CPowerMode_t
 - i2c.h, 111
- I2CTransfer_t, 11
 - Channel, 11
 - DataBuf, 11
 - Mode, 11
 - NumBytes, 11
 - SlaveId, 11
- I2CTransferMode_t
 - i2c.h, 111
- INACTIVE_HIGH
 - constants.h, 59
- INACTIVE_LOW
 - constants.h, 59
- INPUT
 - constants.h, 59
- INT_1024
 - wdt_cfg.h, 239
- INT_256
 - wdt_cfg.h, 239
- INT_262144
 - wdt_cfg.h, 240
- INT_32
 - wdt_cfg.h, 239
- INT_65536
 - wdt_cfg.h, 239
- INT_8192
 - wdt_cfg.h, 239
- INT_CLK
 - tmr_cfg.h, 198
- IntClkEnable
 - McuConfig_t, 13
- IntEnable
 - AdcConfig_t, 6
 - PwmConfig_t, 15
 - UartConfig_t, 24
- IntEnabled
 - TmrConfig_t, 20
- Interval
 - TmrConfig_t, 21
 - WdtConfig_t, 25
- IntFastClkDiv
 - McuConfig_t, 13
- IntPriority
 - TmrConfig_t, 21
- IntRefClkSelec
 - McuConfig_t, 13
- IntRefStopMode
 - McuConfig_t, 13
- isr.c, 118
 - Isr_CriticalSectionEnd, 122
 - Isr_CriticalSectionStart, 121

- Isr_Disable, 119
- Isr_Enable, 118
- Isr_GlobalDisable, 120
- Isr_GlobalEnable, 119
- Isr_GlobalStateGet, 121
- isr.h, 124
 - Isr_CriticalSectionEnd, 128
 - Isr_CriticalSectionStart, 127
 - Isr_Disable, 125
 - ISR_DISABLED, 124
 - Isr_Enable, 124
 - ISR_ENABLED, 124
 - Isr_GlobalDisable, 126
 - Isr_GlobalStateGet, 126
 - IsrState_t, 124
- Isr_CriticalSectionEnd
 - isr.c, 122
 - isr.h, 128
- Isr_CriticalSectionStart
 - isr.c, 121
 - isr.h, 127
- Isr_Disable
 - isr.c, 119
 - isr.h, 125
- ISR_DISABLED
 - isr.h, 124
- Isr_Enable
 - isr.c, 118
 - isr.h, 124
- ISR_ENABLED
 - isr.h, 124
- Isr_GlobalDisable
 - isr.c, 120
 - isr.h, 126
- Isr_GlobalEnable
 - isr.c, 119
- Isr_GlobalStateGet
 - isr.c, 121
 - isr.h, 126
- IsrState_t
 - isr.h, 124
- LESS_THAN_BOTH
 - adc_cfg.h, 49
- LESS_THAN_SINGLE
 - adc_cfg.h, 49
- LngSampleTime
 - AdcConfig_t, 6
- LogicEnum_t
 - constants.h, 59
- LONG_SAMPLE
 - adc_cfg.h, 48
- Loopback
 - UartConfig_t, 23
- LOW
 - constants.h, 59
- LOW_POWER
 - mcu_cfg.h, 140
- LOW_RANGE

- mcu_cfg.h, 140
- lowByte
 - constants.h, 58
- LPTMR0
 - adc_cfg.h, 49
- LSB_FIRST
 - spi.h, 174
- M_CLK
 - flash_cfg.h, 98
- MasterMode
 - SpiConfig_t, 16
- Max
 - constants.h, 58
- MAX_ADC_CHANNELS
 - adc_cfg.h, 50
- mcu.c, 129
 - Mcu_CallbackRegister, 131
 - Mcu_Init, 129
 - Mcu_RegisterRead, 130
 - Mcu_RegisterWrite, 130
- mcu.h, 133
 - CANNED_OSC, 133
 - CRYSTAL, 133
 - Mcu_CallbackRegister, 135
 - Mcu_Init, 133
 - Mcu_RegisterRead, 135
 - Mcu_RegisterWrite, 134
- Mcu_CallbackRegister
 - mcu.c, 131
 - mcu.h, 135
- mcu_cfg.c, 137
 - Mcu_ConfigGet, 137
 - McuConfig, 138
- mcu_cfg.h, 139
 - DIV_1, 140
 - DIV_1024, 140
 - DIV_128, 140
 - DIV_128_HIGH, 140
 - DIV_128_LOW, 140
 - DIV_1280, 140
 - DIV_1536, 140
 - DIV_16, 140
 - DIV_2, 140
 - DIV_256, 140
 - DIV_32, 140
 - DIV_32_HIGH, 140
 - DIV_32_LOW, 140
 - DIV_4, 140
 - DIV_512, 140
 - DIV_64, 140
 - DIV_64_HIGH, 140
 - DIV_64_LOW, 140
 - DIV_8, 140
 - EXT_CLK, 141
 - FAST_INT, 141
 - FEI_MODE, 140
 - GetInstructionClock, 139
 - GetOscillatorFrequency, 139

- GetSystemClock, 139
- HIGH_GAIN, 140
- HIGH_RANGE, 140
- LOW_POWER, 140
- LOW_RANGE, 140
- Mcu_ConfigGet, 141
- McuDiv_t, 140
- McuExtClkSrc_t, 141
- McuFLLFreq_t, 141
- McuFreqMode_t, 140
- McuFreqRange_t, 140
- McuIntRefSrc_t, 141
- McuMode_t, 139
- MHZ_20_25, 141
- MHZ_24, 141
- MHZ_40_50, 141
- MHZ_48, 141
- MHZ_60_75, 141
- MHZ_72, 141
- MHZ_80_100, 141
- MHZ_96, 141
- OSC, 141
- SLOW_INT, 141
- VERY_HIGH_RANGE, 140

- Mcu_ConfigGet
 - mcu_cfg.c, 137
 - mcu_cfg.h, 141

- Mcu_Init
 - mcu.c, 129
 - mcu.h, 133

- Mcu_RegisterRead
 - mcu.c, 130
 - mcu.h, 135

- Mcu_RegisterWrite
 - mcu.c, 130
 - mcu.h, 134

- McuConfig
 - mcu_cfg.c, 138

- McuConfig_t, 12
 - BusClkDiv, 14
 - ClockMode, 12
 - ExRefSrc, 12
 - ExRefStopMode, 13
 - FLLDiv, 12
 - FLLFrequency, 12
 - IntClkEnable, 13
 - IntFastClkDiv, 13
 - IntRefClkSelec, 13
 - IntRefStopMode, 13
 - Osc16P, 13
 - Osc2P, 13
 - Osc4P, 13
 - Osc8P, 13
 - OscillatorMode, 13
 - PLL_VCO_Div, 13
 - PLLDiv, 13
 - PLLStopEnable, 13
 - SysClkDiv, 13

- McuDiv_t
 - mcu_cfg.h, 140
- McuExtClkSrc_t
 - mcu_cfg.h, 141
- McuFLLFreq_t
 - mcu_cfg.h, 141
- McuFreqMode_t
 - mcu_cfg.h, 140
- McuFreqRange_t
 - mcu_cfg.h, 140
- McuIntRefSrc_t
 - mcu_cfg.h, 141
- McuMode_t
 - mcu_cfg.h, 139
- MHZ_20_25
 - mcu_cfg.h, 141
- MHZ_24
 - mcu_cfg.h, 141
- MHZ_40_50
 - mcu_cfg.h, 141
- MHZ_48
 - mcu_cfg.h, 141
- MHZ_60_75
 - mcu_cfg.h, 141
- MHZ_72
 - mcu_cfg.h, 141
- MHZ_80_100
 - mcu_cfg.h, 141
- MHZ_96
 - mcu_cfg.h, 141
- Min
 - constants.h, 58
- Mode
 - I2CConfig_t, 10
 - I2CTransfer_t, 11
- MODULE_CLK
 - tmr_cfg.h, 198
- MSB_FIRST
 - spi.h, 174
- MuxSelect
 - AdcConfig_t, 6
- NINE_BITS
 - constants.h, 61
- NINE_BYTES
 - constants.h, 60
- NINETEEN_BITS
 - constants.h, 61
- NO_AVG
 - adc_cfg.h, 47
- NONE
 - wdt_cfg.h, 239
- NORMAL
 - wdt_cfg.h, 239
- NORMAL_SPEED
 - adc_cfg.h, 48
- null
 - constants.h, 54
- num

- constants.h, 62
- NUM_BAUD_TABLE
 - uart.h, 209
- NUM_CAL_REGISTERS
 - adc_cfg.h, 46
- NUM_FREQ_MULT
 - i2c.h, 111
- NUM_I2C_CHANNELS
 - i2c_cfg.h, 116
- NUM_PWM_CHANNELS
 - pwm_cfg.h, 159
- NUM_TIMERS
 - tmr_cfg.h, 198
- NUM_UART_CHANNELS
 - uart_cfg.h, 220
- NUMBER_OF_CHANNELS_PER_PORT
 - dio_cfg.h, 81
- NUMBER_OF_PORTS
 - dio_cfg.h, 81
- NumberBase_t
 - constants.h, 59
- NumBytes
 - I2CTransfer_t, 11
 - SpiTransfer_t, 18
- OCT
 - constants.h, 60
- ODD
 - uart_cfg.h, 219
- OFF
 - constants.h, 59
- ON
 - constants.h, 59
- ONE_BIT
 - constants.h, 61
- ONE_BYTES
 - constants.h, 60
- ONE_KHZ
 - wdt_cfg.h, 239
- OnOff_t
 - constants.h, 59
- OperationSpeed
 - AdcConfig_t, 6
- OSC
 - mcu_cfg.h, 141
- Osc16P
 - McuConfig_t, 13
- Osc2P
 - McuConfig_t, 13
- Osc4P
 - McuConfig_t, 13
- Osc8P
 - McuConfig_t, 13
- OSCERCLK
 - tmr_cfg.h, 198
- OscillatorMode
 - McuConfig_t, 13
- OUTPUT
 - constants.h, 59

- Oversampling
 - UartBaud_t, 22
- ParityType
 - UartConfig_t, 24
- Phase
 - SpiTransfer_t, 18
- PHASE_HIGH
 - spi.h, 173
- PHASE_LOW
 - spi.h, 173
- PI
 - constants.h, 54
- PinLevelEnum_t
 - constants.h, 59
- PinModeEnum_t
 - constants.h, 59
- PIT0
 - adc_cfg.h, 49
- PIT1
 - adc_cfg.h, 49
- PLL_VCO_Div
 - McuConfig_t, 13
- PLLDiv
 - McuConfig_t, 13
- PLLStopEnable
 - McuConfig_t, 13
- Polarity
 - SpiTransfer_t, 18
- POLARITY_HIGH
 - spi.h, 173
- POLARITY_LOW
 - spi.h, 173
- PORT1_1
 - dio_cfg.h, 82
- PORT1_2
 - dio_cfg.h, 82
- PORT1_3
 - dio_cfg.h, 82
- PORT1_5
 - dio_cfg.h, 82
- PORT1_6
 - dio_cfg.h, 82
- PORT1_7
 - dio_cfg.h, 82
- PowerMode
 - AdcConfig_t, 6
- Pretrigger
 - AdcConfig_t, 6
- pwm.c, 143
 - Pwm_CallbackRegister, 149
 - Pwm_Disable, 146
 - Pwm_DutyCycleSet, 144
 - Pwm_Enable, 145
 - Pwm_FrequencySet, 145
 - Pwm_Init, 143
 - Pwm_RegisterRead, 148
 - Pwm_RegisterWrite, 147
- pwm.h, 150

- Pwm_CallbackRegister, 156
- Pwm_Disable, 153
- Pwm_DutyCycleSet, 151
- Pwm_Enable, 152
- Pwm_FrequencySet, 152
- Pwm_Init, 150
- Pwm_RegisterRead, 155
- Pwm_RegisterWrite, 154
- Pwm_CallbackRegister
 - pwm.c, 149
 - pwm.h, 156
- pwm_cfg.c, 157
 - Pwm_ConfigGet, 157
 - PwmConfig, 158
- pwm_cfg.h, 159
 - NUM_PWM_CHANNELS, 159
 - Pwm_ConfigGet, 159
 - PWM0_0, 159
 - PWM0_1, 159
 - PWM0_2, 159
 - PWM0_3, 159
 - PWM0_4, 159
 - PWM0_5, 159
 - PWM1_0, 159
 - PWM1_1, 159
 - PWM2_0, 159
 - PWM2_1, 159
 - PwmChannel_t, 159
- Pwm_ConfigGet
 - pwm_cfg.c, 157
 - pwm_cfg.h, 159
- Pwm_Disable
 - pwm.c, 146
 - pwm.h, 153
- Pwm_DutyCycleSet
 - pwm.c, 144
 - pwm.h, 151
- Pwm_Enable
 - pwm.c, 145
 - pwm.h, 152
- Pwm_FrequencySet
 - pwm.c, 145
 - pwm.h, 152
- Pwm_Init
 - pwm.c, 143
 - pwm.h, 150
- Pwm_RegisterRead
 - pwm.c, 148
 - pwm.h, 155
- Pwm_RegisterWrite
 - pwm.c, 147
 - pwm.h, 154
- PWM0_0
 - pwm_cfg.h, 159
- PWM0_1
 - pwm_cfg.h, 159
- PWM0_2
 - pwm_cfg.h, 159

PWM0_3
 pwm_cfg.h, 159

PWM0_4
 pwm_cfg.h, 159

PWM0_5
 pwm_cfg.h, 159

PWM1_0
 pwm_cfg.h, 159

PWM1_1
 pwm_cfg.h, 159

PWM2_0
 pwm_cfg.h, 159

PWM2_1
 pwm_cfg.h, 159

PwmChannel_t
 pwm_cfg.h, 159

PwmConfig
 pwm_cfg.c, 158

PwmConfig_t, 15
 ChannelName, 15
 DutyCycle, 15
 IntEnable, 15
 PwmEnable, 15

PwmEnable
 PwmConfig_t, 15

RAD_TO_DEG
 constants.h, 54

Radians
 constants.h, 58

REGBIT0
 constants.h, 54

REGBIT1
 constants.h, 54

REGBIT10
 constants.h, 55

REGBIT11
 constants.h, 55

REGBIT12
 constants.h, 55

REGBIT13
 constants.h, 55

REGBIT14
 constants.h, 55

REGBIT15
 constants.h, 55

REGBIT16
 constants.h, 55

REGBIT17
 constants.h, 55

REGBIT18
 constants.h, 55

REGBIT19
 constants.h, 55

REGBIT2
 constants.h, 54

REGBIT20
 constants.h, 55

REGBIT21

- constants.h, 56
- REGBIT22
 - constants.h, 56
- REGBIT23
 - constants.h, 56
- REGBIT24
 - constants.h, 56
- REGBIT25
 - constants.h, 56
- REGBIT26
 - constants.h, 56
- REGBIT27
 - constants.h, 56
- REGBIT28
 - constants.h, 56
- REGBIT29
 - constants.h, 56
- REGBIT3
 - constants.h, 54
- REGBIT30
 - constants.h, 56
- REGBIT31
 - constants.h, 56
- REGBIT4
 - constants.h, 54
- REGBIT5
 - constants.h, 54
- REGBIT6
 - constants.h, 55
- REGBIT7
 - constants.h, 55
- REGBIT8
 - constants.h, 55
- REGBIT9
 - constants.h, 55
- Resistor
 - DioConfig_t, 8
- Resolution
 - AdcConfig_t, 5
- Round
 - constants.h, 58
- RTC_ALARM
 - adc_cfg.h, 49
- RTC_SEC
 - adc_cfg.h, 49
- RX_ONLY
 - uart_cfg.h, 220
- RX_TX
 - uart_cfg.h, 220
- SampleTime
 - AdcConfig_t, 6
- setBits
 - constants.h, 58
- SEVEN_BITS
 - constants.h, 61
- SEVEN_BYTES
 - constants.h, 60
- SEVENTEEN_BITS

- constants.h, 61
- SHORT_SAMPLE
 - adc_cfg.h, 48
- SINGLE
 - adc_cfg.h, 47
- SIX_BITS
 - constants.h, 61
- SIX_BYTES
 - constants.h, 60
- SIXTEEN_BITS
 - constants.h, 61
- SlaveId
 - I2CTransfer_t, 11
- SLOW
 - dio_cfg.h, 82
- SLOW_INT
 - mcu_cfg.h, 141
- SOFTWARE
 - adc_cfg.h, 49
- spi.c, 161
 - Spi_BaudRateSet, 161
 - Spi_CalcDelay, 163
 - Spi_CallbackRegister, 171
 - Spi_ChipSelectClear, 168
 - Spi_ChipSelectSet, 167
 - Spi_DeInit, 165
 - Spi_Init, 164
 - Spi_RegisterRead, 170
 - Spi_RegisterWrite, 169
 - Spi_Setup, 166
 - Spi_SSSet, 162
 - Spi_Transfer, 165
- spi.h, 173
 - CS_ACTIVE_HIGH, 174
 - CS_ACTIVE_LOW, 174
 - LSB_FIRST, 174
 - MSB_FIRST, 174
 - PHASE_HIGH, 173
 - PHASE_LOW, 173
 - POLARITY_HIGH, 173
 - POLARITY_LOW, 173
 - Spi_CallbackRegister, 178
 - Spi_DeInit, 175
 - Spi_Init, 174
 - Spi_RegisterRead, 177
 - Spi_RegisterWrite, 176
 - Spi_Transfer, 176
 - SpiBitOrder_t, 173
 - SpiChipSelect_t, 174
 - SpiPhase_t, 173
 - SpiPolarity_t, 173
- Spi_BaudRateSet
 - spi.c, 161
- Spi_CalcDelay
 - spi.c, 163
- Spi_CallbackRegister
 - spi.c, 171
 - spi.h, 178

- `spi_cfg.c`, 180
 - `Spi_ConfigGet`, 180
 - `SpiConfig`, 181
- `Spi_ChipSelectClear`
 - `spi.c`, 168
- `Spi_ChipSelectSet`
 - `spi.c`, 167
- `Spi_ConfigGet`
 - `spi_cfg.c`, 180
- `Spi_DeInit`
 - `spi.c`, 165
 - `spi.h`, 175
- `Spi_Init`
 - `spi.c`, 164
 - `spi.h`, 174
- `Spi_RegisterRead`
 - `spi.c`, 170
 - `spi.h`, 177
- `Spi_RegisterWrite`
 - `spi.c`, 169
 - `spi.h`, 176
- `Spi_Setup`
 - `spi.c`, 166
- `Spi_SSSet`
 - `spi.c`, 162
- `Spi_Transfer`
 - `spi.c`, 165
 - `spi.h`, 176
- `SpiBitOrder_t`
 - `spi.h`, 173
- `SpiChannel`
 - `SpiTransfer_t`, 18
- `SpiChipSelect_t`
 - `spi.h`, 174
- `SpiConfig`
 - `spi_cfg.c`, 181
- `SpiConfig_t`, 16
 - `BaudRate`, 16
 - `Bidirection`, 16
 - `ChannelName`, 16
 - `MasterMode`, 16
 - `SpiEnable`, 16
 - `SSPinMode`, 16
 - `WaitMode`, 16
- `SpiEnable`
 - `SpiConfig_t`, 16
- `SpiPhase_t`
 - `spi.h`, 173
- `SpiPolarity_t`
 - `spi.h`, 173
- `SpiTransfer_t`, 18
 - `ChipSelect`, 18
 - `CsPolarity`, 18
 - `Direction`, 18
 - `NumBytes`, 18
 - `Phase`, 18
 - `Polarity`, 18
 - `SpiChannel`, 18

- TxRxData, 18
- Sq
 - constants.h, 58
- SSPinMode
 - SpiConfig_t, 16
- STOP
 - tmr_cfg.h, 198
- StopBits
 - UartConfig_t, 24
- SUBMCLK
 - uart_cfg.h, 219
- SYS_CLK
 - flash_cfg.h, 98
- SysClkDiv
 - McuConfig_t, 13
- TEMP
 - adc_cfg.h, 50
- TEN_BITS
 - constants.h, 61
- THIRTEEN_BITS
 - constants.h, 61
- THIRTY_BITS
 - constants.h, 61
- THIRTYONE_BITS
 - constants.h, 62
- THIRTYTWO_BITS
 - constants.h, 62
- THREE_BITS
 - constants.h, 61
- THREE_BYTES
 - constants.h, 60
- TimerChannel
 - TmrConfig_t, 20
- TimerEnable
 - TmrConfig_t, 20
- TimerMode
 - TmrConfig_t, 20
- tmr.c, 182
 - Tmr_CallbackRegister, 187
 - Tmr_ClkDivSet, 182
 - Tmr_Disable, 185
 - Tmr_Enable, 184
 - Tmr_Init, 183
 - TMR_PERIOD_DIV, 182
 - Tmr_RegisterRead, 186
 - Tmr_RegisterWrite, 185
- tmr.h, 189
 - Tmr_CallbackRegister, 193
 - Tmr_Disable, 191
 - Tmr_Enable, 190
 - Tmr_Init, 189
 - Tmr_RegisterRead, 192
 - Tmr_RegisterWrite, 191
- Tmr_CallbackRegister
 - tmr.c, 187
 - tmr.h, 193
- tmr_cfg.c, 195
 - Tmr_ConfigGet, 195

- TmrConfig, 196
- tmr_cfg.h, 197
 - EXTERNAL_CLK, 198
 - FLL_PLL, 198
 - INT_CLK, 198
 - MODULE_CLK, 198
 - NUM_TIMERS, 198
 - OSCERCLK, 198
 - STOP, 198
- Tmr_ConfigGet, 198
- TMR_DIV_1, 197
- TMR_DIV_128, 197
- TMR_DIV_16, 197
- TMR_DIV_2, 197
- TMR_DIV_32, 197
- TMR_DIV_4, 197
- TMR_DIV_64, 197
- TMR_DIV_8, 197
- TMR0, 198
- TMR1, 198
- TMR2, 198
- TmrClkSrc_t, 197
- TmrClockMode_t, 198
- TmrCounter_t, 197
- TmrPrescale_t, 197
- TmrRegister_t, 198
- UP_COUNT, 197
- UP_DOWN, 197
- Tmr_ClkDivSet
 - tmr.c, 182
- Tmr_ConfigGet
 - tmr_cfg.c, 195
 - tmr_cfg.h, 198
- Tmr_Disable
 - tmr.c, 185
 - tmr.h, 191
- TMR_DIV_1
 - tmr_cfg.h, 197
- TMR_DIV_128
 - tmr_cfg.h, 197
- TMR_DIV_16
 - tmr_cfg.h, 197
- TMR_DIV_2
 - tmr_cfg.h, 197
- TMR_DIV_32
 - tmr_cfg.h, 197
- TMR_DIV_4
 - tmr_cfg.h, 197
- TMR_DIV_64
 - tmr_cfg.h, 197
- TMR_DIV_8
 - tmr_cfg.h, 197
- Tmr_Enable
 - tmr.c, 184
 - tmr.h, 190
- Tmr_Init
 - tmr.c, 183
 - tmr.h, 189

- TMR_PERIOD_DIV
 - tmr.c, 182
- Tmr_RegisterRead
 - tmr.c, 186
 - tmr.h, 192
- Tmr_RegisterWrite
 - tmr.c, 185
 - tmr.h, 191
- TMR0
 - tmr_cfg.h, 198
- TMR1
 - tmr_cfg.h, 198
- TMR2
 - tmr_cfg.h, 198
- TmrClkSrc_t
 - tmr_cfg.h, 197
- TmrClockMode_t
 - tmr_cfg.h, 198
- TmrConfig
 - tmr_cfg.c, 196
- TmrConfig_t, 20
 - ClkMode, 20
 - ClkPrescaler, 20
 - ClockSource, 20
 - IntEnabled, 20
 - Interval, 21
 - IntPriority, 21
 - TimerChannel, 20
 - TimerEnable, 20
 - TimerMode, 20
- TmrCounter_t
 - tmr_cfg.h, 197
- TmrPrescale_t
 - tmr_cfg.h, 197
- TmrRegister_t
 - tmr_cfg.h, 198
- TPM0
 - adc_cfg.h, 49
- TPM1
 - adc_cfg.h, 49
- TPM2
 - adc_cfg.h, 49
- TriggerSrc
 - AdcConfig_t, 6
- TRUE
 - constants.h, 59
- TWELVE_BITS
 - constants.h, 61
- TWENTY_BITS
 - constants.h, 61
- TWENTYEIGHT_BITS
 - constants.h, 61
- TWENTYFIVE_BITS
 - constants.h, 61
- TWENTYFOUR_BITS
 - constants.h, 61
- TWENTYNINE_BITS
 - constants.h, 61

- TWENTYONE_BITS
 - constants.h, 61
- TWENTYSEVEN_BITS
 - constants.h, 61
- TWENTYSIX_BITS
 - constants.h, 61
- TWENTYTHREE_BITS
 - constants.h, 61
- TWENTYTWO_BITS
 - constants.h, 61
- TWO_BITS
 - constants.h, 61
- TWO_BYTES
 - constants.h, 60
- TWO_PI
 - constants.h, 54
- TX_ONLY
 - uart_cfg.h, 220
- TxRxData
 - SpiTransfer_t, 18
- UART
 - uart_cfg.h, 220
- uart.c, 200
 - CharacterArray, 208
 - Uart_BaudRateSet, 203
 - Uart_CallbackRegister, 207
 - Uart_CharGet, 203
 - Uart_CharPut, 205
 - Uart_Init, 200
 - Uart_IsDataPresent, 204
 - Uart_IsrModeSet, 202
 - Uart_ParitySet, 201
 - Uart_RegisterRead, 207
 - Uart_RegisterWrite, 206
 - UartBaudTable, 208
- uart.h, 209
 - NUM_BAUD_TABLE, 209
 - Uart_BaudRateSet, 210
 - Uart_CallbackRegister, 215
 - Uart_CharGet, 211
 - Uart_CharPut, 212
 - Uart_Init, 209
 - Uart_IsDataPresent, 212
 - Uart_RegisterRead, 214
 - Uart_RegisterWrite, 213
- UART_0
 - uart_cfg.h, 220
- UART_1
 - uart_cfg.h, 220
- UART_2
 - uart_cfg.h, 220
- Uart_BaudRateSet
 - uart.c, 203
 - uart.h, 210
- Uart_CallbackRegister
 - uart.c, 207
 - uart.h, 215
- uart_cfg.c, 217

- Uart_ConfigGet, 217
- UartConfig, 218
- uart_cfg.h, 219
 - ACLOCK, 219
 - BITS_EIGHT, 220
 - BITS_NINE, 220
 - EVEN, 219
 - NUM_UART_CHANNELS, 220
 - ODD, 219
 - RX_ONLY, 220
 - RX_TX, 220
 - SUBMCLK, 219
 - TX_ONLY, 220
 - UART, 220
 - UART_0, 220
 - UART_1, 220
 - UART_2, 220
 - Uart_ConfigGet, 220
 - UART_LSB_FIRST, 219
 - UART_MSB_FIRST, 219
 - UartBitOrder_t, 219
 - UartChannel_t, 220
 - UartClkSrc_t, 219
 - UartComm_t, 220
 - UartInt_t, 220
 - UartMode_t, 220
 - UartParity_t, 219
 - UCLK, 219
- Uart_CharGet
 - uart.c, 203
 - uart.h, 211
- Uart_CharPut
 - uart.c, 205
 - uart.h, 212
- Uart_ConfigGet
 - uart_cfg.c, 217
 - uart_cfg.h, 220
- Uart_Init
 - uart.c, 200
 - uart.h, 209
- Uart_IsDataPresent
 - uart.c, 204
 - uart.h, 212
- Uart_IsrModeSet
 - uart.c, 202
- UART_LSB_FIRST
 - uart_cfg.h, 219
- UART_MSB_FIRST
 - uart_cfg.h, 219
- Uart_ParitySet
 - uart.c, 201
- Uart_RegisterRead
 - uart.c, 207
 - uart.h, 214
- Uart_RegisterWrite
 - uart.c, 206
 - uart.h, 213
- UartBaud_t, 22

- BaudRate, 22
- ClkFreq, 22
- Oversampling, 22
- UCBRFx, 22
- UCBRSx, 22
- UCBRx, 22
- UartBaudTable
 - uart.c, 208
- UartBitOrder_t
 - uart_cfg.h, 219
- UartChannel
 - UartConfig_t, 23
- UartChannel_t
 - uart_cfg.h, 220
- UartClkSrc_t
 - uart_cfg.h, 219
- UartComm_t
 - uart_cfg.h, 220
- UartConfig
 - uart_cfg.c, 218
- UartConfig_t, 23
 - BaudRate, 23
 - BitDirection, 23
 - ClkSrc, 23
 - DataLength, 24
 - Delimiter, 24
 - IntEnable, 24
 - Loopback, 23
 - ParityType, 24
 - StopBits, 24
 - UartChannel, 23
 - UartEnable, 23
 - UartMode, 23
- UartEnable
 - UartConfig_t, 23
- UartInt_t
 - uart_cfg.h, 220
- UartMode
 - UartConfig_t, 23
- UartMode_t
 - uart_cfg.h, 220
- UartParity_t
 - uart_cfg.h, 219
- UCBRFx
 - UartBaud_t, 22
- UCBRSx
 - UartBaud_t, 22
- UCBRx
 - UartBaud_t, 22
- UCLK
 - uart_cfg.h, 219
- UHF_SEL
 - dio_cfg.h, 82
- UP_COUNT
 - tmr_cfg.h, 197
- UP_DOWN
 - tmr_cfg.h, 197
- V_ALT

- adc_cfg.h, 47
- V_REF
 - adc_cfg.h, 47
- V_REFSH
 - adc_cfg.h, 50
- V_REFSL
 - adc_cfg.h, 50
- VERY_HIGH_RANGE
 - mcu_cfg.h, 140
- VRefSrc
 - AdcConfig_t, 6
- WaitMode
 - SpiConfig_t, 16
- wdt.c, 222
 - Wdt_CallbackRegister, 228
 - Wdt_Clear, 225
 - Wdt_Disable, 224
 - Wdt_Enable, 223
 - Wdt_Init, 222
 - Wdt_RegisterRead, 227
 - Wdt_RegisterWrite, 226
 - Wdt_Reset, 225
 - WDT_SOFTWARE_RESET, 222
 - wdtcon, 229
 - wdtsvr, 229
- wdt.h, 230
 - Wdt_CallbackRegister, 236
 - Wdt_Clear, 233
 - Wdt_Disable, 232
 - Wdt_Enable, 231
 - Wdt_Init, 230
 - Wdt_RegisterRead, 235
 - Wdt_RegisterWrite, 234
 - Wdt_Reset, 232
- Wdt_CallbackRegister
 - wdt.c, 228
 - wdt.h, 236
- wdt_cfg.c, 237
 - Wdt_ConfigGet, 237
 - WdtConfig, 238
- wdt_cfg.h, 239
 - BUS, 239
 - INT_1024, 239
 - INT_256, 239
 - INT_262144, 240
 - INT_32, 239
 - INT_65536, 239
 - INT_8192, 239
 - NONE, 239
 - NORMAL, 239
 - ONE_KHZ, 239
 - Wdt_ConfigGet, 240
 - WdtClkSrc_t, 239
 - WdtInterval_t, 239
 - WdtMode_t, 239
 - WINDOWED, 239
- Wdt_Clear
 - wdt.c, 225

- wdt.h, 233
- Wdt_ConfigGet
 - wdt_cfg.c, 237
 - wdt_cfg.h, 240
- Wdt_Disable
 - wdt.c, 224
 - wdt.h, 232
- Wdt_Enable
 - wdt.c, 223
 - wdt.h, 231
- Wdt_Init
 - wdt.c, 222
 - wdt.h, 230
- Wdt_RegisterRead
 - wdt.c, 227
 - wdt.h, 235
- Wdt_RegisterWrite
 - wdt.c, 226
 - wdt.h, 234
- Wdt_Reset
 - wdt.c, 225
 - wdt.h, 232
- WDT_SOFTWARE_RESET
 - wdt.c, 222
- WdtClkSrc_t
 - wdt_cfg.h, 239
- wdtcon
 - wdt.c, 229
- WdtConfig
 - wdt_cfg.c, 238
- WdtConfig_t, 25
 - ClockSource, 25
 - Interval, 25
 - WdtMode, 25
- WdtInterval_t
 - wdt_cfg.h, 239
- WdtMode
 - WdtConfig_t, 25
- WdtMode_t
 - wdt_cfg.h, 239
- wdtsvr
 - wdt.c, 229
- WINDOWED
 - wdt_cfg.h, 239
- X12
 - adc_cfg.h, 47
- X2
 - adc_cfg.h, 47
- X20
 - adc_cfg.h, 47
- X6
 - adc_cfg.h, 47
- ZERO
 - constants.h, 54
- ZERO_BITS
 - constants.h, 61