




MSP430® Peripheral Driver Library for FR5xx and FR6xx Devices

USER'S GUIDE

Copyright

Copyright © 2014 Texas Instruments Incorporated. All rights reserved. MSP430 and 430ware are registered trademarks of Texas Instruments. Other names and brands may be claimed as the property of others.

 Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

Texas Instruments
Post Office Box 655303
Dallas, TX 75265
<http://www.ti.com/msp430>



Revision Information

This is version 1.90.00.65 of this document, last updated on 2014-06-25.

Table of Contents

Copyright	1
Revision Information	1
1 Introduction	3
2 Navigating to driverlib through CCS Resource Explorer	5
3 How to create a new user project that uses Driverlib	13
4 How to include driverlib into your existing project	15
5 12-Bit Analog-to-Digital Converter (ADC12_B)	17
5.1 Introduction	17
5.2 API Functions	18
5.3 Programming Example	45
6 Advanced Encryption Standard (AES256)	47
6.1 Introduction	47
6.2 API Functions	47
6.3 Programming Example	57
7 Comparator (COMP_E)	58
7.1 Introduction	58
7.2 API Functions	58
7.3 Programming Example	71
8 Cyclical Redundancy Check (CRC)	73
8.1 Introduction	73
8.2 API Functions	73
8.3 Programming Example	78
9 Cyclical Redundancy Check (CRC32)	79
9.1 Introduction	79
9.2 API Functions	79
9.3 Programming Example	82
10 Clock System (CS)	83
10.1 Introduction	83
10.2 API Functions	84
10.3 Programming Example	96
11 Direct Memory Access (DMA)	98
11.1 Introduction	98
11.2 API Functions	98
11.3 Programming Example	111
12 EUSCI Universal Asynchronous Receiver/Transmitter (EUSCI_A_UART)	113
12.1 Introduction	113
12.2 API Functions	113
12.3 Programming Example	125
13 EUSCI Synchronous Peripheral Interface (EUSCI_A_SPI)	126
13.1 Introduction	126
13.2 Functions	126
13.3 Programming Example	138
14 EUSCI Synchronous Peripheral Interface (EUSCI_B_SPI)	139
14.1 Introduction	139

14.2 Functions	139
14.3 Programming Example	151
15 EUSCI Inter-Integrated Circuit (EUSCI_B_I2C)	152
15.1 Introduction	152
15.2 API Functions	154
15.3 Programming Example	177
16 FRAM Controller	178
16.1 Introduction	178
16.2 API Functions	178
16.3 Programming Example	184
17 GPIO	185
17.1 Introduction	185
17.2 API Functions	186
17.3 Programming Example	203
18 LCD_C Controller	205
18.1 Introduction	205
18.2 API Functions	205
18.3 Programming Example	222
19 Memory Protection Unit (MPU)	224
19.1 Introduction	224
19.2 API Functions	224
19.3 Programming Example	233
20 32-Bit Hardware Multiplier (MPY32)	234
20.1 Introduction	234
20.2 API Functions	234
20.3 Programming Example	247
21 Power Management Module (PMM)	248
21.1 Introduction	248
21.2 API Functions	248
21.3 Programming Example	254
22 RAM Controller	256
22.1 Introduction	256
22.2 API Functions	256
22.3 Programming Example	257
23 Internal Reference (REF_A)	258
23.1 Introduction	258
23.2 API Functions	258
23.3 Programming Example	267
24 Real-Time Clock (RTC_B)	268
24.1 Introduction	268
24.2 API Functions	268
24.3 Programming Example	280
25 Real-Time Clock (RTC_C)	282
25.1 Introduction	282
25.2 API Functions	282
25.3 Programming Example	292
26 SFR Module	294
26.1 Introduction	294

26.2	API Functions	294
26.3	Programming Example	299
27	SYS Module	300
27.1	Introduction	300
27.2	API Functions	300
27.3	Programming Example	310
28	16-Bit Timer_A (TIMER_A)	311
28.1	Introduction	311
28.2	API Functions	312
28.3	Programming Example	337
29	16-Bit Timer_B (TIMER_B)	339
29.1	Introduction	339
29.2	API Functions	340
29.3	Programming Example	367
30	Tag Length Value	369
30.1	Introduction	369
30.2	API Functions	369
30.3	Programming Example	374
31	WatchDog Timer (WDT_A)	375
31.1	Introduction	375
31.2	API Functions	375
31.3	Programming Example	379
IMPORTANT NOTICE		380

1 Introduction

The Texas Instruments® MSP430® Peripheral Driver Library is a set of drivers for accessing the peripherals found on the MSP430 FR5xx/FR6xx family of microcontrollers. While they are not drivers in the pure operating system sense (that is, they do not have a common interface and do not connect into a global device driver infrastructure), they do provide a mechanism that makes it easy to use the device's peripherals.

The capabilities and organization of the drivers are governed by the following design goals:

- They are written entirely in C except where absolutely not possible.
- They demonstrate how to use the peripheral in its common mode of operation.
- They are easy to understand.
- They are reasonably efficient in terms of memory and processor usage.
- They are as self-contained as possible.
- Where possible, computations that can be performed at compile time are done there instead of at run time.
- They can be built with more than one tool chain.

Some consequences of these design goals are:

- The drivers are not necessarily as efficient as they could be (from a code size and/or execution speed point of view). While the most efficient piece of code for operating a peripheral would be written in assembly and custom tailored to the specific requirements of the application, further size optimizations of the drivers would make them more difficult to understand.
- The drivers do not support the full capabilities of the hardware. Some of the peripherals provide complex capabilities which cannot be utilized by the drivers in this library, though the existing code can be used as a reference upon which to add support for the additional capabilities.
- The APIs have a means of removing all error checking code. Because the error checking is usually only useful during initial program development, it can be removed to improve code size and speed.

For many applications, the drivers can be used as is. But in some cases, the drivers will have to be enhanced or rewritten in order to meet the functionality, memory, or processing requirements of the application. If so, the existing driver can be used as a reference on how to operate the peripheral.

Each MSP430ware driverlib API takes in the base address of the corresponding peripheral as the first parameter. This base address is obtained from the msp430 device specific header files (or from the device datasheet). The example code for the various peripherals show how base address is used. When using CCS, the eclipse shortcut "Ctrl + Space" helps. Type `__MSP430` and "Ctrl + Space", and the list of base addresses from the included device specific header files is listed.

The following tool chains are supported:

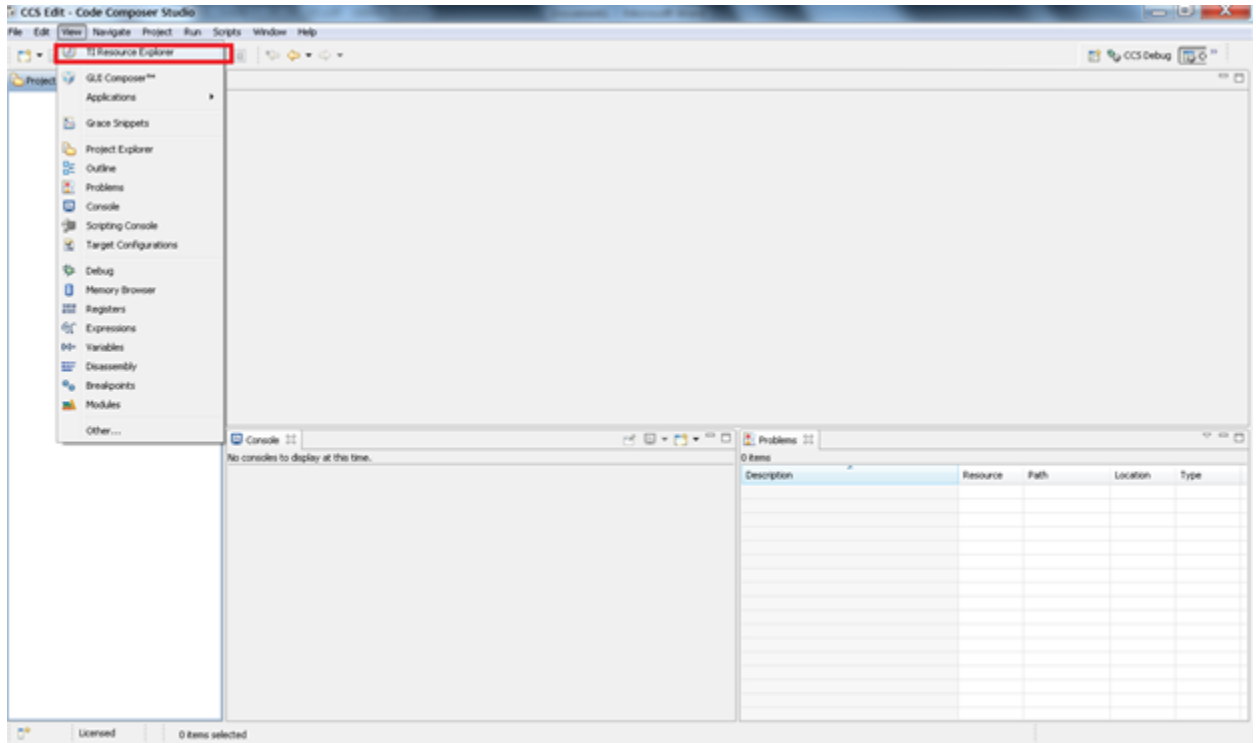
- IAR Embedded Workbench®
- Texas Instruments Code Composer Studio™

Using assert statements to debug

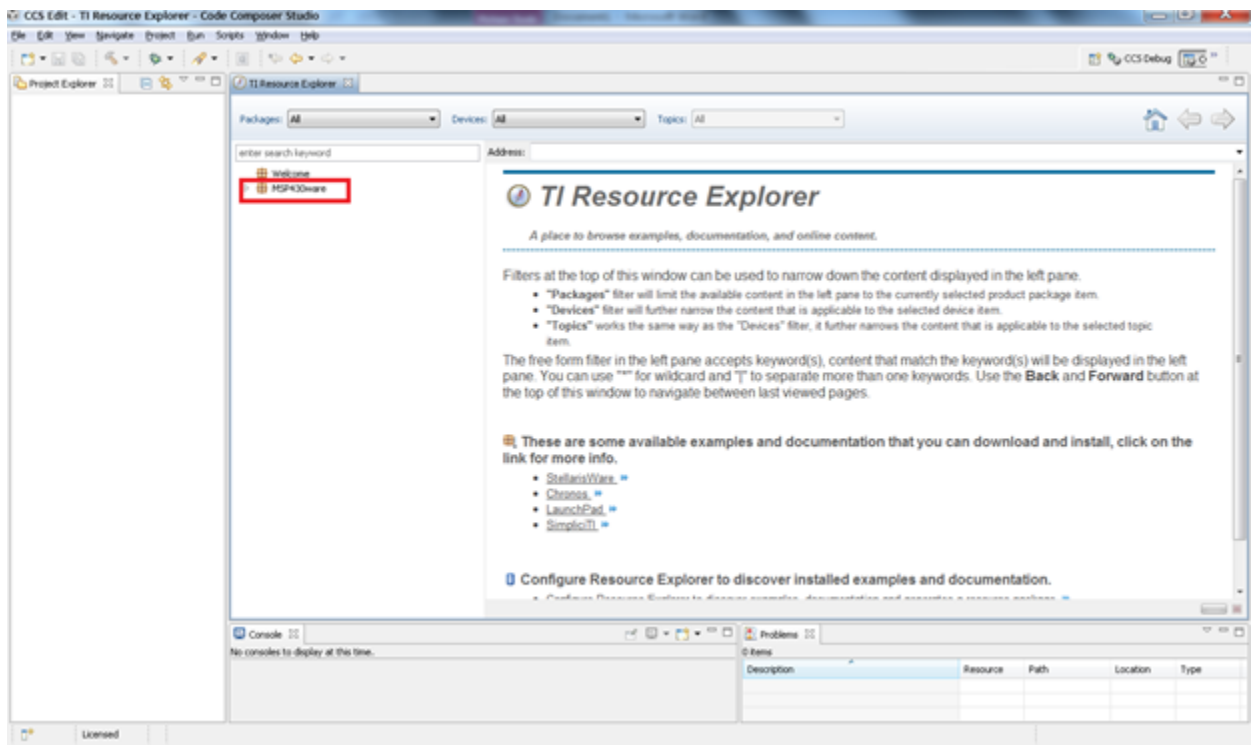
Assert statements are disabled by default. To enable the assert statement edit the hw_regaccess.h file in the inc folder. Comment out the statement `define NDEBUG -> //define NDEBUG` Asserts in CCS work only if the project is optimized for size.

2 Navigating to driverlib through CCS Resource Explorer

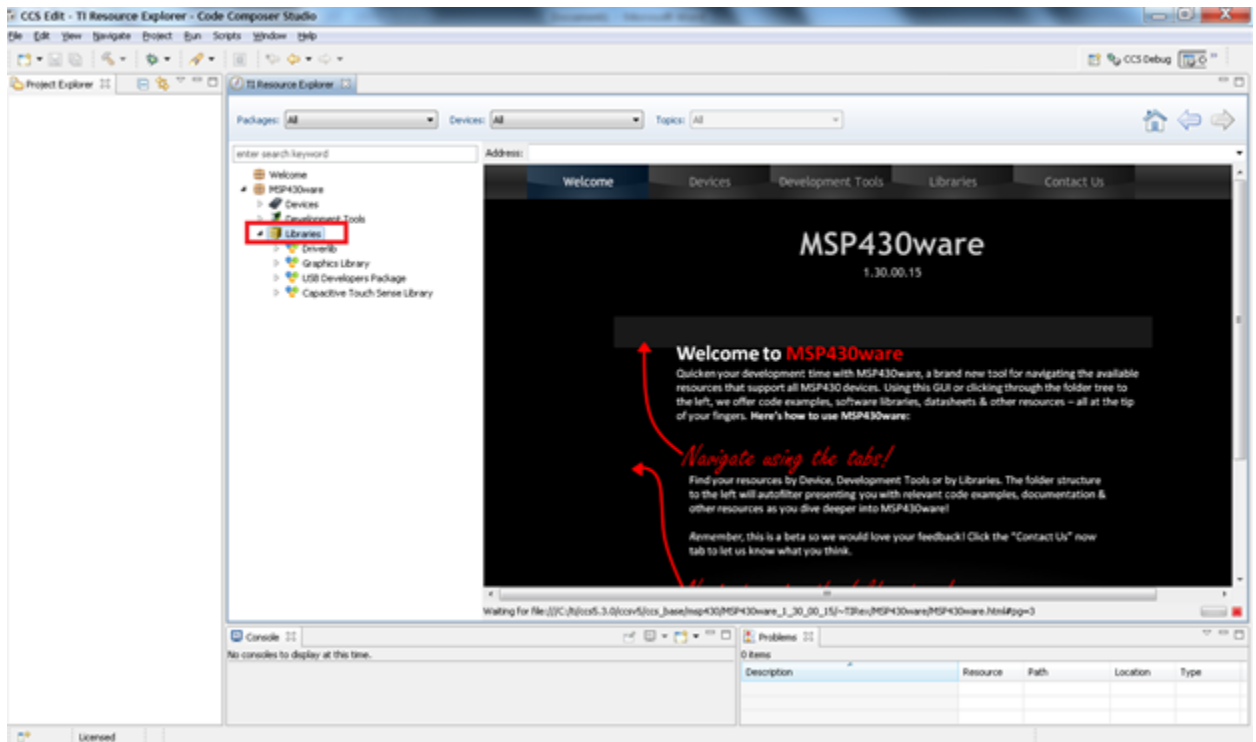
In CCS, click View->TI Resource Explorer

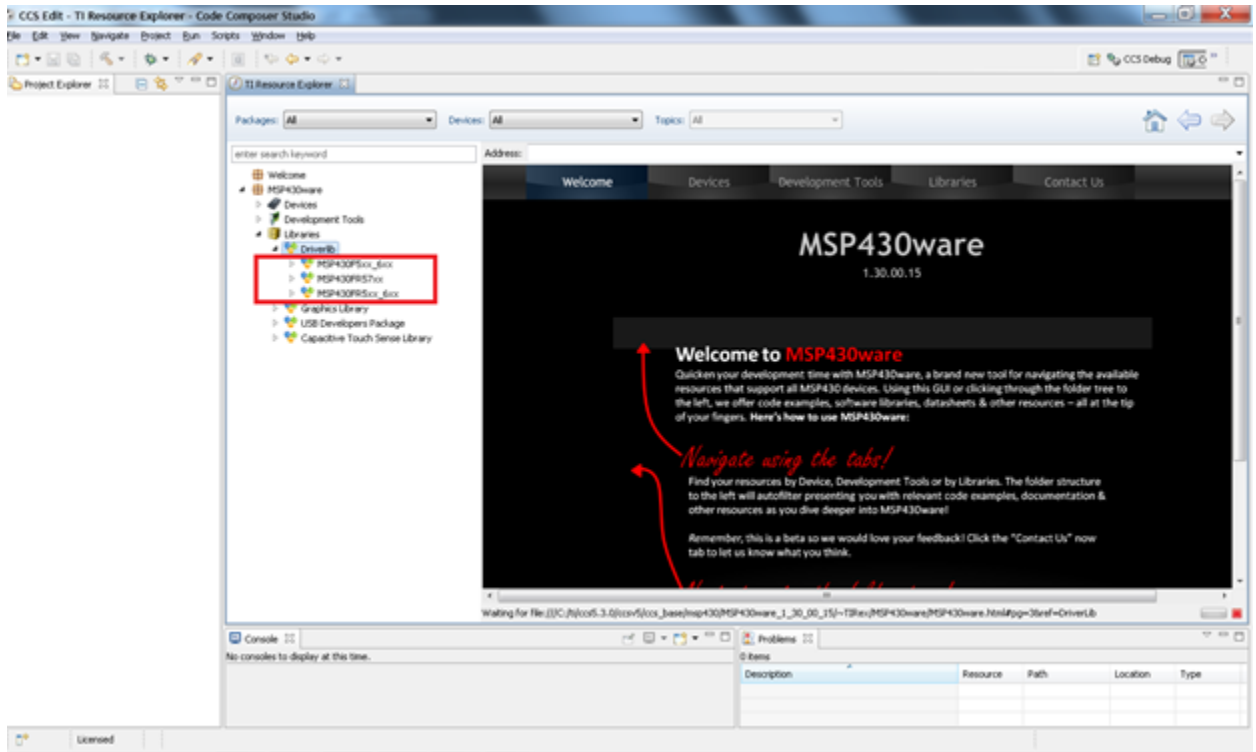


In Resource Explorer View, click on MSP430ware

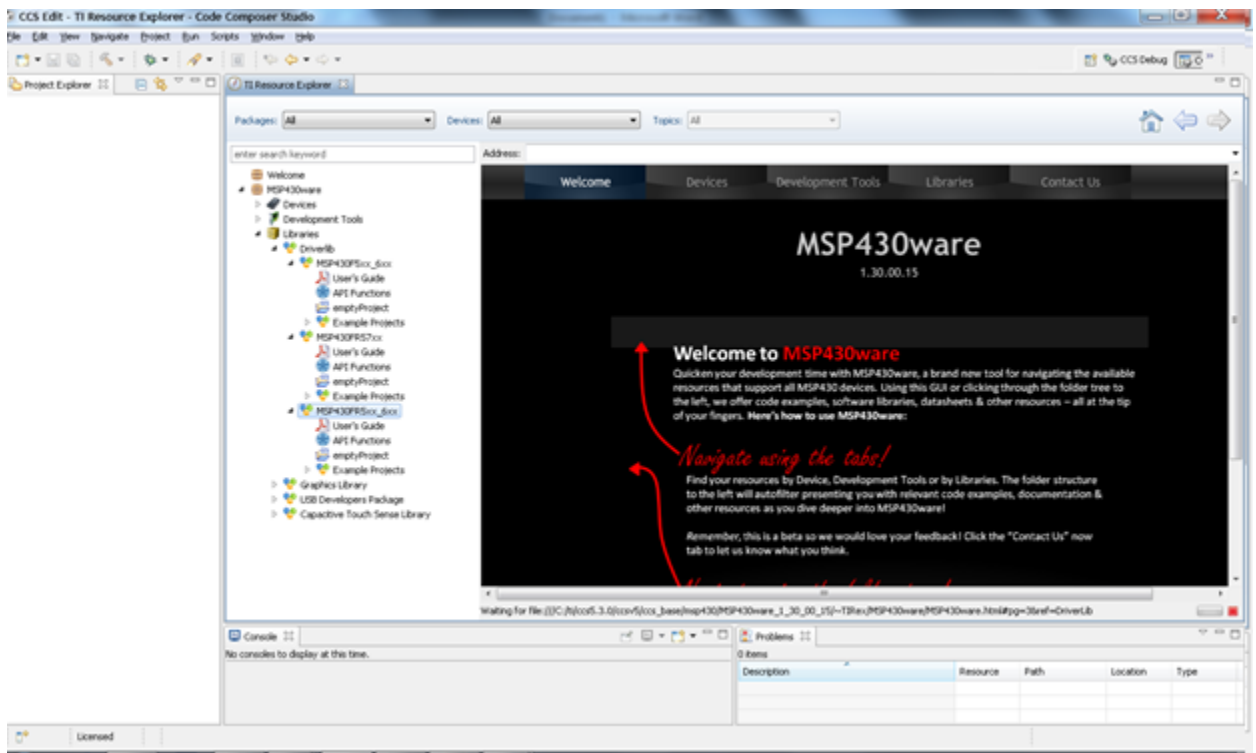


Clicking MSP430ware takes you to the introductory page. The version of the latest MSP430ware installed is available in this page. In this screenshot the version is 1.30.00.15 The various software, collateral, code examples, datasheets and user guides can be navigated by clicking the different topics under MSP430ware. To proceed to driverlib, click on Libraries->Driverlib as shown in the next two screenshots.





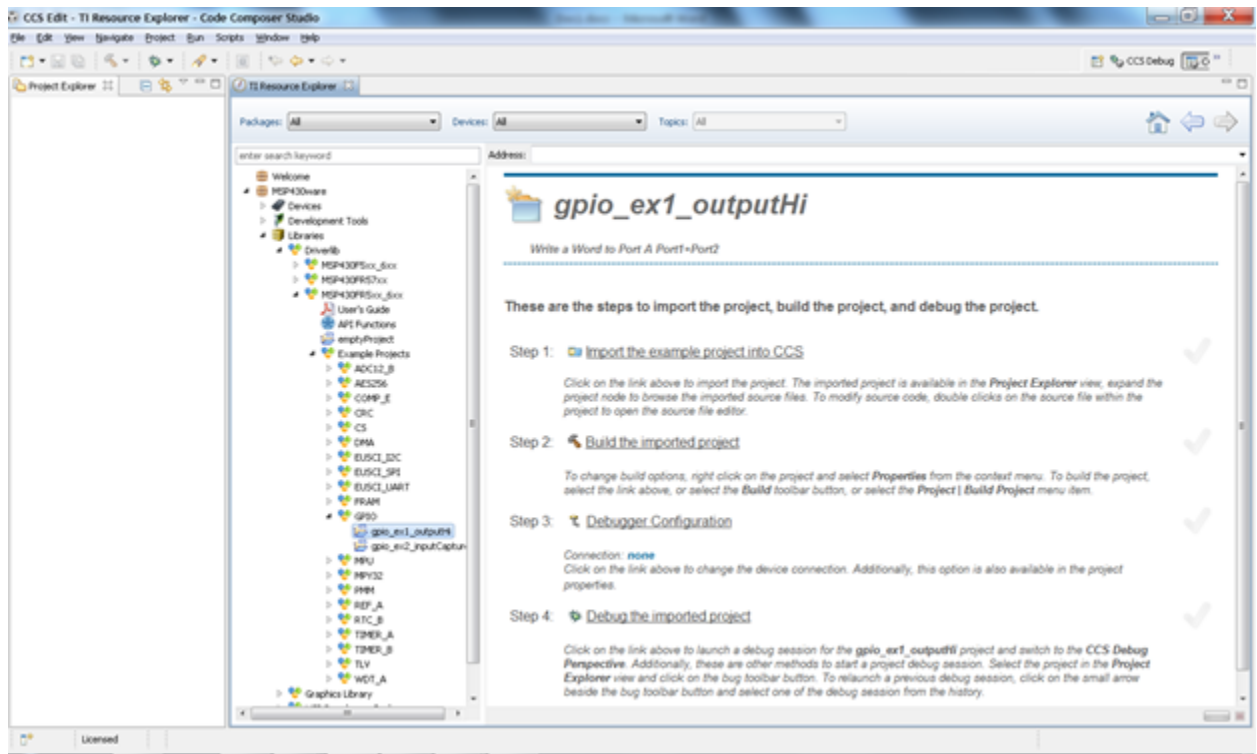
Driverlib is designed per Family. If a common device family user's guide exists for a group of devices, these devices belong to the same 'family'. Currently driverlib is available for the following family of devices. MSP430F5xx_6xx MSP430FR57xx MSP430FR5xx_6xx MSP430i2xx



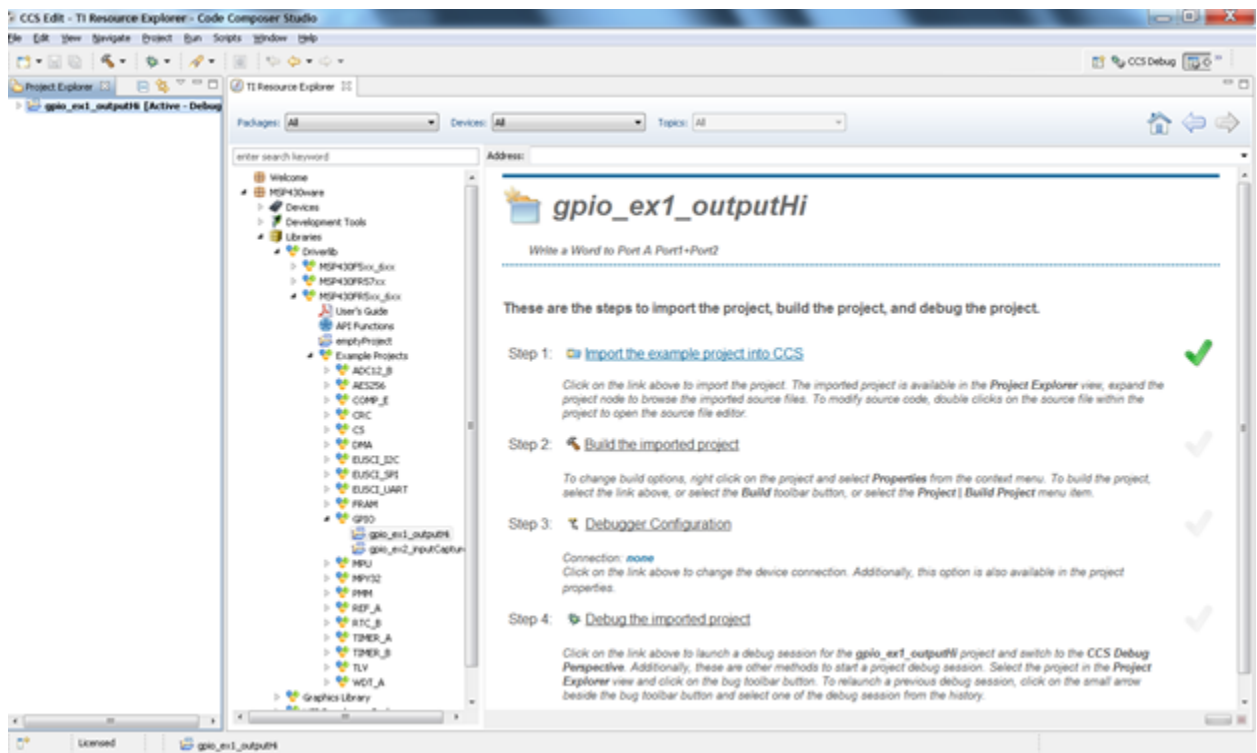
Click on the MSP430FR5xx_6xx to navigate to the driverlib based example code for that family.



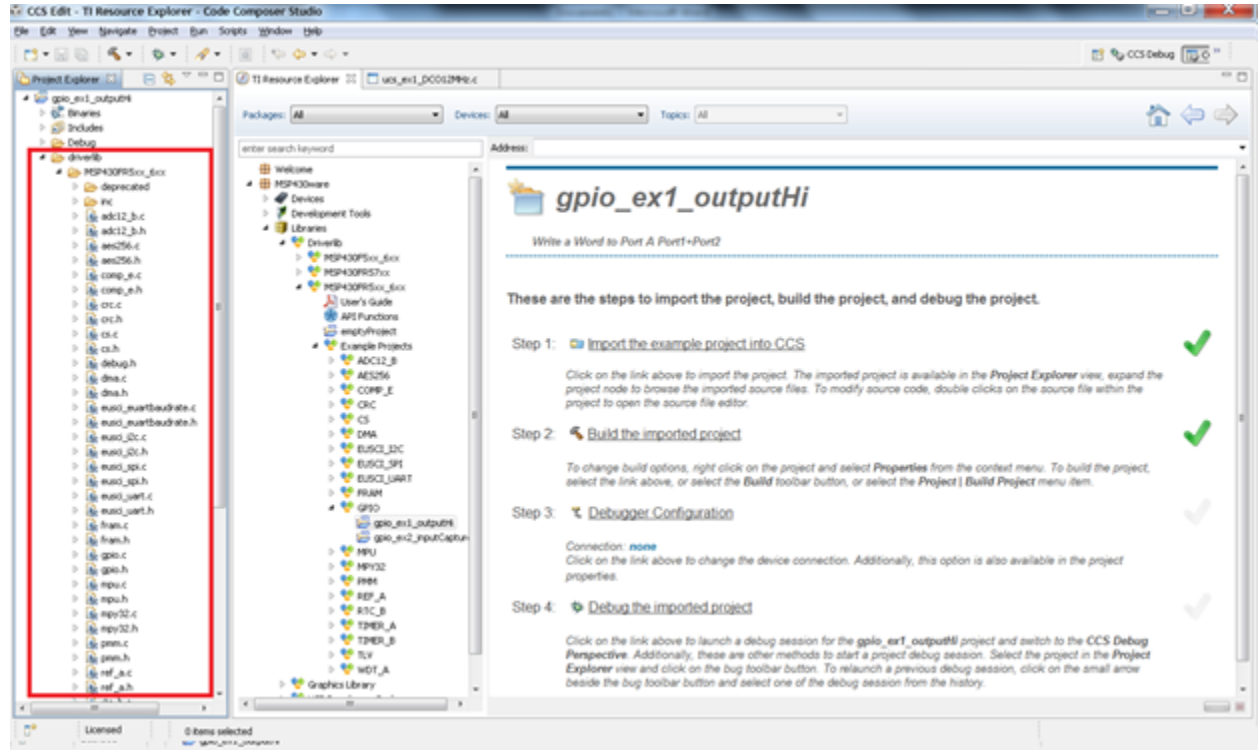
The various peripherals are listed in alphabetical order. The names of peripherals are as in device family user's guide. Clicking on a peripheral name lists the driverlib example code for that peripheral. The screenshot below shows an example when the user clicks on GPIO peripheral.



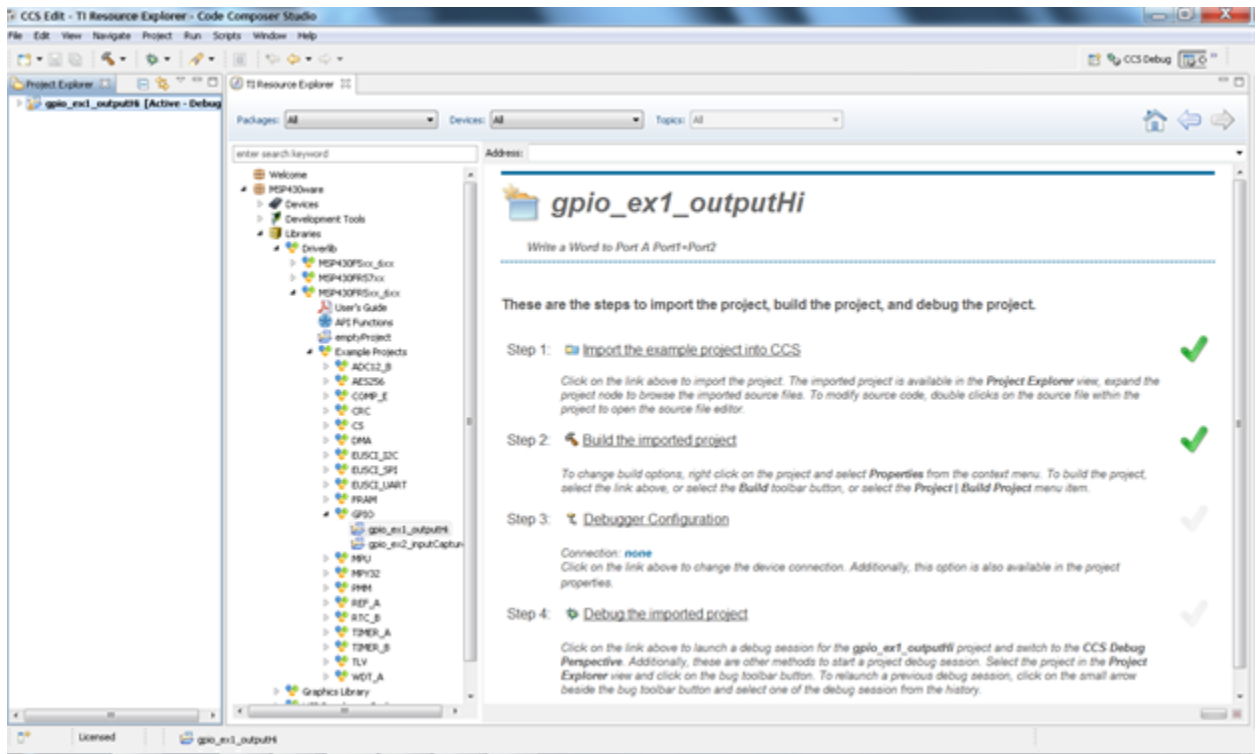
Now click on the specific example you are interested in. On the right side there are options to Import/Build/Download and Debug. Import the project by clicking on the "Import the example project into CCS"



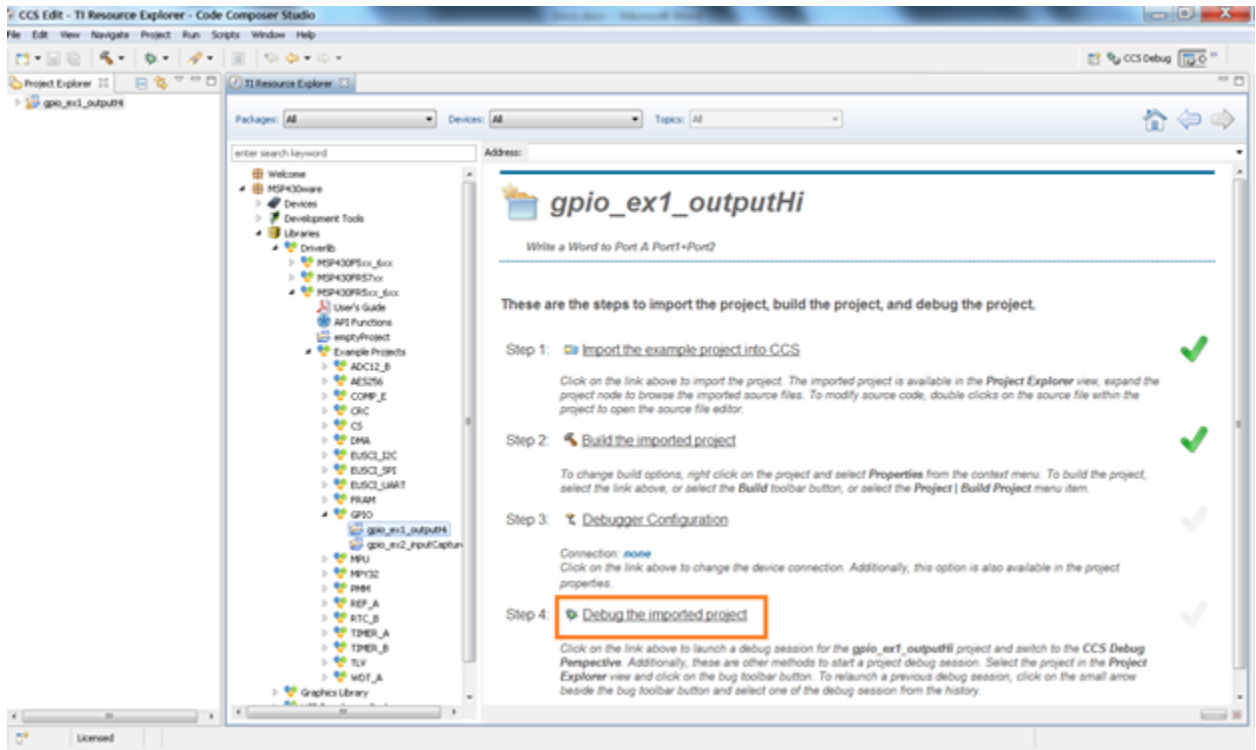
The imported project can be viewed on the left in the Project Explorer. All required driverlib source and header files are included inside the driverlib folder. All driverlib source and header files are linked to the example projects. So if the user modifies any of these source or header files, the original copy of the installed MSP430ware driverlib source and header files get modified.



Now click on Build the imported project on the right to build the example project.

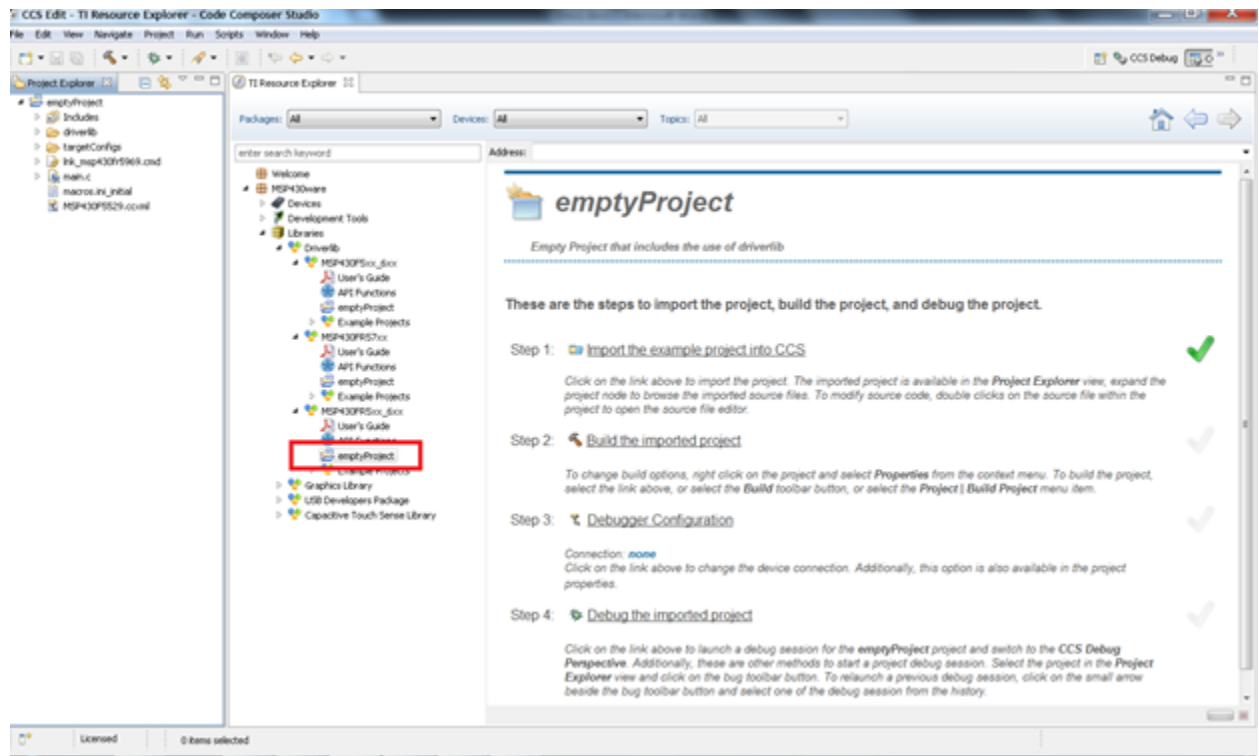


Now click on Build the imported project on the right to build the example project.



The COM port to download to can be changed using the Debugger Configuration option on the right if required.

To get started on a new project we recommend getting started on an empty project we provide. This project has all the driverlib source files, header files, project paths are set by default.

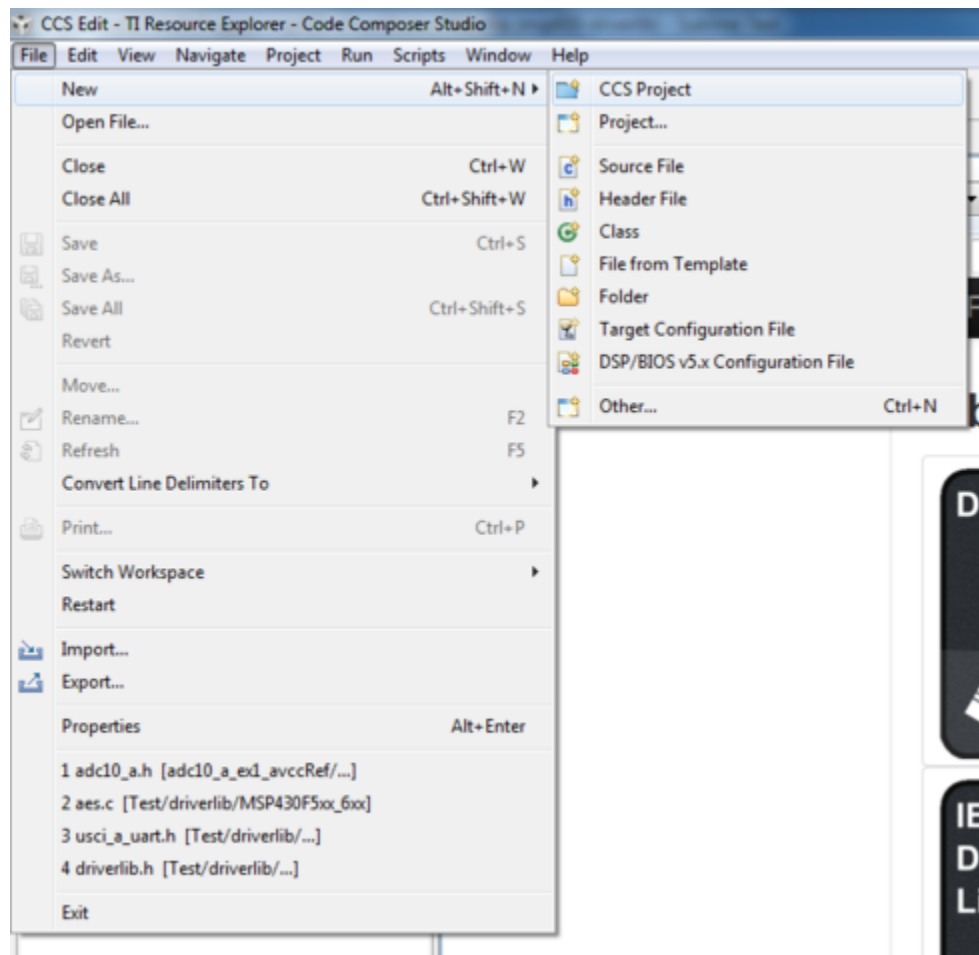


The main.c included with the empty project can be modified to include user code.

3 How to create a new user project that uses Driverlib

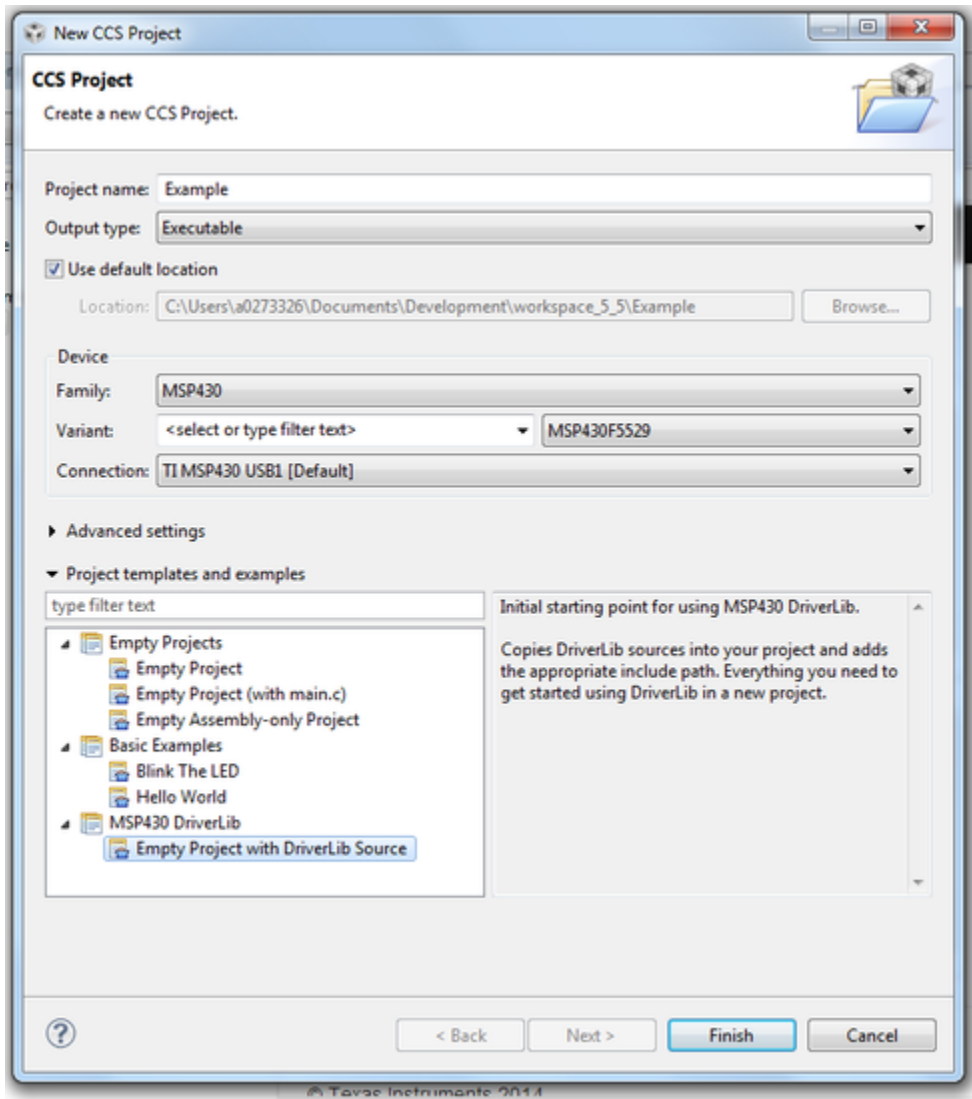
To get started on a new project we recommend using the new project wizard. For driver library to work with the new project wizard CCS must have discovered the driver library RTSC product. For more information refer to the installation steps of the release notes. The new project wizard adds the needed driver library source files and adds the driver library include path.

To open the new project wizard go to File -> New -> CCS Project as seen in the screenshot below.



Once the new project wizard has been opened name your project and choose the device you would like to create a Driver Library project for. The device must be supported by driver library.

Then under "Project templates and examples" choose "Empty Project with DriverLib Source" as seen below.

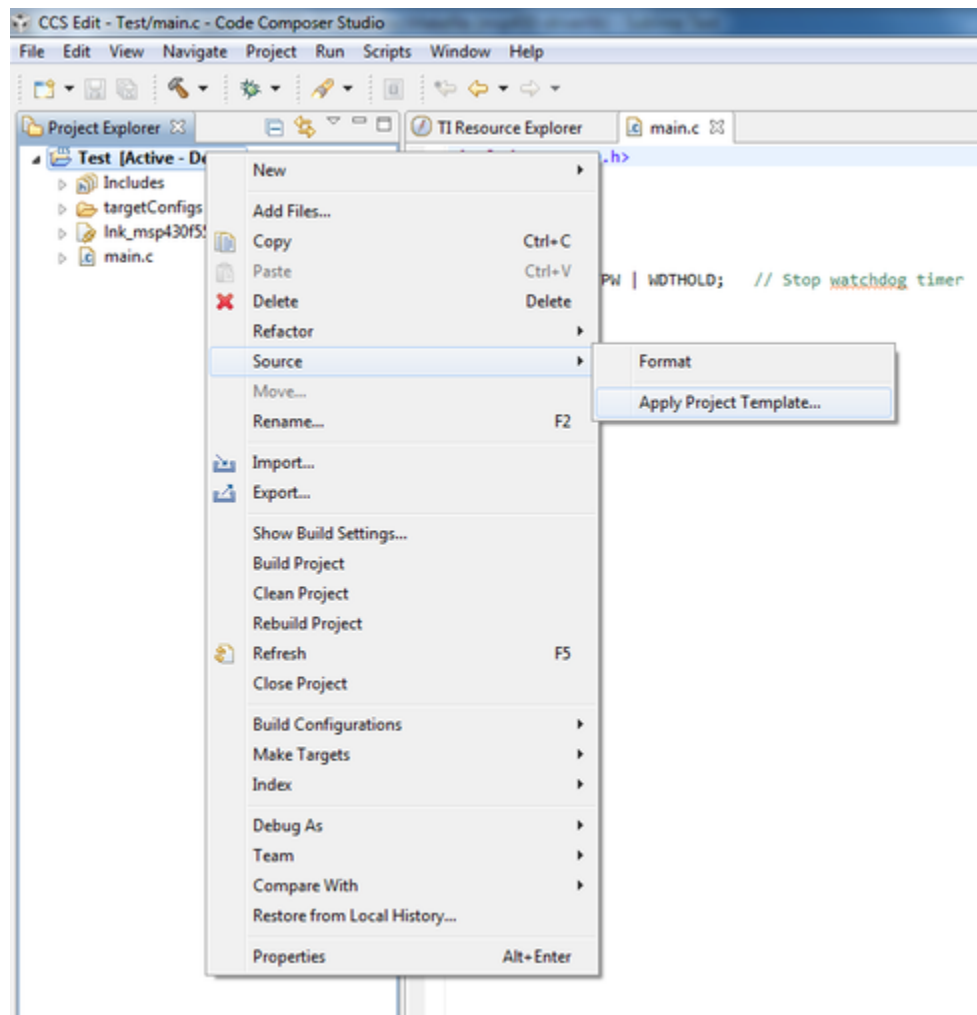


Finally click "Finish" and begin developing with your Driver Library enabled project.

4 How to include driverlib into your existing project

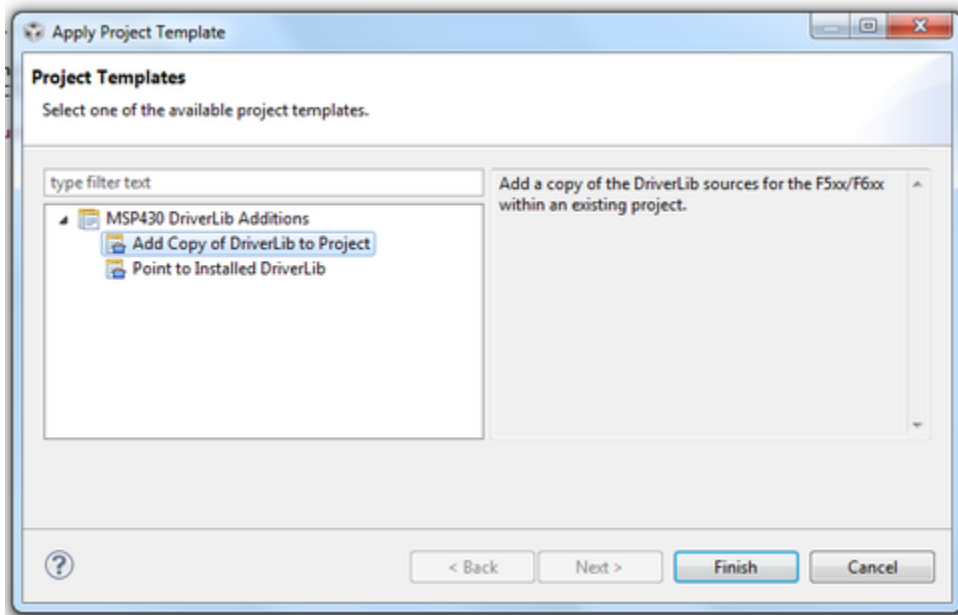
To add driver library to an existing project we recommend using CCS project templates. For driver library to work with project templates CCS must have discovered the driver library RTSC product. For more information refer to the installation steps of the release notes. CCS project templates adds the needed driver library source files and adds the driver library include path.

To apply a project template right click on an existing project then go to Source -> Apply Project Template as seen in the screenshot below.



In the "Apply Project Template" dialog box under "MSP430 DriverLib Additions" choose either "Add Local Copy" or "Point to Installed DriverLib" as seen in the screenshot below. Most users will want to add a local copy which copies the DriverLib source into the project and sets the compiler settings needed.

Pointing to an installed DriverLib is for advanced users who are including a static library in their project and want to add the DriverLib header files to their include path.



Click "Finish" and start developing with driver library in your project.

5 12-Bit Analog-to-Digital Converter (ADC12_B)

Introduction	17
API Functions	18
Programming Example	45

5.1 Introduction

The 12-Bit Analog-to-Digital (ADC12_B) API provides a set of functions for using the MSP430Ware ADC12_B modules. Functions are provided to initialize the ADC12_B modules, setup signal sources and reference voltages for each memory buffer, and manage interrupts for the ADC12_B modules.

The ADC12_B module provides the ability to convert analog signals into a digital value in respect to given reference voltages. The module implements a 12-bit SAR core, sample select control, and up to 32 independent conversion-and-control buffers. The conversion-and-control buffer allows up to 32 independent analog-to-digital converter (ADC) samples to be converted and stored without any CPU intervention. The ADC12_B can also generate digital values from 0 to V_{CC} with an 8-, 10- or 12-bit resolution and it can operate in 2 different sampling modes, and 4 different conversion modes. The sampling modes are extended sampling and pulse sampling, in extended sampling the sample/hold signal must stay high for the duration of sampling, while in pulse mode a sampling timer is setup to start on a rising edge of the sample/hold signal and sample for a specified amount of clock cycles. The 4 conversion modes are single-channel single conversion, sequence of channels single-conversion, repeated single channel conversions, and repeated sequence of channels conversions.

The ADC12_B module can generate multiple interrupts. An interrupt can be asserted for each memory buffer when a conversion is complete, or when a conversion is about to overwrite the converted data in any of the memory buffers before it has been read out, and/or when a conversion is about to start before the last conversion is complete.

ADC12_B features include:

- 200 ksp/s maximum conversion rate at maximum resolution of 12-bits
- Monotonic 12-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers.
- Conversion initiation by software or timers.
- Software-selectable on-chip reference voltage generation (1.2 V, 2.0 V, or 2.5 V) with option to make available externally
- Software-selectable internal or external reference
- Up to 32 individually configurable external input channels, single-ended or differential input selection available
- Internal conversion channels for internal temperature sensor and 2/3 V_{AVCC} and four more internal channels available on select devices see device data sheet for availability as well as function
- Independent channel-selectable reference sources for both positive and negative references

- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence (autoscan), and repeat-sequence (repeated autoscan) conversion modes
- Interrupt vector register for fast decoding of 38 ADC interrupts
- 32 conversion-result storage registers
- Window comparator for low power monitoring of input signals of conversion-result registers

This driver is contained in `adc12_b.c`, with `adc12_b.h` containing the API definitions for use by applications.

T

The following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	1146
TI Compiler 4.2.1	Size	558
TI Compiler 4.2.1	Speed	554
IAR 5.51.6	None	832
IAR 5.51.6	Size	640
IAR 5.51.6	Speed	680
MSPGCC 4.8.0	None	1806
MSPGCC 4.8.0	Size	606
MSPGCC 4.8.0	Speed	618

5.2 API Functions

Functions

- void [ADC12_B_clearInterrupt](#) (uint16_t baseAddress, uint8_t interruptRegisterChoice, uint16_t memoryInterruptFlagMask)
- void [ADC12_B_configureMemory](#) (uint16_t baseAddress, ADC12_B_configureMemoryParam *param)
- void [ADC12_B_disable](#) (uint16_t baseAddress)
- void [ADC12_B_disableConversions](#) (uint16_t baseAddress, bool preempt)
- void [ADC12_B_disableInterrupt](#) (uint16_t baseAddress, uint16_t interruptMask0, uint16_t interruptMask1, uint16_t interruptMask2)
- void [ADC12_B_disableSamplingTimer](#) (uint16_t baseAddress)
- void [ADC12_B_enable](#) (uint16_t baseAddress)
- void [ADC12_B_enableInterrupt](#) (uint16_t baseAddress, uint16_t interruptMask0, uint16_t interruptMask1, uint16_t interruptMask2)
- uint16_t [ADC12_B_getInterruptStatus](#) (uint16_t baseAddress, uint8_t interruptRegisterChoice, uint16_t memoryInterruptFlagMask)

- uint32_t [ADC12_B_getMemoryAddressForDMA](#) (uint16_t baseAddress, uint8_t memoryIndex)
- uint16_t [ADC12_B_getResults](#) (uint16_t baseAddress, uint8_t memoryBufferIndex)
- bool [ADC12_B_init](#) (uint16_t baseAddress, uint16_t sampleHoldSignalSourceSelect, uint8_t clockSourceSelect, uint16_t clockSourceDivider, uint16_t clockSourcePredivider, uint16_t internalChannelMap)
- bool [ADC12_B_initialize](#) (uint16_t baseAddress, ADC12_B_initializeParam *param)
- uint8_t [ADC12_B_isBusy](#) (uint16_t baseAddress)
- void [ADC12_B_memoryConfigure](#) (uint16_t baseAddress, uint8_t memoryBufferControlIndex, uint8_t inputSourceSelect, uint16_t refVoltageSourceSelect, uint16_t endOfSequence, uint16_t windowComparatorSelect, uint16_t differentialModeSelect)
- void [ADC12_B_setAdcPowerMode](#) (uint16_t baseAddress, uint8_t powerMode)
- void [ADC12_B_setDataReadBackFormat](#) (uint16_t baseAddress, uint8_t readBackFormat)
- void [ADC12_B_setResolution](#) (uint16_t baseAddress, uint8_t resolutionSelect)
- void [ADC12_B_setSampleHoldSignalInversion](#) (uint16_t baseAddress, uint16_t invertedSignal)
- void [ADC12_B_setupSamplingTimer](#) (uint16_t baseAddress, uint16_t clockCycleHoldCountLowMem, uint16_t clockCycleHoldCountHighMem, uint16_t multipleSamplesEnabled)
- void [ADC12_B_setWindowCompAdvanced](#) (uint16_t baseAddress, uint16_t highThreshold, uint16_t lowThreshold)
- void [ADC12_B_startConversion](#) (uint16_t baseAddress, uint16_t startingMemoryBufferIndex, uint8_t conversionSequenceModeSelect)

5.2.1 Detailed Description

The ADC12_B API is broken into three groups of functions: those that deal with initialization and conversions, those that handle interrupts, and those that handle auxiliary features of the ADC12_B.

The ADC12_B initialization and conversion functions are

- [ADC12_B_init](#)
- [ADC12_B_memoryConfigure](#)
- [ADC12_B_setWindowCompAdvanced](#)
- [ADC12_B_setupSamplingTimer](#)
- [ADC12_B_disableSamplingTimer](#)
- [ADC12_B_startConversion](#)
- [ADC12_B_disableConversions](#)
- [ADC12_B_getResults](#)
- [ADC12_B_isBusy](#)

The ADC12_B interrupts are handled by

- [ADC12_B_enableInterrupt](#)
- [ADC12_B_disableInterrupt](#)

- ADC12_B_clearInterrupt
- ADC12_B_getInterruptStatus

Auxiliary features of the ADC12_B are handled by

- ADC12_B_setResolution
- ADC12_B_setSampleHoldSignalInversion
- ADC12_B_setDataReadBackFormat
- ADC12_B_enableReferenceBurst
- ADC12_B_disableReferenceBurst
- ADC12_B_setAdcPowerMode
- ADC12_B_getMemoryAddressForDMA
- ADC12_B_enable
- ADC12_B_disable

5.2.2 Function Documentation

5.2.2.1 void ADC12_B_clearInterrupt (uint16_t *baseAddress*, uint8_t *interruptRegisterChoice*, uint16_t *memoryInterruptFlagMask*)

Clears ADC12B selected interrupt flags.

Modified registers are ADC12IFG .

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	40
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	22
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	16
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:

baseAddress is the base address of the ADC12B module.

interruptRegisterChoice is either 0, 1, or 2, to choose the correct interrupt register to update

memoryInterruptFlagMask is the bit mask of the memory buffer and overflow interrupt flags to be cleared. Valid values are:

- **ADC12_B_IFG0** - interruptRegisterChoice = 0
- **ADC12_B_IFG1**
- **ADC12_B_IFG2**
- **ADC12_B_IFG3**

- ADC12_B_IFG4
- ADC12_B_IFG5
- ADC12_B_IFG6
- ADC12_B_IFG7
- ADC12_B_IFG8
- ADC12_B_IFG9
- ADC12_B_IFG10
- ADC12_B_IFG11
- ADC12_B_IFG12
- ADC12_B_IFG13
- ADC12_B_IFG14
- ADC12_B_IFG15
- ADC12_B_IFG16 - interruptRegisterChoice = 1
- ADC12_B_IFG17
- ADC12_B_IFG18
- ADC12_B_IFG19
- ADC12_B_IFG20
- ADC12_B_IFG21
- ADC12_B_IFG22
- ADC12_B_IFG23
- ADC12_B_IFG24
- ADC12_B_IFG25
- ADC12_B_IFG26
- ADC12_B_IFG27
- ADC12_B_IFG28
- ADC12_B_IFG29
- ADC12_B_IFG30
- ADC12_B_IFG31
- ADC12_B_INIFG - interruptRegisterChoice = 2
- ADC12_B_LOIFG
- ADC12_B_HIIFG
- ADC12_B_OVIFG
- ADC12_B_TOVIFG
- ADC12_B_RDYIFG - The selected ADC12B interrupt flags are cleared, so that it no longer asserts. The memory buffer interrupt flags are only cleared when the memory buffer is accessed. Note that the overflow interrupts do not have an interrupt flag to clear; they must be accessed directly from the interrupt vector.

Returns:

None

5.2.2.2 void ADC12_B_configureMemory (uint16_t *baseAddress*,
ADC12_B_configureMemoryParam * *param*)

Configures the controls of the selected memory buffer.

Maps an input signal conversion into the selected memory buffer, as well as the positive and negative reference voltages for each conversion being stored into this memory buffer. If the internal reference is used for the positive reference voltage, the internal REF module must be used to control the voltage level. Note that if a conversion has been started with the startConversion() function, then a call to disableConversions() is required before this function may be called.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	102
TI Compiler 4.2.1	Size	50
TI Compiler 4.2.1	Speed	50
IAR 5.51.6	None	76
IAR 5.51.6	Size	50
IAR 5.51.6	Speed	50
MSPGCC 4.8.0	None	178
MSPGCC 4.8.0	Size	52
MSPGCC 4.8.0	Speed	52

Parameters:

baseAddress is the base address of the ADC12B module.

param is the pointer to struct for ADC12B memory configuration.

Returns:

None

5.2.2.3 void ADC12_B_disable (uint16_t baseAddress)

Disables the ADC12B block.

This will disable operation of the ADC12B block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	16
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the ADC12B module.

Modified bits are **ADC12ON** of **ADC12CTL0** register.

Returns:

None

5.2.2.4 void ADC12_B_disableConversions (uint16_t *baseAddress*, bool *preempt*)

Disables the ADC from converting any more signals.

Disables the ADC from converting any more signals. If there is a conversion in progress, this function can stop it immediately if the preempt parameter is set as ADC12_B_PREEMPTCONVERSION, by changing the conversion mode to single-channel, single-conversion and disabling conversions. If the conversion mode is set as single-channel, single-conversion and this function is called without preemption, then the ADC core conversion status is polled until the conversion is complete before disabling conversions to prevent unpredictable data. If the [ADC12_B_startConversion\(\)](#) has been called, then this function has to be called to re-initialize the ADC, reconfigure a memory buffer control, enable/disable the sampling pulse mode, or change the internal reference voltage.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	62
TI Compiler 4.2.1	Size	28
TI Compiler 4.2.1	Speed	28
IAR 5.51.6	None	42
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	34

Parameters:

baseAddress is the base address of the ADC12B module.

preempt specifies if the current conversion should be preemptively stopped before the end of the conversion. Valid values are:

- **ADC12_B_COMPLETECONVERSION** - Allows the ADC12B to end the current conversion before disabling conversions.
- **ADC12_B_PREEMPTCONVERSION** - Stops the ADC12B immediately, with unpredictable results of the current conversion.

Modified bits of **ADC12CTL1** register and bits of **ADC12CTL0** register.

Returns:

None

5.2.2.5 void ADC12_B_disableInterrupt (uint16_t *baseAddress*, uint16_t *interruptMask0*, uint16_t *interruptMask1*, uint16_t *interruptMask2*)

Disables selected ADC12B interrupt sources.

Disables the indicated ADC12B interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	58
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	36
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	110
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the ADC12B module.

interruptMask0 is the bit mask of the memory buffer and overflow interrupt sources to be disabled. If the desired interrupt is not available in the selection for *interruptMask0*, then simply pass in a '0' for this value. Valid values are:

- ADC12_B_IE0
- ADC12_B_IE1
- ADC12_B_IE2
- ADC12_B_IE3
- ADC12_B_IE4
- ADC12_B_IE5
- ADC12_B_IE6
- ADC12_B_IE7
- ADC12_B_IE8
- ADC12_B_IE9
- ADC12_B_IE10
- ADC12_B_IE11
- ADC12_B_IE12
- ADC12_B_IE13
- ADC12_B_IE14
- ADC12_B_IE15

interruptMask1 is the bit mask of the memory buffer and overflow interrupt sources to be disabled. If the desired interrupt is not available in the selection for *interruptMask1*, then simply pass in a '0' for this value. Valid values are:

- ADC12_B_IE16
- ADC12_B_IE17
- ADC12_B_IE18
- ADC12_B_IE19
- ADC12_B_IE20
- ADC12_B_IE21
- ADC12_B_IE22

- **ADC12_B_IE23**
- **ADC12_B_IE24**
- **ADC12_B_IE25**
- **ADC12_B_IE26**
- **ADC12_B_IE27**
- **ADC12_B_IE28**
- **ADC12_B_IE29**
- **ADC12_B_IE30**
- **ADC12_B_IE31**

interruptMask2 is the bit mask of the memory buffer and overflow interrupt sources to be disabled. If the desired interrupt is not available in the selection for **interruptMask2**, then simply pass in a '0' for this value. Valid values are:

- **ADC12_B_INIE** - Interrupt enable for a conversion in the result register is either greater than the ADC12LO or lower than the ADC12HI threshold. GIE bit must be set to enable the interrupt.
- **ADC12_B_LOIE** - Interrupt enable for the falling short of the lower limit interrupt of the window comparator for the result register. GIE bit must be set to enable the interrupt.
- **ADC12_B_HIIE** - Interrupt enable for the exceeding the upper limit of the window comparator for the result register. GIE bit must be set to enable the interrupt.
- **ADC12_B_OVIE** - Interrupt enable for a conversion that is about to save to a memory buffer that has not been read out yet. GIE bit must be set to enable the interrupt.
- **ADC12_B_TOVIE** - enable for a conversion that is about to start before the previous conversion has been completed. GIE bit must be set to enable the interrupt.
- **ADC12_B_RDYIE** - enable for the local buffered reference ready signal. GIE bit must be set to enable the interrupt.

Modified bits of **ADC12IERx** register.

Returns:

None

5.2.2.6 void ADC12_B_disableSamplingTimer (uint16_t *baseAddress*)

Disables Sampling Timer Pulse Mode.

Disables the Sampling Timer Pulse Mode. Note that if a conversion has been started with the `startConversion()` function, then a call to `disableConversions()` is required before this function may be called.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the ADC12B module.

Returns:

None

5.2.2.7 void ADC12_B_enable (uint16_t *baseAddress*)

Enables the ADC12B block.

This will enable operation of the ADC12B block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	16
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the ADC12B module.

Modified bits are **ADC12ON** of **ADC12CTL0** register.

Returns:

None

5.2.2.8 void ADC12_B_enableInterrupt (uint16_t *baseAddress*, uint16_t *interruptMask0*, uint16_t *interruptMask1*, uint16_t *interruptMask2*)

Enables selected ADC12B interrupt sources.

Enables the indicated ADC12B interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. **Does not clear interrupt flags.**

Parameters:

baseAddress is the base address of the ADC12B module.

interruptMask0 is the bit mask of the memory buffer and overflow interrupt sources to be enabled. If the desired interrupt is not available in the selection for *interruptMask0*, then simply pass in a '0' for this value. Valid values are:

- ADC12_B_IE0
- ADC12_B_IE1
- ADC12_B_IE2
- ADC12_B_IE3
- ADC12_B_IE4
- ADC12_B_IE5
- ADC12_B_IE6
- ADC12_B_IE7
- ADC12_B_IE8
- ADC12_B_IE9
- ADC12_B_IE10
- ADC12_B_IE11
- ADC12_B_IE12
- ADC12_B_IE13
- ADC12_B_IE14
- ADC12_B_IE15

interruptMask1 is the bit mask of the memory buffer and overflow interrupt sources to be enabled. If the desired interrupt is not available in the selection for *interruptMask1*, then simply pass in a '0' for this value. Valid values are:

- ADC12_B_IE16
- ADC12_B_IE17
- ADC12_B_IE18
- ADC12_B_IE19
- ADC12_B_IE20
- ADC12_B_IE21
- ADC12_B_IE22
- ADC12_B_IE23
- ADC12_B_IE24
- ADC12_B_IE25
- ADC12_B_IE26
- ADC12_B_IE27
- ADC12_B_IE28
- ADC12_B_IE29
- ADC12_B_IE30
- ADC12_B_IE31

interruptMask2 is the bit mask of the memory buffer and overflow interrupt sources to be enabled. If the desired interrupt is not available in the selection for *interruptMask2*, then simply pass in a '0' for this value. Valid values are:

- **ADC12_B_INIE** - Interrupt enable for a conversion in the result register is either greater than the ADC12LO or lower than the ADC12HI threshold. GIE bit must be set to enable the interrupt.
- **ADC12_B_LOIE** - Interrupt enable for the falling short of the lower limit interrupt of the window comparator for the result register. GIE bit must be set to enable the interrupt.
- **ADC12_B_HIIE** - Interrupt enable for the exceeding the upper limit of the window comparator for the result register. GIE bit must be set to enable the interrupt.
- **ADC12_B_OVIE** - Interrupt enable for a conversion that is about to save to a memory buffer that has not been read out yet. GIE bit must be set to enable the interrupt.
- **ADC12_B_TOVIE** - enable for a conversion that is about to start before the previous conversion has been completed. GIE bit must be set to enable the interrupt.
- **ADC12_B_RDYIE** - enable for the local buffered reference ready signal. GIE bit must be set to enable the interrupt.

Modified bits of ADC12IERx register.

Returns:

None

5.2.2.9 `uint16_t ADC12_B_getInterruptStatus (uint16_t baseAddress, uint8_t interruptRegisterChoice, uint16_t memoryInterruptFlagMask)`

Returns the status of the selected memory interrupt flags.

Returns the status of the selected memory interrupt flags. Note that the overflow interrupts do not have an interrupt flag to clear; they must be accessed directly from the interrupt vector.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	58
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	36
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	98
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the ADC12B module.

interruptRegisterChoice is either 0, 1, or 2, to choose the correct interrupt register to update

memoryInterruptFlagMask is the bit mask of the memory buffer and overflow interrupt flags to be cleared. Valid values are:

- **ADC12_B_IFG0** - interruptRegisterChoice = 0
- **ADC12_B_IFG1**
- **ADC12_B_IFG2**
- **ADC12_B_IFG3**
- **ADC12_B_IFG4**
- **ADC12_B_IFG5**
- **ADC12_B_IFG6**
- **ADC12_B_IFG7**
- **ADC12_B_IFG8**
- **ADC12_B_IFG9**
- **ADC12_B_IFG10**
- **ADC12_B_IFG11**
- **ADC12_B_IFG12**
- **ADC12_B_IFG13**
- **ADC12_B_IFG14**
- **ADC12_B_IFG15**
- **ADC12_B_IFG16** - interruptRegisterChoice = 1
- **ADC12_B_IFG17**
- **ADC12_B_IFG18**
- **ADC12_B_IFG19**
- **ADC12_B_IFG20**
- **ADC12_B_IFG21**
- **ADC12_B_IFG22**
- **ADC12_B_IFG23**
- **ADC12_B_IFG24**
- **ADC12_B_IFG25**
- **ADC12_B_IFG26**
- **ADC12_B_IFG27**
- **ADC12_B_IFG28**
- **ADC12_B_IFG29**
- **ADC12_B_IFG30**
- **ADC12_B_IFG31**
- **ADC12_B_INIFG** - interruptRegisterChoice = 2
- **ADC12_B_LOIFG**
- **ADC12_B_HIIFG**
- **ADC12_B_OVIFG**
- **ADC12_B_TOVIFG**
- **ADC12_B_RDYIFG** - The selected ADC12B interrupt flags are cleared, so that it no longer asserts. The memory buffer interrupt flags are only cleared when the memory buffer is accessed. Note that the overflow interrupts do not have an interrupt flag to clear; they must be accessed directly from the interrupt vector.

Returns:

The current interrupt flag status for the corresponding mask.

5.2.2.10 uint32_t ADC12_B_getMemoryAddressForDMA (uint16_t *baseAddress*, uint8_t *memoryIndex*)

Returns the address of the specified memory buffer for the DMA module.

Returns the address of the specified memory buffer. This can be used in conjunction with the DMA to store the converted data directly to memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the ADC12B module.

memoryIndex is the memory buffer to return the address of. Valid values are:

- ADC12_B_MEMORY_0
- ADC12_B_MEMORY_1
- ADC12_B_MEMORY_2
- ADC12_B_MEMORY_3
- ADC12_B_MEMORY_4
- ADC12_B_MEMORY_5
- ADC12_B_MEMORY_6
- ADC12_B_MEMORY_7
- ADC12_B_MEMORY_8
- ADC12_B_MEMORY_9
- ADC12_B_MEMORY_10
- ADC12_B_MEMORY_11
- ADC12_B_MEMORY_12
- ADC12_B_MEMORY_13
- ADC12_B_MEMORY_14
- ADC12_B_MEMORY_15
- ADC12_B_MEMORY_16
- ADC12_B_MEMORY_17
- ADC12_B_MEMORY_18
- ADC12_B_MEMORY_19
- ADC12_B_MEMORY_20
- ADC12_B_MEMORY_21
- ADC12_B_MEMORY_22
- ADC12_B_MEMORY_23

- ADC12_B_MEMORY_24
- ADC12_B_MEMORY_25
- ADC12_B_MEMORY_26
- ADC12_B_MEMORY_27
- ADC12_B_MEMORY_28
- ADC12_B_MEMORY_29
- ADC12_B_MEMORY_30
- ADC12_B_MEMORY_31

Returns:

address of the specified memory buffer

5.2.2.11 uint16_t ADC12_B_getResults (uint16_t *baseAddress*, uint8_t *memoryBufferIndex*)

Returns the raw contents of the specified memory buffer.

Returns the raw contents of the specified memory buffer. The format of the content depends on the read-back format of the data: if the data is in signed 2's complement format then the contents in the memory buffer will be left-justified with the least-significant bits as 0's, whereas if the data is in unsigned format then the contents in the memory buffer will be right-justified with the most-significant bits as 0's.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the ADC12B module.

memoryBufferIndex is the specified memory buffer to read. Valid values are:

- ADC12_B_MEMORY_0
- ADC12_B_MEMORY_1
- ADC12_B_MEMORY_2
- ADC12_B_MEMORY_3
- ADC12_B_MEMORY_4
- ADC12_B_MEMORY_5
- ADC12_B_MEMORY_6
- ADC12_B_MEMORY_7
- ADC12_B_MEMORY_8

- ADC12_B_MEMORY_9
- ADC12_B_MEMORY_10
- ADC12_B_MEMORY_11
- ADC12_B_MEMORY_12
- ADC12_B_MEMORY_13
- ADC12_B_MEMORY_14
- ADC12_B_MEMORY_15
- ADC12_B_MEMORY_16
- ADC12_B_MEMORY_17
- ADC12_B_MEMORY_18
- ADC12_B_MEMORY_19
- ADC12_B_MEMORY_20
- ADC12_B_MEMORY_21
- ADC12_B_MEMORY_22
- ADC12_B_MEMORY_23
- ADC12_B_MEMORY_24
- ADC12_B_MEMORY_25
- ADC12_B_MEMORY_26
- ADC12_B_MEMORY_27
- ADC12_B_MEMORY_28
- ADC12_B_MEMORY_29
- ADC12_B_MEMORY_30
- ADC12_B_MEMORY_31

Returns:

A signed integer of the contents of the specified memory buffer.

5.2.2.12 `bool ADC12_B_init (uint16_t baseAddress, uint16_t sampleHoldSignalSourceSelect, uint8_t clockSourceSelect, uint16_t clockSourceDivider, uint16_t clockSourcePredivider, uint16_t internalChannelMap)`

DEPRECATED - Initializes the ADC12B Module.

This function initializes the ADC module to allow for analog-to-digital conversions. Specifically this function sets up the sample-and-hold signal and clock sources for the ADC core to use for conversions. Upon successful completion of the initialization all of the ADC control registers will be reset, excluding the memory controls and reference module bits, the given parameters will be set, and the ADC core will be turned on (Note, that the ADC core only draws power during conversions and remains off when not converting). Note that sample/hold signal sources are device dependent. Note that if re-initializing the ADC after starting a conversion with the `startConversion()` function, the `disableConversion()` must be called BEFORE this function can be called.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	82
TI Compiler 4.2.1	Size	72
TI Compiler 4.2.1	Speed	72
IAR 5.51.6	None	72
IAR 5.51.6	Size	60
IAR 5.51.6	Speed	68
MSPGCC 4.8.0	None	92
MSPGCC 4.8.0	Size	70
MSPGCC 4.8.0	Speed	110

Parameters:

baseAddress is the base address of the ADC12B module.

sampleHoldSignalSourceSelect is the signal that will trigger a sample-and-hold for an input signal to be converted. Valid values are:

- **ADC12_B_SAMPLEHOLDSOURCE_SC** [Default]
 - **ADC12_B_SAMPLEHOLDSOURCE_1**
 - **ADC12_B_SAMPLEHOLDSOURCE_2**
 - **ADC12_B_SAMPLEHOLDSOURCE_3**
 - **ADC12_B_SAMPLEHOLDSOURCE_4**
 - **ADC12_B_SAMPLEHOLDSOURCE_5**
 - **ADC12_B_SAMPLEHOLDSOURCE_6**
 - **ADC12_B_SAMPLEHOLDSOURCE_7** - This parameter is device specific and sources should be found in the device's datasheet.
- Modified bits are **ADC12SHSx** of **ADC12CTL1** register.

clockSourceSelect selects the clock that will be used by the ADC12B core, and the sampling timer if a sampling pulse mode is enabled. Valid values are:

- **ADC12_B_CLOCKSOURCE_ADC12OSC** [Default] - MODOSC 5 MHz oscillator from the UCS
 - **ADC12_B_CLOCKSOURCE_ACLK** - The Auxiliary Clock
 - **ADC12_B_CLOCKSOURCE_MCLK** - The Master Clock
 - **ADC12_B_CLOCKSOURCE_SMCLK** - The Sub-Master Clock
- Modified bits are **ADC12SSELx** of **ADC12CTL1** register.

clockSourceDivider selects the amount that the clock will be divided. Valid values are:

- **ADC12_B_CLOCKDIVIDER_1** [Default]
 - **ADC12_B_CLOCKDIVIDER_2**
 - **ADC12_B_CLOCKDIVIDER_3**
 - **ADC12_B_CLOCKDIVIDER_4**
 - **ADC12_B_CLOCKDIVIDER_5**
 - **ADC12_B_CLOCKDIVIDER_6**
 - **ADC12_B_CLOCKDIVIDER_7**
 - **ADC12_B_CLOCKDIVIDER_8**
- Modified bits are **ADC12DIVx** of **ADC12CTL1** register.

clockSourcePredivider selects the amount that the clock will be predivided. Valid values are:

- **ADC12_B_CLOCKPREDIVIDER__1** [Default]
- **ADC12_B_CLOCKPREDIVIDER__4**

- **ADC12_B_CLOCKPREDIVIDER_32**
- **ADC12_B_CLOCKPREDIVIDER_64**

Modified bits are **ADC12PDIV** of **ADC12CTL1** register.

internalChannelMap selects what internal channel to map for ADC input channels Valid values are:

- **ADC12_B_MAPINTCH3**
- **ADC12_B_MAPINTCH2**
- **ADC12_B_MAPINTCH1**
- **ADC12_B_MAPINTCH0**
- **ADC12_B_TEMPSENSEMAP**
- **ADC12_B_BATTMAP**
- **ADC12_B_NOINTCH**

Modified bits of **ADC12CTL3** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the initialization process.

5.2.2.13 `bool ADC12_B_initialize (uint16_t baseAddress, ADC12_B_initializeParam *param)`

Initializes the ADC12B Module.

This function initializes the ADC module to allow for analog-to-digital conversions. Specifically this function sets up the sample-and-hold signal and clock sources for the ADC core to use for conversions. Upon successful completion of the initialization all of the ADC control registers will be reset, excluding the memory controls and reference module bits, the given parameters will be set, and the ADC core will be turned on (Note, that the ADC core only draws power during conversions and remains off when not converting). Note that sample/hold signal sources are device dependent. Note that if re-initializing the ADC after starting a conversion with the `startConversion()` function, the `disableConversion()` must be called BEFORE this function can be called.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	168
TI Compiler 4.2.1	Size	104
TI Compiler 4.2.1	Speed	104
IAR 5.51.6	None	188
IAR 5.51.6	Size	122
IAR 5.51.6	Speed	152
MSPGCC 4.8.0	None	274
MSPGCC 4.8.0	Size	110
MSPGCC 4.8.0	Speed	110

Parameters:

baseAddress is the base address of the ADC12B module.

param is the pointer to struct for initialization.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the initialization process.

5.2.2.14 uint8_t ADC12_B_isBusy (uint16_t *baseAddress*)

Returns the busy status of the ADC12B core.

Returns the status of the ADC core if there is a conversion currently taking place.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the ADC12B module.

Returns:

ADC12_B_BUSY or ADC12_B_NOTBUSY dependent if there is a conversion currently taking place. Return one of the following:

- **ADC12_B_NOTBUSY**
- **ADC12_B_BUSY**
indicating if a conversion is taking place

5.2.2.15 void ADC12_B_memoryConfigure (uint16_t *baseAddress*, uint8_t *memoryBufferControlIndex*, uint8_t *inputSourceSelect*, uint16_t *refVoltageSourceSelect*, uint16_t *endOfSequence*, uint16_t *windowComparatorSelect*, uint16_t *differentialModeSelect*)

DEPRECATED - Configures the controls of the selected memory buffer.

Maps an input signal conversion into the selected memory buffer, as well as the positive and negative reference voltages for each conversion being stored into this memory buffer. If the internal reference is used for the positive reference voltage, the internal REF module must be used to control the voltage level. Note that if a conversion has been started with the startConversion() function, then a call to disableConversions() is required before this function may be called.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	92
TI Compiler 4.2.1	Size	50
TI Compiler 4.2.1	Speed	50
IAR 5.51.6	None	80
IAR 5.51.6	Size	68
IAR 5.51.6	Speed	76
MSPGCC 4.8.0	None	94
MSPGCC 4.8.0	Size	80
MSPGCC 4.8.0	Speed	50

Parameters:

baseAddress is the base address of the ADC12B module.

memoryBufferControlIndex is the selected memory buffer to set the configuration for. Valid values are:

- ADC12_B_MEMORY_0
- ADC12_B_MEMORY_1
- ADC12_B_MEMORY_2
- ADC12_B_MEMORY_3
- ADC12_B_MEMORY_4
- ADC12_B_MEMORY_5
- ADC12_B_MEMORY_6
- ADC12_B_MEMORY_7
- ADC12_B_MEMORY_8
- ADC12_B_MEMORY_9
- ADC12_B_MEMORY_10
- ADC12_B_MEMORY_11
- ADC12_B_MEMORY_12
- ADC12_B_MEMORY_13
- ADC12_B_MEMORY_14
- ADC12_B_MEMORY_15
- ADC12_B_MEMORY_16
- ADC12_B_MEMORY_17
- ADC12_B_MEMORY_18
- ADC12_B_MEMORY_19
- ADC12_B_MEMORY_20
- ADC12_B_MEMORY_21
- ADC12_B_MEMORY_22
- ADC12_B_MEMORY_23
- ADC12_B_MEMORY_24
- ADC12_B_MEMORY_25
- ADC12_B_MEMORY_26
- ADC12_B_MEMORY_27
- ADC12_B_MEMORY_28
- ADC12_B_MEMORY_29
- ADC12_B_MEMORY_30

■ ADC12_B_MEMORY_31

inputSourceSelect is the input that will store the converted data into the specified memory buffer. Valid values are:

- ADC12_B_INPUT_A0 [Default]
- ADC12_B_INPUT_A1
- ADC12_B_INPUT_A2
- ADC12_B_INPUT_A3
- ADC12_B_INPUT_A4
- ADC12_B_INPUT_A5
- ADC12_B_INPUT_A6
- ADC12_B_INPUT_A7
- ADC12_B_INPUT_A8
- ADC12_B_INPUT_A9
- ADC12_B_INPUT_A10
- ADC12_B_INPUT_A11
- ADC12_B_INPUT_A12
- ADC12_B_INPUT_A13
- ADC12_B_INPUT_A14
- ADC12_B_INPUT_A15
- ADC12_B_INPUT_A16
- ADC12_B_INPUT_A17
- ADC12_B_INPUT_A18
- ADC12_B_INPUT_A19
- ADC12_B_INPUT_A20
- ADC12_B_INPUT_A21
- ADC12_B_INPUT_A22
- ADC12_B_INPUT_A23
- ADC12_B_INPUT_A24
- ADC12_B_INPUT_A25
- ADC12_B_INPUT_A26
- ADC12_B_INPUT_A27
- ADC12_B_INPUT_A28
- ADC12_B_INPUT_A29
- ADC12_B_INPUT_TCMAP
- ADC12_B_INPUT_BATMAP

Modified bits are **ADC12INCHx** of **ADC12MCTLx** register.

refVoltageSourceSelect is the reference voltage source to set as the upper/lower limits for the conversion stored in the specified memory. Valid values are:

- ADC12_B_VREFPOS_AVCC_VREFNEG_VSS [Default]
- ADC12_B_VREFPOS_INTBUF_VREFNEG_VSS
- ADC12_B_VREFPOS_EXTNEG_VREFNEG_VSS
- ADC12_B_VREFPOS_EXTBUF_VREFNEG_VSS
- ADC12_B_VREFPOS_EXTPOS_VREFNEG_VSS
- ADC12_B_VREFPOS_AVCC_VREFNEG_EXTBUF
- ADC12_B_VREFPOS_AVCC_VREFNEG_EXTPOS
- ADC12_B_VREFPOS_INTBUF_VREFNEG_EXTPOS

- **ADC12_B_VREFPOS_AVCC_VREFNEG_INTBUF**
- **ADC12_B_VREFPOS_EXTPOS_VREFNEG_INTBUF**
- **ADC12_B_VREFPOS_AVCC_VREFNEG_EXTNEG**
- **ADC12_B_VREFPOS_INTBUF_VREFNEG_EXTNEG**
- **ADC12_B_VREFPOS_EXTPOS_VREFNEG_EXTNEG**
- **ADC12_B_VREFPOS_EXTBUF_VREFNEG_EXTNEG**

Modified bits are **ADC12VRSEL** of **ADC12MCTLx** register.

endOfSequence indicates that the specified memory buffer will be the end of the sequence if a sequenced conversion mode is selected Valid values are:

- **ADC12_B_NOTENDOFSEQUENCE** [Default] - The specified memory buffer will NOT be the end of the sequence OR a sequenced conversion mode is not selected.
- **ADC12_B_ENDOFSEQUENCE** - The specified memory buffer will be the end of the sequence.

Modified bits are **ADC12EOS** of **ADC12MCTLx** register.

windowComparatorSelect sets the window comparator mode Valid values are:

- **ADC12_B_WINDOW_COMPARATOR_DISABLE** [Default]
- **ADC12_B_WINDOW_COMPARATOR_ENABLE**

differentialModeSelect sets the differential mode Valid values are:

- **ADC12_B_DIFFERENTIAL_MODE_DISABLE** [Default]
- **ADC12_B_DIFFERENTIAL_MODE_ENABLE**

Returns:

None

5.2.2.16 void ADC12_B_setAdcPowerMode (uint16_t *baseAddress*, uint8_t *powerMode*)

Use to set the ADC's power conservation mode if the sampling rate is at 50-ksps or less.

Sets ADC's power mode. If the user has a sampling rate greater than 50-ksps, then he/she can only enable **ADC12_B_REGULARPOWERMODE**. If the sampling rate is 50-ksps or less, the user can enable **ADC12_B_LOWPOWERMODE** granting additional power savings.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	32
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	18
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the ADC12B module.

powerMode is the specified maximum sampling rate. Valid values are:

- **ADC12_B_REGULARPOWERMODE** [Default] - If sampling rate is greater than 50-ksps, there is no power saving feature available.
- **ADC12_B_LOWPPOWERMODE** - If sampling rate is less than or equal to 50-ksps, select this value to save power
Modified bits are **ADC12SR** of **ADC12CTL2** register.

Returns:

None

5.2.2.17 void ADC12_B_setDataReadBackFormat (uint16_t *baseAddress*, uint8_t *readBackFormat*)

Use to set the read-back format of the converted data.

Sets the format of the converted data: how it will be stored into the memory buffer, and how it should be read back. The format can be set as right-justified (default), which indicates that the number will be unsigned, or left-justified, which indicates that the number will be signed in 2's complement format. This change affects all memory buffers for subsequent conversions.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	32
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	18
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the ADC12B module.

readBackFormat is the specified format to store the conversions in the memory buffer. Valid values are:

- **ADC12_B_UNSIGNED_BINARY** [Default]
- **ADC12_B_SIGNED_2SCOMPLEMENT**
Modified bits are **ADC12DF** of **ADC12CTL2** register.

Returns:

None

5.2.2.18 void ADC12_B_setResolution (uint16_t *baseAddress*, uint8_t *resolutionSelect*)

Use to change the resolution of the converted data.

This function can be used to change the resolution of the converted data from the default of 12-bits.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the ADC12B module.

resolutionSelect determines the resolution of the converted data. Valid values are:

- **ADC12_B_RESOLUTION_8BIT**
- **ADC12_B_RESOLUTION_10BIT**
- **ADC12_B_RESOLUTION_12BIT** [Default]
Modified bits are **ADC12RESx** of **ADC12CTL2** register.

Returns:

None

5.2.2.19 void ADC12_B_setSampleHoldSignalInversion (uint16_t *baseAddress*, uint16_t *invertedSignal*)

Use to invert or un-invert the sample/hold signal.

This function can be used to invert or un-invert the sample/hold signal. Note that if a conversion has been started with the startConversion() function, then a call to disableConversions() is required before this function may be called.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the ADC12B module.

invertedSignal set if the sample/hold signal should be inverted Valid values are:

- **ADC12_B_NONINVERTEDSIGNAL** [Default] - a sample-and-hold of an input signal for conversion will be started on a rising edge of the sample/hold signal.
 - **ADC12_B_INVERTEDSIGNAL** - a sample-and-hold of an input signal for conversion will be started on a falling edge of the sample/hold signal.
- Modified bits are **ADC12ISSH** of **ADC12CTL1** register.

Returns:

None

5.2.2.20 void ADC12_B_setupSamplingTimer (uint16_t *baseAddress*, uint16_t *clockCycleHoldCountLowMem*, uint16_t *clockCycleHoldCountHighMem*, uint16_t *multipleSamplesEnabled*)

Sets up and enables the Sampling Timer Pulse Mode.

This function sets up the sampling timer pulse mode which allows the sample/hold signal to trigger a sampling timer to sample-and-hold an input signal for a specified number of clock cycles without having to hold the sample/hold signal for the entire period of sampling. Note that if a conversion has been started with the startConversion() function, then a call to disableConversions() is required before this function may be called.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	60
TI Compiler 4.2.1	Size	24
TI Compiler 4.2.1	Speed	24
IAR 5.51.6	None	36
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Parameters:

baseAddress is the base address of the ADC12B module.

clockCycleHoldCountLowMem sets the amount of clock cycles to sample- and-hold for the higher memory buffers 0-7. Valid values are:

- **ADC12_B_CYCLEHOLD_4_CYCLES** [Default]
- **ADC12_B_CYCLEHOLD_8_CYCLES**
- **ADC12_B_CYCLEHOLD_16_CYCLES**
- **ADC12_B_CYCLEHOLD_32_CYCLES**
- **ADC12_B_CYCLEHOLD_64_CYCLES**
- **ADC12_B_CYCLEHOLD_96_CYCLES**
- **ADC12_B_CYCLEHOLD_128_CYCLES**
- **ADC12_B_CYCLEHOLD_192_CYCLES**
- **ADC12_B_CYCLEHOLD_256_CYCLES**

- **ADC12_B_CYCLEHOLD_384_CYCLES**
- **ADC12_B_CYCLEHOLD_512_CYCLES**
- **ADC12_B_CYCLEHOLD_768_CYCLES**
- **ADC12_B_CYCLEHOLD_1024_CYCLES**

Modified bits are **ADC12SHT0x** of **ADC12CTL0** register.

clockCycleHoldCountHighMem sets the amount of clock cycles to sample-and-hold for the higher memory buffers 8-15. Valid values are:

- **ADC12_B_CYCLEHOLD_4_CYCLES** [Default]
- **ADC12_B_CYCLEHOLD_8_CYCLES**
- **ADC12_B_CYCLEHOLD_16_CYCLES**
- **ADC12_B_CYCLEHOLD_32_CYCLES**
- **ADC12_B_CYCLEHOLD_64_CYCLES**
- **ADC12_B_CYCLEHOLD_96_CYCLES**
- **ADC12_B_CYCLEHOLD_128_CYCLES**
- **ADC12_B_CYCLEHOLD_192_CYCLES**
- **ADC12_B_CYCLEHOLD_256_CYCLES**
- **ADC12_B_CYCLEHOLD_384_CYCLES**
- **ADC12_B_CYCLEHOLD_512_CYCLES**
- **ADC12_B_CYCLEHOLD_768_CYCLES**
- **ADC12_B_CYCLEHOLD_1024_CYCLES**

Modified bits are **ADC12SHT1x** of **ADC12CTL0** register.

multipleSamplesEnabled allows multiple conversions to start without a trigger signal from the sample/hold signal Valid values are:

- **ADC12_B_MULTIPLESAMPLESDISABLE** [Default] - a timer trigger will be needed to start every ADC conversion.
 - **ADC12_B_MULTIPLESAMPLESENABLE** - during a sequenced and/or repeated conversion mode, after the first conversion, no sample/hold signal is necessary to start subsequent sample/hold and convert processes.
- Modified bits are **ADC12MSC** of **ADC12CTL0** register.

Returns:

None

5.2.2.21 void ADC12_B_setWindowCompAdvanced (uint16_t *baseAddress*, uint16_t *highThreshold*, uint16_t *lowThreshold*)

Sets the high and low threshold for the window comparator feature.

Sets the high and low threshold for the window comparator feature. Use the ADC12HIIE, ADC12INIE, ADC12LOIE interrupts to utilize this feature.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	36
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	20
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the ADC12B module.

highThreshold is the upper bound that could trip an interrupt for the window comparator.

lowThreshold is the lower bound that could trip on interrupt for the window comparator.

Returns:

None

5.2.2.22 void ADC12_B_startConversion (uint16_t baseAddress, uint16_t startingMemoryBufferIndex, uint8_t conversionSequenceModeSelect)

Enables/Starts an Analog-to-Digital Conversion.

Enables/starts the conversion process of the ADC. If the sample/hold signal source chosen during initialization was ADC12OSC, then the conversion is started immediately, otherwise the chosen sample/hold signal source starts the conversion by a rising edge of the signal. Keep in mind when selecting conversion modes, that for sequenced and/or repeated modes, to keep the sample/hold-and-convert process continuing without a trigger from the sample/hold signal source, the multiple samples must be enabled using the [ADC12_B_setupSamplingTimer\(\)](#) function. Note that after this function is called, the `ADC12_B_stopConversions()` has to be called to re-initialize the ADC, reconfigure a memory buffer control, enable/disable the sampling timer, or to change the internal reference voltage.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	82
TI Compiler 4.2.1	Size	44
TI Compiler 4.2.1	Speed	44
IAR 5.51.6	None	56
IAR 5.51.6	Size	48
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	162
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

Parameters:

baseAddress is the base address of the ADC12B module.

startingMemoryBufferIndex is the memory buffer that will hold the first or only conversion.

Valid values are:

- **ADC12_B_START_AT_ADC12MEM0** [Default]
- **ADC12_B_START_AT_ADC12MEM1**
- **ADC12_B_START_AT_ADC12MEM2**
- **ADC12_B_START_AT_ADC12MEM3**
- **ADC12_B_START_AT_ADC12MEM4**
- **ADC12_B_START_AT_ADC12MEM5**
- **ADC12_B_START_AT_ADC12MEM6**
- **ADC12_B_START_AT_ADC12MEM7**
- **ADC12_B_START_AT_ADC12MEM8**
- **ADC12_B_START_AT_ADC12MEM9**
- **ADC12_B_START_AT_ADC12MEM10**
- **ADC12_B_START_AT_ADC12MEM11**
- **ADC12_B_START_AT_ADC12MEM12**
- **ADC12_B_START_AT_ADC12MEM13**
- **ADC12_B_START_AT_ADC12MEM14**
- **ADC12_B_START_AT_ADC12MEM15**
- **ADC12_B_START_AT_ADC12MEM16**
- **ADC12_B_START_AT_ADC12MEM17**
- **ADC12_B_START_AT_ADC12MEM18**
- **ADC12_B_START_AT_ADC12MEM19**
- **ADC12_B_START_AT_ADC12MEM20**
- **ADC12_B_START_AT_ADC12MEM21**
- **ADC12_B_START_AT_ADC12MEM22**
- **ADC12_B_START_AT_ADC12MEM23**
- **ADC12_B_START_AT_ADC12MEM24**
- **ADC12_B_START_AT_ADC12MEM25**
- **ADC12_B_START_AT_ADC12MEM26**
- **ADC12_B_START_AT_ADC12MEM27**
- **ADC12_B_START_AT_ADC12MEM28**
- **ADC12_B_START_AT_ADC12MEM29**
- **ADC12_B_START_AT_ADC12MEM30**
- **ADC12_B_START_AT_ADC12MEM31**

Modified bits are **ADC12CSTARTADDx** of **ADC12CTL1** register.

conversionSequenceModeSelect determines the ADC operating mode. Valid values are:

- **ADC12_B_SINGLECHANNEL** [Default] - one-time conversion of a single channel into a single memory buffer.
- **ADC12_B_SEQOFCHANNELS** - one time conversion of multiple channels into the specified starting memory buffer and each subsequent memory buffer up until the conversion is stored in a memory buffer dedicated as the end-of-sequence by the memory's control register.
- **ADC12_B_REPEATED_SINGLECHANNEL** - repeated conversions of one channel into a single memory buffer.
- **ADC12_B_REPEATED_SEQOFCHANNELS** - repeated conversions of multiple channels into the specified starting memory buffer and each subsequent memory buffer up until the conversion is stored in a memory buffer dedicated as the end-of-sequence by the memory's control register.

Modified bits are **ADC12CONSEQx** of **ADC12CTL1** register.

Modified bits of **ADC12CTL1** register and bits of **ADC12CTL0** register.

Returns:

None

5.3 Programming Example

The following example shows how to initialize and use the ADC12_B API to start a single channel with single conversion using an external positive reference for the ADC12_B.

```
//Initialize the ADC12 Module
/*
Base address of ADC12 Module
Use internal ADC12 bit as sample/hold signal to start conversion
USE MODOSC 5MHZ Digital Oscillator as clock source
Use default clock divider/pre-divider of 1
Map to internal channel 0
*/
ADC12_B_init(ADC12_B_BASE,
             ADC12_B_SAMPLEHOLDSOURCE_SC,
             ADC12_B_CLOCKSOURCE_ADC12OSC,
             ADC12_B_CLOCKDIVIDER_1,
             ADC12_B_CLOCKPREDIVIDER__1,
             ADC12_B_MAPINTCH0);

//Enable the ADC12_B module
ADC12_B_enable(ADC12_B_BASE);

/*
Base address of ADC12 Module
For memory buffers 0-7 sample/hold for 16 clock cycles
For memory buffers 8-15 sample/hold for 4 clock cycles (default)
Disable Multiple Sampling
*/
ADC12_B_setupSamplingTimer(ADC12_B_BASE,
                           ADC12_B_CYCLEHOLD_16_CYCLES,
                           ADC12_B_CYCLEHOLD_4_CYCLES,
                           ADC12_B_MULTIPLESAMPLESDISABLE);

//Configure Memory Buffer
/*
Base address of the ADC12 Module
Configure memory buffer 0
Map input A0 to memory buffer 0
Vref+ = AVcc
Vref- = EXT Positive
Memory buffer 0 is not the end of a sequence
*/
ADC12_B_memoryConfigure(ADC12_B_BASE,
                        ADC12_B_MEMORY_0,
                        ADC12_B_INPUT_A0,
                        ADC12_B_VREFPOS_EXTPOS_VREFNEG_VSS,
                        ADC12_B_NOTENDOFSEQUENCE,
                        ADC12_B_WINDOW_COMPARATOR_DISABLE,
                        ADC12_B_DIFFERENTIAL_MODE_DISABLE);

while (1)
{
    //Enable/Start first sampling and conversion cycle
    /*
Base address of ADC12 Module
```



```
Start the conversion into memory buffer 0
Use the single-channel, single-conversion mode
*/
    ADC12_B_startConversion(ADC12_B_BASE,
                            ADC12_B_MEMORY_0,
                            ADC12_B_SINGLECHANNEL);

//Poll for interrupt on memory buffer 0
while (!ADC12_B_getInterruptStatus(ADC12_B_BASE,
                                   0,
                                   ADC12_B_IFG0));

__no_operation();                // SET BREAKPOINT HERE
}
```

6 Advanced Encryption Standard (AES256)

Introduction	47
API Functions	47
Programming Example	57

6.1 Introduction

The AES256 accelerator module performs encryption and decryption of 128-bit data with 128-bit keys according to the advanced encryption standard (AES256) (FIPS PUB 197) in hardware. The AES256 accelerator features are:

- Encryption and decryption according to AES256 FIPS PUB 197 with 128-bit key
- On-the-fly key expansion for encryption and decryption
- Off-line key generation for decryption
- Byte and word access to key, input, and output data
- AES256 ready interrupt flag The AES256256 accelerator module performs encryption and decryption of 128-bit data with 128-/192-/256-bit keys according to the advanced encryption standard (AES256) (FIPS PUB 197) in hardware. The AES256 accelerator features are: AES256 encryption ? 128 bit - 168 cycles ? 192 bit - 204 cycles ? 256 bit - 234 cycles AES256 decryption ? 128 bit - 168 cycles ? 192 bit - 206 cycles ? 256 bit - 234 cycles
- On-the-fly key expansion for encryption and decryption
- Offline key generation for decryption
- Shadow register storing the initial key for all key lengths
- Byte and word access to key, input data, and output data
- AES256 ready interrupt flag

This driver is contained in `aes256.c`, with `aes256.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	1358
TI Compiler 4.2.1	Size	654
TI Compiler 4.2.1	Speed	1452
IAR 5.51.6	None	994
IAR 5.51.6	Size	474
IAR 5.51.6	Speed	652
MSPGCC 4.8.0	None	1716
MSPGCC 4.8.0	Size	728
MSPGCC 4.8.0	Speed	1430

6.2 API Functions

Functions

- void [AES256_clearErrorFlag](#) (uint16_t baseAddress)
- void [AES256_clearInterruptFlag](#) (uint16_t baseAddress)
- void [AES256_decryptData](#) (uint16_t baseAddress, const uint8_t *data, uint8_t *decryptedData)

- void [AES256_disableInterrupt](#) (uint16_t baseAddress)
- void [AES256_enableInterrupt](#) (uint16_t baseAddress)
- void [AES256_encryptData](#) (uint16_t baseAddress, const uint8_t *data, uint8_t *encryptedData)
- uint8_t [AES256_getDataOut](#) (uint16_t baseAddress, uint8_t *outputData)
- uint32_t [AES256_getErrorFlagStatus](#) (uint16_t baseAddress)
- uint32_t [AES256_getInterruptFlagStatus](#) (uint16_t baseAddress)
- uint16_t [AES256_isBusy](#) (uint16_t baseAddress)
- void [AES256_reset](#) (uint16_t baseAddress)
- uint8_t [AES256_setCipherKey](#) (uint16_t baseAddress, const uint8_t *cipherKey, uint16_t keyLength)
- uint8_t [AES256_setDecipherKey](#) (uint16_t baseAddress, const uint8_t *cipherKey, uint16_t keyLength)
- void [AES256_startDecryptData](#) (uint16_t baseAddress, const uint8_t *data)
- void [AES256_startEncryptData](#) (uint16_t baseAddress, const uint8_t *data)
- uint8_t [AES256_startSetDecipherKey](#) (uint16_t baseAddress, const uint8_t *cipherKey, uint16_t keyLength)

6.2.1 Detailed Description

The AES256 module APIs are

- [AES256_setCipherKey\(\)](#),
- [AES256256_setCipherKey\(\)](#),
- [AES256_encryptData\(\)](#),
- [AES256_decryptDataUsingEncryptionKey\(\)](#),
- [AES256_generateFirstRoundKey\(\)](#),
- [AES256_decryptData\(\)](#),
- [AES256_reset\(\)](#),
- [AES256_startEncryptData\(\)](#),
- [AES256_startDecryptDataUsingEncryptionKey\(\)](#),
- [AES256_startDecryptData\(\)](#),
- [AES256_startGenerateFirstRoundKey\(\)](#),
- [AES256_getDataOut\(\)](#)

The AES256 interrupt handler functions

- [AES256_enableInterrupt\(\)](#),
- [AES256_disableInterrupt\(\)](#),
- [AES256_clearInterruptFlag\(\)](#),

6.2.2 Function Documentation

6.2.2.1 void [AES256_clearErrorFlag](#) (uint16_t *baseAddress*)

Clears the AES256 error flag.

Clears the AES256 error flag that results from a key or data being written while the AES256 module is busy.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the AES256 module.

Modified bits are **AESERRFG** of **AESACTL0** register.

Returns:

None

6.2.2.2 void AES256_clearInterruptFlag (uint16_t baseAddress)

Clears the AES256 ready interrupt flag.

This function clears the AES256 ready interrupt flag. This flag is automatically cleared when AES256ADOUT is read, or when AES256AKEY or AES256ADIN is written. This function should be used when the flag needs to be reset and it has not been automatically cleared by one of the previous actions.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the AES256 module.

Modified bits are **AESRDYIFG** of **AESACTL0** register.

Returns:

None

6.2.2.3 void AES256_decryptData (uint16_t baseAddress, const uint8_t * data, uint8_t * decryptedData)

Decrypts a block of data using the AES256 module.

This function requires a pregenerated decryption key. A key can be loaded and pregenerated by using function [AES256_setDecipherKey\(\)](#) or [AES256_startSetDecipherKey\(\)](#). The decryption takes 167 MCLK.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	188
TI Compiler 4.2.1	Size	78
TI Compiler 4.2.1	Speed	304
IAR 5.51.6	None	138
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	234
MSPGCC 4.8.0	Size	92
MSPGCC 4.8.0	Speed	288

Parameters:

baseAddress is the base address of the AES256 module.

data is a pointer to an uint8_t array with a length of 16 bytes that contains encrypted data to be decrypted.

decryptedData is a pointer to an uint8_t array with a length of 16 bytes in that the decrypted data will be written.

Returns:

None

6.2.2.4 void AES256_disableInterrupt (uint16_t baseAddress)

Disables AES256 ready interrupt.

Disables AES256 ready interrupt. This interrupt is reset by a PUC, but not reset by AES256_reset.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the AES256 module.

Modified bits are **AESRDYIE** of **AESACTL0** register.

Returns:

None

6.2.2.5 void AES256_enableInterrupt (uint16_t baseAddress)

Enables AES256 ready interrupt.

Enables AES256 ready interrupt. This interrupt is reset by a PUC, but not reset by AES256_reset.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the AES256 module.

Modified bits are **AESRDYIE** of **AESACTLO** register.

Returns:

None

6.2.2.6 void AES256_encryptData (uint16_t baseAddress, const uint8_t * data, uint8_t * encryptedData)

Encrypts a block of data using the AES256 module.

The cipher key that is used for encryption should be loaded in advance by using function [AES256_setCipherKey\(\)](#)

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	188
TI Compiler 4.2.1	Size	78
TI Compiler 4.2.1	Speed	302
IAR 5.51.6	None	138
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	56
MSPGCC 4.8.0	None	234
MSPGCC 4.8.0	Size	92
MSPGCC 4.8.0	Speed	288

Parameters:

baseAddress is the base address of the AES256 module.

data is a pointer to an uint8_t array with a length of 16 bytes that contains data to be encrypted.

encryptedData is a pointer to an uint8_t array with a length of 16 bytes in that the encrypted data will be written.

Returns:

None

6.2.2.7 uint8_t AES256_getDataOut (uint16_t baseAddress, uint8_t * outputData)

Reads back the output data from AES256 module.

This function is meant to use after an encryption or decryption process that was started and finished by initiating an interrupt by use of AES256_startEncryptData or AES256_startDecryptData functions.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	98
TI Compiler 4.2.1	Size	42
TI Compiler 4.2.1	Speed	146
IAR 5.51.6	None	82
IAR 5.51.6	Size	56
IAR 5.51.6	Speed	58
MSPGCC 4.8.0	None	124
MSPGCC 4.8.0	Size	52
MSPGCC 4.8.0	Speed	130

Parameters:

baseAddress is the base address of the AES256 module.

outputData is a pointer to an uint8_t array with a length of 16 bytes in that the data will be written.

Returns:

STATUS_SUCCESS if data is valid, otherwise STATUS_FAIL

6.2.2.8 uint32_t AES256_getErrorFlagStatus (uint16_t baseAddress)

Gets the AES256 error flag status.

Checks the AES256 error flag that results from a key or data being written while the AES256 module is busy. If the flag is set, it needs to be cleared using AES256_clearErrorFlag.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the AES256 module.

Returns:

One of the following:

- **AES256_ERROR_OCCURRED**
- **AES256_NO_ERROR**
indicating the error flag status

6.2.2.9 uint32_t AES256_getInterruptFlagStatus (uint16_t baseAddress)

Gets the AES256 ready interrupt flag status.

This function checks the AES256 ready interrupt flag. This flag is automatically cleared when AES256ADOUT is read, or when AES256AKEY or AES256ADIN is written. This function can be used to confirm that this has been done.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	24
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	30
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the AES256 module.

Returns:

One of the following:

- **AES256_READY_INTERRUPT**
- **AES256_NOTREADY_INTERRUPT**
indicating the status of the AES256 ready status

6.2.2.10 uint16_t AES256_isBusy (uint16_t baseAddress)

Gets the AES256 module busy status.

Gets the AES256 module busy status. If a key or data are written while the AES256 module is busy, an error flag will be thrown.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the AES256 module.

Returns:

One of the following:

- **AES256_BUSY**
- **AES256_NOT_BUSY**
indicating if the AES256 module is busy

6.2.2.11 void AES256_reset (uint16_t baseAddress)

Resets AES256 Module immediately.

This function performs a software reset on the AES256 Module, note that this does not affect the AES256 ready interrupt.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the AES256 module.

Modified bits are **AESSWRST** of **AESACTL0** register.

Returns:

None

6.2.2.12 uint8_t AES256_setCipherKey (uint16_t baseAddress, const uint8_t * cipherKey, uint16_t keyLength)

Loads a 128, 192 or 256 bit cipher key to AES256 module.

This function loads a 128, 192 or 256 bit cipher key to AES256 module. Requires both a key as well as the length of the key provided. Acceptable key lengths are AES256_KEYLENGTH_128BIT, AES256_KEYLENGTH_192BIT, or AES256_KEYLENGTH_256BIT

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	174
TI Compiler 4.2.1	Size	98
TI Compiler 4.2.1	Speed	98
IAR 5.51.6	None	130
IAR 5.51.6	Size	82
IAR 5.51.6	Speed	102
MSPGCC 4.8.0	None	220
MSPGCC 4.8.0	Size	110
MSPGCC 4.8.0	Speed	112

Parameters:

baseAddress is the base address of the AES256 module.

cipherKey is a pointer to an uint8_t array with a length of 16 bytes that contains a 128 bit cipher key.

keyLength is the length of the key. Valid values are:

- AES256_KEYLENGTH_128BIT
- AES256_KEYLENGTH_192BIT
- AES256_KEYLENGTH_256BIT

Returns:

STATUS_SUCCESS or STATUS_FAIL of key loading

6.2.2.13 uint8_t AES256_setDecipherKey (uint16_t baseAddress, const uint8_t * cipherKey, uint16_t keyLength)

Sets the decipher key.

The API AES256_startSetDecipherKey or AES256_setDecipherKey must be invoked before invoking AES256_startDecryptData.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	186
TI Compiler 4.2.1	Size	100
TI Compiler 4.2.1	Speed	100
IAR 5.51.6	None	138
IAR 5.51.6	Size	74
IAR 5.51.6	Speed	104
MSPGCC 4.8.0	None	238
MSPGCC 4.8.0	Size	112
MSPGCC 4.8.0	Speed	114

Parameters:

baseAddress is the base address of the AES256 module.

cipherKey is a pointer to an uint8_t array with a length of 16 bytes that contains a 128 bit cipher key.

keyLength is the length of the key. Valid values are:

- AES256_KEYLENGTH_128BIT
- AES256_KEYLENGTH_192BIT
- AES256_KEYLENGTH_256BIT

Returns:

STATUS_SUCCESS or STATUS_FAIL of key loading

6.2.2.14 void AES256_startDecryptData (uint16_t baseAddress, const uint8_t * data)

Decrypts a block of data using the AES256 module.

This is the non-blocking equivalent of [AES256_decryptData\(\)](#). This function requires a pregenerated decryption key. A key can be loaded and pregenerated by using function [AES256_setDecipherKey\(\)](#) or [AES256_startSetDecipherKey\(\)](#). The decryption takes 167 MCLK. It is recommended to use interrupt to check for procedure completion then use the [AES256_getDataOut\(\)](#) API to retrieve the decrypted data.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	100
TI Compiler 4.2.1	Size	44
TI Compiler 4.2.1	Speed	168
IAR 5.51.6	None	74
IAR 5.51.6	Size	30
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	120
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	162

Parameters:

baseAddress is the base address of the AES256 module.

data is a pointer to an uint8_t array with a length of 16 bytes that contains encrypted data to be decrypted.

Returns:

None

6.2.2.15 void AES256_startEncryptData (uint16_t *baseAddress*, const uint8_t * *data*)

Starts an encryption process on the AES256 module.

The cipher key that is used for decryption should be loaded in advance by using function [AES256_setCipherKey\(\)](#). This is a non-blocking equivalent of [AES256_encryptData\(\)](#). It is recommended to use the interrupt functionality to check for procedure completion then use the [AES256_getDataOut\(\)](#) API to retrieve the encrypted data.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	100
TI Compiler 4.2.1	Size	44
TI Compiler 4.2.1	Speed	164
IAR 5.51.6	None	74
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	56
MSPGCC 4.8.0	None	120
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	162

Parameters:

baseAddress is the base address of the AES256 module.

data is a pointer to an uint8_t array with a length of 16 bytes that contains data to be encrypted.

Returns:

None

6.2.2.16 uint8_t AES256_startSetDecipherKey (uint16_t *baseAddress*, const uint8_t * *cipherKey*, uint16_t *keyLength*)

Sets the decipher key.

The API [AES256_startSetDecipherKey\(\)](#) or [AES256_setDecipherKey\(\)](#) must be invoked before invoking [AES256_startDecryptData](#).

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	174
TI Compiler 4.2.1	Size	94
TI Compiler 4.2.1	Speed	94
IAR 5.51.6	None	128
IAR 5.51.6	Size	72
IAR 5.51.6	Speed	90
MSPGCC 4.8.0	None	222
MSPGCC 4.8.0	Size	98
MSPGCC 4.8.0	Speed	102

Parameters:

baseAddress is the base address of the AES256 module.

cipherKey is a pointer to an uint8_t array with a length of 16 bytes that contains a 128 bit cipher key.

keyLength is the length of the key. Valid values are:

- **AES256_KEYLENGTH_128BIT**
- **AES256_KEYLENGTH_192BIT**
- **AES256_KEYLENGTH_256BIT**

Returns:

STATUS_SUCCESS or STATUS_FAIL of key loading

6.3 Programming Example

The following example shows some AES256 operations using the APIs

```

unsigned char Data[16] =
{
    0x30, 0x30, 0x30, 0x30,
    0x30, 0x30, 0x30, 0x30,
    0x30, 0x30, 0x30, 0x30,
    0x30, 0x30, 0x30, 0x30
};

unsigned char CipherKey[32] =
{
    0xAA, 0xBB, 0x02, 0x03,
    0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0A, 0x0B,
    0x0C, 0x0D, 0x0E, 0x0F,
    0x30, 0x31, 0x32, 0x33,
    0x34, 0x35, 0x36, 0x37,
    0x30, 0x31, 0x32, 0x33,
    0x34, 0x35, 0x36, 0x37
};

unsigned char DataAESencrypted[16];          // Encrypted data
unsigned char DataAESdecrypted[16];          // Decrypted data

// Load a cipher key to module
AES256_setCipherKey(AES256_BASE, CipherKey, Key_256BIT);

// Encrypt data with preloaded cipher key
AES256_encryptData(AES256_BASE, Data, DataAESencrypted);

// Decrypt data with keys that were generated during encryption - takes 214 MCLK
// This function will generate all round keys needed for decryption first and then
// the encryption process starts
AES256_decryptDataUsingEncryptionKey(AES256_BASE, DataAESencrypted, DataAESdecrypted);

```

7 Comparator (COMP_E)

Introduction	58
API Functions	58
Programming Example	71

7.1 Introduction

The Comparator E (COMP_E) API provides a set of functions for using the MSP430Ware COMP_E modules. Functions are provided to initialize the COMP_E modules, setup reference voltages for input, and manage interrupts for the COMP_E modules.

The COMP_E module provides the ability to compare two analog signals and use the output in software and on an output pin. The output represents whether the signal on the positive terminal is higher than the signal on the negative terminal. The COMP_E may be used to generate a hysteresis. There are 16 different inputs that can be used, as well as the ability to short 2 input together. The COMP_E module also has control over the REF module to generate a reference voltage as an input.

The COMP_E module can generate multiple interrupts. An interrupt may be asserted for the output, with separate interrupts on whether the output rises, or falls.

This driver is contained in `comp_e.c`, with `comp_e.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	802
TI Compiler 4.2.1	Size	400
TI Compiler 4.2.1	Speed	398
IAR 5.51.6	None	566
IAR 5.51.6	Size	422
IAR 5.51.6	Speed	792
MSPGCC 4.8.0	None	1212
MSPGCC 4.8.0	Size	422
MSPGCC 4.8.0	Speed	1948

7.2 API Functions

Functions

- void [COMP_E_clearInterrupt](#) (uint16_t baseAddress, uint16_t interruptFlagMask)
- void [COMP_E_disable](#) (uint16_t baseAddress)
- void [COMP_E_disableInputBuffer](#) (uint16_t baseAddress, uint16_t inputPort)
- void [COMP_E_disableInterrupt](#) (uint16_t baseAddress, uint16_t interruptMask)
- void [COMP_E_enable](#) (uint16_t baseAddress)
- void [COMP_E_enableInputBuffer](#) (uint16_t baseAddress, uint16_t inputPort)
- void [COMP_E_enableInterrupt](#) (uint16_t baseAddress, uint16_t interruptMask)
- uint8_t [COMP_E_getInterruptStatus](#) (uint16_t baseAddress, uint16_t interruptFlagMask)
- bool [COMP_E_init](#) (uint16_t baseAddress, uint16_t posTerminalInput, uint16_t negTerminalInput, uint8_t outputFilterEnableAndDelayLevel, uint16_t invertedOutputPolarity)

- bool [COMP_E_initialize](#) (uint16_t baseAddress, COMP_E_initializeParam *param)
- void [COMP_E_interruptSetEdgeDirection](#) (uint16_t baseAddress, uint16_t edgeDirection)
- void [COMP_E_interruptToggleEdgeDirection](#) (uint16_t baseAddress)
- void [COMP_E_IOSwap](#) (uint16_t baseAddress)
- uint16_t [COMP_E_outputValue](#) (uint16_t baseAddress)
- void [COMP_E_setPowerMode](#) (uint16_t baseAddress, uint16_t powerMode)
- void [COMP_E_setReferenceAccuracy](#) (uint16_t baseAddress, uint16_t referenceAccuracy)
- void [COMP_E_setReferenceVoltage](#) (uint16_t baseAddress, uint16_t supplyVoltageReferenceBase, uint16_t lowerLimitSupplyVoltageFractionOf32, uint16_t upperLimitSupplyVoltageFractionOf32)
- void [COMP_E_shortInputs](#) (uint16_t baseAddress)
- void [COMP_E_unshortInputs](#) (uint16_t baseAddress)

7.2.1 Detailed Description

The COMP_E API is broken into three groups of functions: those that deal with initialization and output, those that handle interrupts, and those that handle auxiliary features of the COMP_E.

The COMP_E initialization and output functions are

- [COMP_E_init\(\)](#)
- [COMP_E_setReferenceVoltage\(\)](#)
- [COMP_E_enable\(\)](#)
- [COMP_E_disable\(\)](#)
- [COMP_E_outputValue\(\)](#)
- [COMP_E_setPowerMode\(\)](#)

The COMP_E interrupts are handled by

- [COMP_E_enableInterrupt\(\)](#)
- [COMP_E_disableInterrupt\(\)](#)
- [COMP_E_clearInterrupt\(\)](#)
- [COMP_E_getInterruptStatus\(\)](#)
- [COMP_E_interruptSetEdgeDirection\(\)](#)
- [COMP_E_interruptToggleEdgeDirection\(\)](#)

Auxiliary features of the COMP_E are handled by

- [COMP_E_enableShortOfInputs\(\)](#)
- [COMP_E_disableShortOfInputs\(\)](#)
- [COMP_E_disableInputBuffer\(\)](#)
- [COMP_E_enableInputBuffer\(\)](#)
- [COMP_E_IOSwap\(\)](#)
- [COMP_E_setReferenceAccuracy\(\)](#)
- [COMP_E_setPowerMode\(\)](#)

7.2.2 Function Documentation

7.2.2.1 void [COMP_E_clearInterrupt](#) (uint16_t *baseAddress*, uint16_t *interruptFlagMask*)

Clears COMP_E interrupt flags.

The COMP_E interrupt source is cleared, so that it no longer asserts. The highest interrupt flag is automatically cleared when an interrupt vector generator is used.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	12
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the COMP_E module.

interruptFlagMask Mask value is the logical OR of any of the following:

- **COMP_E_OUTPUT_INTERRUPT_FLAG** - Output interrupt flag
- **COMP_E_INTERRUPT_FLAG_INVERTED_POLARITY** - Output interrupt flag inverted polarity
- **COMP_E_INTERRUPT_FLAG_READY** - Ready interrupt flag

Returns:

None

7.2.2.2 void COMP_E_disable (uint16_t baseAddress)

Turns off the COMP_E module.

This function clears the CEON bit disabling the operation of the COMP_E module, saving from excess power consumption.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the COMP_E module.

Modified bits are **CEON** of **CECTL1** register.

Returns:

None

7.2.2.3 void COMP_E_disableInputBuffer (uint16_t baseAddress, uint16_t inputPort)

Disables the input buffer of the selected input port to effectively allow for analog signals.

This function sets the bit to disable the buffer for the specified input port to allow for analog signals from any of the COMP_E input pins. This bit is automatically set when the input is initialized to be used with the COMP_E module. This function should be used whenever an analog input is connected to one of these pins to prevent parasitic voltage from causing unexpected results.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	12
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the COMP_E module.

inputPort is the port in which the input buffer will be disabled. Mask value is the logical OR of any of the following:

- **COMP_E_INPUT0** [Default]
 - **COMP_E_INPUT1**
 - **COMP_E_INPUT2**
 - **COMP_E_INPUT3**
 - **COMP_E_INPUT4**
 - **COMP_E_INPUT5**
 - **COMP_E_INPUT6**
 - **COMP_E_INPUT7**
 - **COMP_E_INPUT8**
 - **COMP_E_INPUT9**
 - **COMP_E_INPUT10**
 - **COMP_E_INPUT11**
 - **COMP_E_INPUT12**
 - **COMP_E_INPUT13**
 - **COMP_E_INPUT14**
 - **COMP_E_INPUT15**
 - **COMP_E_VREF**
- Modified bits are **CEPDx** of **CECTL3** register.

Returns:

None

7.2.2.4 void COMP_E_disableInterrupt (uint16_t baseAddress, uint16_t interruptMask)

Disables selected COMP_E interrupt sources.

Disables the indicated COMP_E interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	12
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the COMP_E module.

interruptMask Mask value is the logical OR of any of the following:

- **COMP_E_OUTPUT_INTERRUPT** - Output interrupt
- **COMP_E_INVERTED_POLARITY_INTERRUPT** - Output interrupt inverted polarity
- **COMP_E_READY_INTERRUPT** - Ready interrupt

Returns:

None

7.2.2.5 void COMP_E_enable (uint16_t baseAddress)

Turns on the COMP_E module.

This function sets the bit that enables the operation of the COMP_E module.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the COMP_E module.

Returns:

None

7.2.2.6 void COMP_E_enableInputBuffer (uint16_t baseAddress, uint16_t inputPort)

Enables the input buffer of the selected input port to allow for digital signals.

This function clears the bit to enable the buffer for the specified input port to allow for digital signals from any of the COMP_E input pins. This should not be reset if there is an analog signal connected to the specified input pin to prevent from unexpected results.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	12
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the COMP_E module.

inputPort is the port in which the input buffer will be enabled. Mask value is the logical OR of any of the following:

- **COMP_E_INPUT0** [Default]
 - **COMP_E_INPUT1**
 - **COMP_E_INPUT2**
 - **COMP_E_INPUT3**
 - **COMP_E_INPUT4**
 - **COMP_E_INPUT5**
 - **COMP_E_INPUT6**
 - **COMP_E_INPUT7**
 - **COMP_E_INPUT8**
 - **COMP_E_INPUT9**
 - **COMP_E_INPUT10**
 - **COMP_E_INPUT11**
 - **COMP_E_INPUT12**
 - **COMP_E_INPUT13**
 - **COMP_E_INPUT14**
 - **COMP_E_INPUT15**
 - **COMP_E_VREF**
- Modified bits are **CEPDx** of **CECTL3** register.

Returns:
None

7.2.2.7 void COMP_E_enableInterrupt (uint16_t *baseAddress*, uint16_t *interruptMask*)

Enables selected COMP_E interrupt sources.

Enables the indicated COMP_E interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. **Does not clear interrupt flags.**

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	12
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the COMP_E module.

interruptMask Mask value is the logical OR of any of the following:

- **COMP_E_OUTPUT_INTERRUPT** - Output interrupt
- **COMP_E_INVERTED_POLARITY_INTERRUPT** - Output interrupt inverted polarity
- **COMP_E_READY_INTERRUPT** - Ready interrupt

Returns:
None

7.2.2.8 uint8_t COMP_E_getInterruptStatus (uint16_t *baseAddress*, uint16_t *interruptFlagMask*)

Gets the current COMP_E interrupt status.

This returns the interrupt status for the COMP_E module based on which flag is passed.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the COMP_E module.

interruptFlagMask Mask value is the logical OR of any of the following:

- **COMP_E_OUTPUT_INTERRUPT_FLAG** - Output interrupt flag
- **COMP_E_INTERRUPT_FLAG_INVERTED_POLARITY** - Output interrupt flag inverted polarity
- **COMP_E_INTERRUPT_FLAG_READY** - Ready interrupt flag

Returns:

Logical OR of any of the following:

- **COMP_E_OUTPUT_INTERRUPT_FLAG** Output interrupt flag
- **COMP_E_INTERRUPT_FLAG_INVERTED_POLARITY** Output interrupt flag inverted polarity
- **COMP_E_INTERRUPT_FLAG_READY** Ready interrupt flag
indicating the status of the masked flags

7.2.2.9 bool COMP_E_init (uint16_t *baseAddress*, uint16_t *posTerminalInput*, uint16_t *negTerminalInput*, uint8_t *outputFilterEnableAndDelayLevel*, uint16_t *invertedOutputPolarity*)

DEPRECATED - Initializes the COMP_E Module.

Upon successful initialization of the COMP_E module, this function will have reset all necessary register bits and set the given options in the registers. To actually use the COMP_E module, the [COMP_E_enable\(\)](#) function must be explicitly called before use. If a Reference Voltage is set to a terminal, the Voltage should be set using the [setReferenceVoltage\(\)](#) function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	82
TI Compiler 4.2.1	Size	54
TI Compiler 4.2.1	Speed	54
IAR 5.51.6	None	64
IAR 5.51.6	Size	54
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	34
MSPGCC 4.8.0	Speed	850

Parameters:

baseAddress is the base address of the COMP_E module.

posTerminalInput selects the input to the positive terminal. Valid values are:

- COMP_E_INPUT0 [Default]
- COMP_E_INPUT1
- COMP_E_INPUT2
- COMP_E_INPUT3
- COMP_E_INPUT4
- COMP_E_INPUT5
- COMP_E_INPUT6
- COMP_E_INPUT7
- COMP_E_INPUT8
- COMP_E_INPUT9
- COMP_E_INPUT10
- COMP_E_INPUT11
- COMP_E_INPUT12
- COMP_E_INPUT13
- COMP_E_INPUT14
- COMP_E_INPUT15
- COMP_E_VREF

Modified bits are **CEPDx** of **CECTL3** register; bits **CERSEL** of **CECTL2** register; bits **CEIPEN** and **CEIPSEL** of **CECTL0** register.

negTerminalInput selects the input to the negative terminal. Valid values are:

- COMP_E_INPUT0 [Default]
- COMP_E_INPUT1
- COMP_E_INPUT2
- COMP_E_INPUT3
- COMP_E_INPUT4
- COMP_E_INPUT5
- COMP_E_INPUT6
- COMP_E_INPUT7
- COMP_E_INPUT8
- COMP_E_INPUT9
- COMP_E_INPUT10
- COMP_E_INPUT11
- COMP_E_INPUT12
- COMP_E_INPUT13
- COMP_E_INPUT14
- COMP_E_INPUT15
- COMP_E_VREF

Modified bits are **CEPDx** of **CECTL3** register; bits **CERSEL** of **CECTL2** register; bits **CEIMSEL** of **CECTL0** register; bits **CEIMEN** of **CECTL0** register.

outputFilterEnableAndDelayLevel controls the output filter delay state, which is either off or enabled with a specified delay level. This parameter is device specific and delay levels should be found in the device's datasheet. Valid values are:

- COMP_E_FILTEROUTPUT_OFF [Default]
- COMP_E_FILTEROUTPUT_DLYLVL1
- COMP_E_FILTEROUTPUT_DLYLVL2
- COMP_E_FILTEROUTPUT_DLYLVL3
- COMP_E_FILTEROUTPUT_DLYLVL4

Modified bits are **CEFDLY** and **CEF** of **CECTL1** register.

invertedOutputPolarity controls if the output will be inverted or not Valid values are:

- COMP_E_NORMALOUTPUTPOLARITY - indicates the output should be normal
- COMP_E_INVERTEDOUTPUTPOLARITY - the output should be inverted

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the initialization process

7.2.2.10 bool COMP_E_initialize (uint16_t *baseAddress*, COMP_E_initializeParam * *param*)

Initializes the COMP_E Module.

Upon successful initialization of the COMP_E module, this function will have reset all necessary register bits and set the given options in the registers. To actually use the COMP_E module, the [COMP_E_enable\(\)](#) function must be explicitly called before use. If a Reference Voltage is set to a terminal, the Voltage should be set using the [setReferenceVoltage\(\)](#) function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	202
TI Compiler 4.2.1	Size	136
TI Compiler 4.2.1	Speed	136
IAR 5.51.6	None	188
IAR 5.51.6	Size	160
IAR 5.51.6	Speed	528
MSPGCC 4.8.0	None	324
MSPGCC 4.8.0	Size	178
MSPGCC 4.8.0	Speed	862

Parameters:

baseAddress is the base address of the COMP_E module.

param is the pointer to struct for initialization.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the initialization process

7.2.2.11 void COMP_E_interruptSetEdgeDirection (uint16_t *baseAddress*, uint16_t *edgeDirection*)

Explicitly sets the edge direction that would trigger an interrupt.

This function will set which direction the output will have to go, whether rising or falling, to generate an interrupt based on a non-inverted interrupt.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	44
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	28
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	70
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	20

Parameters:

baseAddress is the base address of the COMP_E module.

edgeDirection determines which direction the edge would have to go to generate an interrupt based on the non-inverted interrupt flag. Valid values are:

- **COMP_E_FALLINGEDGE** [Default] - sets the bit to generate an interrupt when the output of the COMP_E falls from HIGH to LOW if the normal interrupt bit is set (and LOW to HIGH if the inverted interrupt enable bit is set).

- **COMP_E_RISINGEDGE** - sets the bit to generate an interrupt when the output of the COMP_E rises from LOW to HIGH if the normal interrupt bit is set (and HIGH to LOW if the inverted interrupt enable bit is set). Modified bits are **CEIES** of **CECTL1** register.

Returns:
None

7.2.2.12 void COMP_E_interruptToggleEdgeDirection (uint16_t *baseAddress*)

Toggles the edge direction that would trigger an interrupt.

This function will toggle which direction the output will have to go, whether rising or falling, to generate an interrupt based on a non-inverted interrupt. If the direction was rising, it is now falling, if it was falling, it is now rising.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	10
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:
baseAddress is the base address of the COMP_E module.

Modified bits are **CEIES** of **CECTL1** register.

Returns:
None

7.2.2.13 void COMP_E_IOSwap (uint16_t *baseAddress*)

Toggles the bit that swaps which terminals the inputs go to, while also inverting the output of the COMP_E.

This function toggles the bit that controls which input goes to which terminal. After initialization, this bit is set to 0, after toggling it once the inputs are routed to the opposite terminal and the output is inverted.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:
baseAddress is the base address of the COMP_E module.

Returns:
None

7.2.2.14 uint16_t COMP_E_outputValue (uint16_t *baseAddress*)

Returns the output value of the COMP_E module.

Returns the output value of the COMP_E module.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the COMP_E module.

Returns:

One of the following:

- **COMP_E_LOW**
- **COMP_E_HIGH**
indicating the output value of the COMP_E module

7.2.2.15 void COMP_E_setPowerMode (uint16_t *baseAddress*, uint16_t *powerMode*)

Sets the power mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the COMP_E module.

powerMode decides the power mode Valid values are:

- **COMP_E_HIGH_SPEED_MODE**
- **COMP_E_NORMAL_MODE**
- **COMP_E_ULTRA_LOW_POWER_MODE**
Modified bits are **CEPWRMD** of **CECTL1** register.

Returns:
None

7.2.2.16 void COMP_E_setReferenceAccuracy (uint16_t *baseAddress*, uint16_t *referenceAccuracy*)

Sets the reference accuracy.

The reference accuracy is set to the desired setting. Clocked is better for low power operations but has a lower accuracy.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the COMP_E module.

referenceAccuracy is the reference accuracy setting of the COMP_E. Valid values are:

- **COMP_E_ACCURACY_STATIC**
 - **COMP_E_ACCURACY_CLOCKED** - for low power / low accuracy
- Modified bits are **CEREFACC** of **CECTL2** register.

Returns:

None

7.2.2.17 void COMP_E_setReferenceVoltage (uint16_t *baseAddress*, uint16_t *supplyVoltageReferenceBase*, uint16_t *lowerLimitSupplyVoltageFractionOf32*, uint16_t *upperLimitSupplyVoltageFractionOf32*)

Generates a Reference Voltage to the terminal selected during initialization.

Use this function to generate a voltage to serve as a reference to the terminal selected at initialization. The voltage is determined by the equation: $V_{base} * (Numerator / 32)$. If the upper and lower limit voltage numerators are equal, then a static reference is defined, whereas they are different then a hysteresis effect is generated.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	118
TI Compiler 4.2.1	Size	68
TI Compiler 4.2.1	Speed	66
IAR 5.51.6	None	98
IAR 5.51.6	Size	72
IAR 5.51.6	Speed	72
MSPGCC 4.8.0	None	192
MSPGCC 4.8.0	Size	68
MSPGCC 4.8.0	Speed	96

Parameters:

baseAddress is the base address of the COMP_E module.

supplyVoltageReferenceBase decides the source and max amount of Voltage that can be used as a reference.
Valid values are:

- **COMP_E_REFERENCE_AMPLIFIER_DISABLED**
- **COMP_E_VREFBASE1_2V**
- **COMP_E_VREFBASE2_0V**
- **COMP_E_VREFBASE2_5V**

Modified bits are **CEREFL** of **CECTL2** register.

lowerLimitSupplyVoltageFractionOf32 is the numerator of the equation to generate the reference voltage for the lower limit reference voltage.

Modified bits are **CEREF0** of **CECTL2** register.

upperLimitSupplyVoltageFractionOf32 is the numerator of the equation to generate the reference voltage for the upper limit reference voltage.

Modified bits are **CEREF1** of **CECTL2** register.

Returns:

None

7.2.2.18 void COMP_E_shortInputs (uint16_t baseAddress)

Shorts the two input pins chosen during initialization.

This function sets the bit that shorts the devices attached to the input pins chosen from the initialization of the COMP_E.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the COMP_E module.

Modified bits are **CESHORT** of **CECTL1** register.

Returns:

None

7.2.2.19 void COMP_E_unshortInputs (uint16_t baseAddress)

Disables the short of the two input pins chosen during initialization.

This function clears the bit that shorts the devices attached to the input pins chosen from the initialization of the COMP_E.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the COMP_E module.

Modified bits are **CESHORT** of **CECTL1** register.

Returns:

None

7.3 Programming Example

The following example shows how to initialize and use the COMP_E API to turn on an LED when the input to the positive terminal is higher than the input to the negative terminal.

```
// Initialize the Comparator E module
/* Base Address of Comparator E,
   Pin CD2 to Positive(+) Terminal,
   Reference Voltage to Negative(-) Terminal,
   Normal Power Mode,
   Output Filter On with minimal delay,
   Non-Inverted Output Polarity
*/
COMP_E_init(COMP_E_BASE,
            COMP_E_INPUT2,
            COMP_E_VREF,
            COMP_E_FILTEROUTPUT_OFF,
            COMP_E_NORMALOUTPUTPOLARITY
            );

// Set the reference voltage that is being supplied to the (-) terminal
/* Base Address of Comparator E,
   Reference Voltage of 2.0 V,
   Upper Limit of 2.0*(32/32) = 2.0V,
   Lower Limit of 2.0*(32/32) = 2.0V
   Static Accuracy
*/
COMP_E_setReferenceVoltage(COMP_E_BASE,
                           COMP_E_VREFBASE2_0V,
                           32,
                           32,
                           COMP_E_ACCURACY_STATIC
                           );

//Disable Input Buffer on P1.2/CD2
/* Base Address of Comparator E,
   Input Buffer port
   Selecting the CEx input pin to the comparator
   multiplexer with the CEx bits automatically
   disables output driver and input buffer for
   that pin, regardless of the state of the
```

```
associated CEPD.x bit
*/
COMP_E_disableInputBuffer (COMP_E_BASE,
    COMP_E_INPUT2);
// Allow power to Comparator module
COMP_E_enable (COMP_E_BASE);

__delay_cycles (400);          // delay for the reference to settle
```

8 Cyclical Redundancy Check (CRC)

Introduction	73
API Functions	73
Programming Example	78

8.1 Introduction

The Cyclical Redundancy Check (CRC) API provides a set of functions for using the MSP430Ware CRC module. Functions are provided to initialize the CRC and create a CRC signature to check the validity of data. This is mostly useful in the communication of data, or as a startup procedure to as a more complex and accurate check of data.

The CRC module offers no interrupts and is used only to generate CRC signatures to verify against pre-made CRC signatures (Checksums).

This driver is contained in `crc.c`, with `crc.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	146
TI Compiler 4.2.1	Size	46
TI Compiler 4.2.1	Speed	46
IAR 5.51.6	None	60
IAR 5.51.6	Size	30
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	168
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

8.2 API Functions

Functions

- `uint16_t CRC_getData (uint16_t baseAddress)`
- `uint16_t CRC_getResult (uint16_t baseAddress)`
- `uint16_t CRC_getResultBitsReversed (uint16_t baseAddress)`
- `void CRC_set16BitData (uint16_t baseAddress, uint16_t dataIn)`
- `void CRC_set16BitDataReversed (uint16_t baseAddress, uint16_t dataIn)`
- `void CRC_set8BitData (uint16_t baseAddress, uint8_t dataIn)`
- `void CRC_set8BitDataReversed (uint16_t baseAddress, uint8_t dataIn)`
- `void CRC_setSeed (uint16_t baseAddress, uint16_t seed)`

8.2.1 Detailed Description

The CRC API is one group that controls the CRC module. The APIs that are used to set the seed and data are

- `CRC_setSeed()`

- [CRC_set16BitData\(\)](#)
- [CRC_set8BitData\(\)](#)
- [CRC_set16BitDataReversed\(\)](#)
- [CRC_set8BitDataReversed\(\)](#)
- [CRC_setSeed\(\)](#)

The APIs that are used to get the data and results are

- [CRC_getData\(\)](#)
- [CRC_getResult\(\)](#)
- [CRC_getResultBitsReversed\(\)](#)

8.2.2 Function Documentation

8.2.2.1 uint16_t CRC_getData (uint16_t *baseAddress*)

Returns the value currently in the Data register.

This function returns the value currently in the data register. If set in byte bits reversed format, then the translated data would be returned.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	4
TI Compiler 4.2.1	Speed	4
IAR 5.51.6	None	4
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	14
MSPGCC 4.8.0	Size	4
MSPGCC 4.8.0	Speed	4

Parameters:

baseAddress is the base address of the CRC module.

Returns:

The value currently in the data register

8.2.2.2 uint16_t CRC_getResult (uint16_t *baseAddress*)

Returns the value of the Signature Result.

This function returns the value of the signature result generated by the CRC.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	16
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the CRC module.

Returns:

The value currently in the data register

8.2.2.3 uint16_t CRC_getResultBitsReversed (uint16_t baseAddress)

Returns the bit-wise reversed format of the Signature Result.

This function returns the bit-wise reversed format of the Signature Result.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the CRC module.

Returns:

The bit-wise reversed format of the Signature Result

8.2.2.4 void CRC_set16BitData (uint16_t baseAddress, uint16_t dataIn)

Sets the 16 bit data to add into the CRC module to generate a new signature.

This function sets the given data into the CRC module to generate the new signature from the current signature and new data.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the CRC module.

dataIn is the data to be added, through the CRC module, to the signature.

Modified bits are **CRCDI** of **CRCDI** register.

Returns:

None

8.2.2.5 void CRC_set16BitDataReversed (uint16_t *baseAddress*, uint16_t *dataIn*)

Translates the 16 bit data by reversing the bits in each byte and then sets this data to add into the CRC module to generate a new signature.

This function first reverses the bits in each byte of the data and then generates the new signature from the current signature and new translated data.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	10
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the CRC module.

dataIn is the data to be added, through the CRC module, to the signature.

Modified bits are **CRCDIRB** of **CRCDIRB** register.

Returns:

None

8.2.2.6 void CRC_set8BitData (uint16_t *baseAddress*, uint8_t *dataIn*)

Sets the 8 bit data to add into the CRC module to generate a new signature.

This function sets the given data into the CRC module to generate the new signature from the current signature and new data.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the CRC module.

dataIn is the data to be added, through the CRC module, to the signature.

Modified bits are **CRCDI** of **CRCDI** register.

Returns:

None

8.2.2.7 void CRC_set8BitDataReversed (uint16_t *baseAddress*, uint8_t *dataIn*)

Translates the 8 bit data by reversing the bits in each byte and then sets this data to add into the CRC module to generate a new signature.

This function first reverses the bits in each byte of the data and then generates the new signature from the current signature and new translated data.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	10
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the CRC module.

dataIn is the data to be added, through the CRC module, to the signature.

Modified bits are **CRCDIRB** of **CRCDIRB** register.

Returns:

None

8.2.2.8 void CRC_setSeed (uint16_t *baseAddress*, uint16_t *seed*)

Sets the seed for the CRC.

This function sets the seed for the CRC to begin generating a signature with the given seed and all passed data. Using this function resets the CRC signature.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	10
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the CRC module.

seed is the seed for the CRC to start generating a signature from.

Modified bits are **CRCINIRES** of **CRCINIRES** register.

Returns:

None

8.3 Programming Example

The following example shows how to initialize and use the CRC API to generate a CRC signature on an array of data.

```
unsigned int crcSeed = 0xBEEF;
unsigned int data[] = {0x0123,
                      0x4567,
                      0x8910,
                      0x1112,
                      0x1314};

unsigned int crcResult;
int i;

// Stop WDT
WDT_hold(WDT_A_BASE);

// Set P1.0 as an output
GPIO_setAsOutputPin(GPIO_PORT_P1,
                    GPIO_PIN0);

// Set the CRC seed
CRC_setSeed(CRC_BASE,
            crcSeed);

for (i = 0; i < 5; i++)
{
    //Add all of the values into the CRC signature
    CRC_set16BitData(CRC_BASE,
                    data[i]);
}

// Save the current CRC signature checksum to be compared for later
crcResult = CRC_getResult(CRC_BASE);
```

9 Cyclical Redundancy Check (CRC32)

Introduction	79
API Functions	79
Programming Example	82

9.1 Introduction

The Cyclic Redundancy Check (CRC) API provides a set of functions for using the MSP430Ware CRC module. Functions are provided to initialize the CRC and create a CRC signature to check the validity of data. This is mostly useful in the communication of data, or as a startup procedure to as a more complex and accurate check of data.

The CRC module offers no interrupts and is used only to generate CRC signatures to verify against pre-made CRC signatures (Checksums).

This driver is contained in `crc.c`, with `crc.h` containing the API definitions for use by applications.

9.2 API Functions

Functions

- `uint32_t CRC32_getResult (uint8_t crcMode)`
- `uint32_t CRC32_getResultReversed (uint8_t crcMode)`
- `void CRC32_set16BitData (uint16_t dataIn, uint8_t crcMode)`
- `void CRC32_set16BitDataReversed (uint16_t dataIn, uint8_t crcMode)`
- `void CRC32_set32BitData (uint32_t dataIn)`
- `void CRC32_set32BitDataReversed (uint32_t dataIn)`
- `void CRC32_set8BitData (uint8_t dataIn, uint8_t crcMode)`
- `void CRC32_set8BitDataReversed (uint8_t dataIn, uint8_t crcMode)`
- `void CRC32_setSeed (uint32_t seed, uint8_t crcMode)`

9.2.1 Detailed Description

The CRC32 API is one group that controls the CRC32 module.

- `CRC32_setSeed`
- `CRC32_set8BitData`
- `CRC32_set16BitData`
- `CRC32_set32BitData`
- `CRC32_set8BitDataReversed`
- `CRC32_set16BitDataReversed`
- `CRC32_set32BitDataReversed`
- `CRC32_getResult`
- `CRC32_getResultReversed`

9.2.2 Function Documentation

9.2.2.1 `uint32_t CRC32_getResult (uint8_t crcMode)`

Returns the value of CRC Signature Result.

Parameters:

crcMode selects between CRC32 and CRC16 Valid values are **CRC16_MODE** and **CRC32_MODE**

This function returns the value of the signature result generated by the CRC. Bit 0 is treated as LSB.

Returns:

uint32_t Result

9.2.2.2 uint32_t CRC32_getResultReversed (uint8_t crcMode)

Returns the bit-wise reversed format of the 32 bit Signature Result.

Parameters:

crcMode selects between CRC32 and CRC16 Valid values are **CRC16_MODE** and **CRC32_MODE**

This function returns the bit-wise reversed format of the Signature Result. Bit 0 is treated as MSB.

Returns:

uint32_t Result

9.2.2.3 void CRC32_set16BitData (uint16_t dataIn, uint8_t crcMode)

Sets the 16 Bit data to add into the CRC module to generate a new signature.

Parameters:

dataIn is the data to be added, through the CRC module, to the signature. Modified bits are **CRC16DIW0** of **CRC16DIW0** register. **CRC32DIW0** of **CRC32DIW0** register.

crcMode selects between CRC32 and CRC16 Valid values are **CRC16_MODE** and **CRC32_MODE**

This function sets the given data into the CRC module to generate the new signature from the current signature and new data. Bit 0 is treated as LSB

Returns:

NONE

9.2.2.4 void CRC32_set16BitDataReversed (uint16_t dataIn, uint8_t crcMode)

Translates the data by reversing the bits in each 16 bit data and then sets this data to add into the CRC module to generate a new signature.

Parameters:

dataIn is the data to be added, through the CRC module, to the signature. Modified bits are **CRC16DIRBW0** of **CRC16DIRBW0** register. **CRC32DIRBW0** of **CRC32DIRBW0** register.

crcMode selects between CRC32 and CRC16 Valid values are **CRC16_MODE** and **CRC32_MODE**

This function first reverses the bits in each byte of the data and then generates the new signature from the current signature and new translated data. Bit 0 is treated as MSB.

Returns:

NONE

9.2.2.5 void CRC32_set32BitData (uint32_t *dataIn*)

Sets the 32 Bit data to add into the CRC module to generate a new signature. Available only for CRC32_MODE and not for CRC16_MODE

Parameters:

dataIn is the data to be added, through the CRC module, to the signature. Modified bits are **CRC32DIL0** of **CRC32DIL0** register.

This function sets the given data into the CRC module to generate the new signature from the current signature and new data. Bit 0 is treated as LSB

Returns:

NONE

9.2.2.6 void CRC32_set32BitDataReversed (uint32_t *dataIn*)

Translates the data by reversing the bits in each 32 Bit Data and then sets this data to add into the CRC module to generate a new signature. Available only for CRC32 mode and not for CRC16 mode

Parameters:

dataIn is the data to be added, through the CRC module, to the signature. Modified bits are **CRC32DIRBL0** of **CRC32DIRBL0** register.

This function first reverses the bits in each byte of the data and then generates the new signature from the current signature and new translated data. Bit 0 is treated as MSB.

Returns:

NONE

9.2.2.7 void CRC32_set8BitData (uint8_t *dataIn*, uint8_t *crcMode*)

Sets the 8 Bit data to add into the CRC module to generate a new signature.

Parameters:

dataIn is the data to be added, through the CRC module, to the signature. Modified bits are **CRC16DIB0** of **CRC16DIB0** register. **CRC32DIB0** of **CRC32DIB0** register.

crcMode selects between CRC32 and CRC16 Valid values are **CRC16_MODE** and **CRC32_MODE**

This function sets the given data into the CRC module to generate the new signature from the current signature and new data. Bit 0 is treated as LSB.

Returns:

NONE

9.2.2.8 void CRC32_set8BitDataReversed (uint8_t *dataIn*, uint8_t *crcMode*)

Translates the data by reversing the bits in each 8 bit data and then sets this data to add into the CRC module to generate a new signature.

Parameters:

dataIn is the data to be added, through the CRC module, to the signature. Modified bits are **CRC16DIRBB0** of **CRC16DIRBB0** register. **CRC32DIRBB0** of **CRC32DIRBB0** register.

crcMode selects between CRC32 and CRC16 Valid values are **CRC16_MODE** and **CRC32_MODE**

This function first reverses the bits in each byte of the data and then generates the new signature from the current signature and new translated data. Bit 0 is treated as MSB.

Returns:

NONE

9.2.2.9 void CRC32_setSeed (uint32_t seed, uint8_t crcMode)

Sets the seed for the CRC.

Parameters:

seed is the seed for the CRC to start generating a signature from. Modified bits are **CRC16INIRESL0** of **CRC16INIRESL0** register. **CRC32INIRESL0** of **CRC32INIRESL0** register

crcMode selects between CRC32 and CRC16 Valid values are **CRC16_MODE** and **CRC32_MODE**

This function sets the seed for the CRC to begin generating a signature with the given seed and all passed data. Using this function resets the CRC32 signature.

Returns:

NONE

9.3 Programming Example

The following example shows how to initialize and use the CRC API to generate a CRC signature on an array of data

```
CRC32_setSeed(CRC16_INIT, CRC16_MODE);

for(ii=0;ii<9;ii++)
    CRC32_set8BitDataReversed(myData[ii], CRC16_MODE);

/* Getting the result from the hardware module */
hwCalculatedCRC = CRC32_getResult(CRC16_MODE);
```

10 Clock System (CS)

Introduction	83
API Functions	84
Programming Example	96

10.1 Introduction

The clock system module supports low system cost and low power consumption. Using three internal clock signals, the user can select the best balance of performance and low power consumption. The clock module can be configured to operate without any external components, with one or two external crystals, or with resonators, under full software control.

The clock system module includes the following clock sources:

- LFXTCLK - Low-frequency oscillator that can be used either with low-frequency 32768-Hz watch crystals, standard crystals, resonators, or external clock sources in the 50 kHz or below range. When in bypass mode, LFXTCLK can be driven with an external square wave signal.
- VLOCLK - Internal very-low-power low-frequency oscillator with 10-kHz typical frequency
- DCOCLK - Internal digitally controlled oscillator (DCO) with selectable frequencies
- MODCLK - Internal low-power oscillator with 5-MHz typical frequency. LFMODCLK is MODCLK divided by 128.
- HFXTCLK - High-frequency oscillator that can be used with standard crystals or resonators in the 4-MHz to 24-MHz range. When in bypass mode, HFXTCLK can be driven with an external square wave signal.

Four system clock signals are available from the clock module:

- ACLK - Auxiliary clock. The ACLK is software selectable as LFXTCLK, VLOCLK, or LFMODCLK. ACLK can be divided by 1, 2, 4, 8, 16, or 32. ACLK is software selectable by individual peripheral modules.
- MCLK - Master clock. MCLK is software selectable as LFXTCLK, VLOCLK, LFMODCLK, DCOCLK, MODCLK, or HFXTCLK. MCLK can be divided by 1, 2, 4, 8, 16, or 32. MCLK is used by the CPU and system.
- SMCLK - Sub-system master clock. SMCLK is software selectable as LFXTCLK, VLOCLK, LFMODCLK, DCOCLK, MODCLK, or HFXTCLK. SMCLK is software selectable by individual peripheral modules.
- MODCLK - Module clock. MODCLK may also be used by various peripheral modules and is sourced by MODOSC.
- VLOCLK - VLO clock. VLOCLK may also be used directly by various peripheral modules and is sourced by VLO.

Fail-Safe logic The crystal oscillator faults are set if the corresponding crystal oscillator is turned on and not operating properly. Once set, the fault bits remain set until reset in software, regardless if the fault condition no longer exists. If the user clears the fault bits and the fault condition still exists, the fault bits are automatically set, otherwise they remain cleared.

The OFIFG oscillator-fault interrupt flag is set and latched at POR or when any oscillator fault is detected. When OFIFG is set and OFIE is set, the OFIFG requests a user NMI. When the interrupt is granted, the OFIE is not reset automatically as it is in previous MSP430 families. It is no longer required to reset the OFIE. NMI entry/exit circuitry removes this requirement. The OFIFG flag must be cleared by software. The source of the fault can be identified by checking the individual fault bits.

If LFXT is sourcing any system clock (ACLK, MCLK, or SMCLK) and a fault is detected, the system clock is automatically switched to LFMODCLK for its clock source. The LFXT fault logic works in all power modes, including LPM3.5.

If HFXT is sourcing MCLK or SMCLK, and a fault is detected, the system clock is automatically switched to MODCLK for its clock source. By default, the HFXT fault logic works in all power modes, except LPM3.5 or LPM4.5, because high-frequency operation in these modes is not supported.

The fail-safe logic does not change the respective SELA, SELM, and SELS bit settings. The fail-safe mechanism behaves the same in normal and bypass modes.

This driver is contained in `cs_a.c`, with `cs_a.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	1500
TI Compiler 4.2.1	Size	1170
TI Compiler 4.2.1	Speed	1188
IAR 5.51.6	None	1260
IAR 5.51.6	Size	718
IAR 5.51.6	Speed	1168
MSPGCC 4.8.0	None	2842
MSPGCC 4.8.0	Size	1334
MSPGCC 4.8.0	Speed	3308

10.2 API Functions

Functions

- void [CS_bypassHFXT](#) (void)
- bool [CS_bypassHFXTWithTimeout](#) (uint32_t timeout)
- void [CS_bypassLFXT](#) (void)
- bool [CS_bypassLFXTWithTimeout](#) (uint32_t timeout)
- uint16_t [CS_clearAllOscFlagsWithTimeout](#) (uint32_t timeout)
- void [CS_clearFaultFlag](#) (uint8_t mask)
- void [CS_clockSignalInit](#) (uint8_t selectedClockSignal, uint16_t clockSource, uint16_t clockSourceDivider)
- void [CS_disableClockRequest](#) (uint8_t selectClock)
- void [CS_enableClockRequest](#) (uint8_t selectClock)
- uint8_t [CS_faultFlagStatus](#) (uint8_t mask)
- uint32_t [CS_getACLK](#) (void)
- uint32_t [CS_getMCLK](#) (void)
- uint32_t [CS_getSMCLK](#) (void)
- void [CS_HFXTOff](#) (void)
- void [CS_HFXTStart](#) (uint16_t hfxtdrive)
- bool [CS_HFXTStartWithTimeout](#) (uint16_t hfxtdrive, uint32_t timeout)
- void [CS_LFXTOff](#) (void)
- void [CS_LFXTStart](#) (uint16_t lfxtdrive)
- bool [CS_LFXTStartWithTimeout](#) (uint16_t lfxtdrive, uint32_t timeout)
- void [CS_setDCOFreq](#) (uint16_t dcofsel, uint16_t dcofsel)
- void [CS_setExternalClockSource](#) (uint32_t LFXTCLK_frequency, uint32_t HFXTCLK_frequency)
- void [CS_VLOff](#) (void)

10.2.1 Detailed Description

The CS API is broken into four groups of functions: an API that initializes the clock module, those that deal with clock configuration and control, and external crystal and bypass specific configuration and initialization, and those that handle interrupts.

General CS configuration and initialization are handled by the following API

- [CS_clockSignalInit](#)()
- [CS_enableClockRequest](#)()
- [CS_disableClockRequest](#)()
- [CS_getACLK](#)()
- [CS_getSMCLK](#)()
- [CS_getMCLK](#)()
- [CS_setDCOFreq](#)()

The following external crystal and bypass specific configuration and initialization functions are available

- `CS_LFXTStart()`
- `CS_bypassLFXT()`
- `CS_bypassLFXTWithTimeout()`
- `CS_LFXTStartWithTimeout()`
- `CS_LFXTOff()`
- `CS_HFXTStart()`
- `CS_bypassHFXT()`
- `CS_HFXTStartWithTimeout()`
- `CS_bypassHFXTWithTimeout()`
- `CS_HFXTOff()`
- `CS_VLOoff()`

The CS interrupts are handled by

- `CS_enableClockRequest()`
- `CS_disableClockRequest()`
- `CS_faultFlagStatus()`
- `CS_clearFaultFlag()`
- `CS_clearAllOscFlagsWithTimeout()`

`CS_setExternalClockSource` must be called if an external crystal LFXT or HFXT is used and the user intends to call `CS_getMCLK`, `CS_getSMCLK` or `CS_getACLK` APIs and `HFXTStart`, `HFXTByPass`, `HFXTStartWithTimeout`, `HFXTByPassWithTimeout`. If not any of the previous API are going to be called, it is not necessary to invoke this API.

10.2.2 Function Documentation

10.2.2.1 void CS_bypassHFXT (void)

Bypasses the HFXT crystal oscillator.

Bypasses the HFXT crystal oscillator, which supports crystal frequencies between 0 MHz and 24 MHz. Loops until all oscillator fault flags are cleared, with no timeout. NOTE: User must call `CS_setExternalClockSource` to set frequency of external clocks before calling this function.

Modified bits of **CSCTL5** register, bits of **CSCTL4** register and bits of **SFRIFG** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	116
TI Compiler 4.2.1	Size	116
TI Compiler 4.2.1	Speed	116
IAR 5.51.6	None	116
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	120
MSPGCC 4.8.0	None	244
MSPGCC 4.8.0	Size	142
MSPGCC 4.8.0	Speed	150

Returns:

None

10.2.2.2 bool CS_bypassHFXTWithTimeout (uint32_t *timeout*)

Bypasses the HFXT crystal oscillator with timeout.

Bypasses the HFXT crystal oscillator, which supports crystal frequencies between 0 MHz and 24 MHz. Loops until all oscillator fault flags are cleared or until a timeout counter is decremented and equals to zero. NOTE: User must call CS_setExternalClockSource to set frequency of external clocks before calling this function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	162
TI Compiler 4.2.1	Size	140
TI Compiler 4.2.1	Speed	140
IAR 5.51.6	None	146
IAR 5.51.6	Size	96
IAR 5.51.6	Speed	134
MSPGCC 4.8.0	None	290
MSPGCC 4.8.0	Size	170
MSPGCC 4.8.0	Speed	230

Parameters:

timeout is the count value that gets decremented every time the loop that clears oscillator fault flags gets executed.

Modified bits of **CSCTL5** register, bits of **CSCTL4** register and bits of **SFRIFG1** register.

Returns:

STATUS_SUCCESS or STATUS_FAIL

10.2.2.3 void CS_bypassLFXT (void)

Bypasses the LFXT crystal oscillator.

Bypasses the LFXT crystal oscillator. Loops until all oscillator fault flags are cleared, with no timeout. IMPORTANT: User must call CS_setExternalClockSource function to set frequency of external clocks before calling this function.

Modified bits of **CSCTL0** register, bits of **CSCTL5** register, bits of **CSCTL4** register and bits of **SFRIFG** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	34
TI Compiler 4.2.1	Speed	34
IAR 5.51.6	None	34
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	94
MSPGCC 4.8.0	Size	40
MSPGCC 4.8.0	Speed	38

Returns:

None

10.2.2.4 bool CS_bypassLFXTWithTimeout (uint32_t *timeout*)

Bypass the LFXT crystal oscillator with timeout.

Bypasses the LFXT crystal oscillator with timeout. Loops until all oscillator fault flags are cleared or until a timeout counter is decremented and equals to zero. NOTE: User must call CS_setExternalClockSource to set frequency of external clocks before calling this function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	80
TI Compiler 4.2.1	Size	58
TI Compiler 4.2.1	Speed	58
IAR 5.51.6	None	64
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	60
MSPGCC 4.8.0	None	144
MSPGCC 4.8.0	Size	64
MSPGCC 4.8.0	Speed	80

Parameters:

timeout is the count value that gets decremented every time the loop that clears oscillator fault flags gets executed.

Modified bits of **CSCTL0** register, bits of **CSCTL5** register, bits of **CSCTL4** register and bits of **SFRIFG** register.

Returns:

STATUS_SUCCESS or STATUS_FAIL

10.2.2.5 uint16_t CS_clearAllOscFlagsWithTimeout (uint32_t *timeout*)

Clears all the Oscillator Flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	64
TI Compiler 4.2.1	Size	46
TI Compiler 4.2.1	Speed	46
IAR 5.51.6	None	54
IAR 5.51.6	Size	40
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	114
MSPGCC 4.8.0	Size	52
MSPGCC 4.8.0	Speed	52

Parameters:

timeout is the count value that gets decremented every time the loop that clears oscillator fault flags gets executed.

Modified bits of **CSCTL5** register and bits of **SFRIFG1** register.

Returns:

the mask of the oscillator flag status

10.2.2.6 void CS_clearFaultFlag (uint8_t *mask*)

Clears the current CS fault flag status for the masked bit.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	16
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

mask is the masked interrupt flag status to be returned. mask parameter can be any one of the following Mask value is the logical OR of any of the following:

- **CS_LFXTOFFG** - LFXT oscillator fault flag
- **CS_HFXTOFFG** - HFXT oscillator fault flag

Modified bits of **CSCTL5** register.

Returns:

None

10.2.2.7 void CS_clockSignalInit (uint8_t *selectedClockSignal*, uint16_t *clockSource*, uint16_t *clockSourceDivider*)

Initializes clock signal.

This function initializes each of the clock signals. The user must ensure that this function is called for each clock signal. If not, the default state is assumed for the particular clock signal. Refer to MSP430ware documentation for CS module or Device Family User's Guide for details of default clock signal states.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	154
TI Compiler 4.2.1	Size	94
TI Compiler 4.2.1	Speed	102
IAR 5.51.6	None	108
IAR 5.51.6	Size	88
IAR 5.51.6	Speed	90
MSPGCC 4.8.0	None	320
MSPGCC 4.8.0	Size	100
MSPGCC 4.8.0	Speed	118

Parameters:

selectedClockSignal Selected clock signal Valid values are:

- **CS_ACLK**
- **CS_MCLK**
- **CS_SMCLK**
- **CS_MODOSC**

clockSource is the selected clock signal Valid values are:

- **CS_VLOCLK_SELECT**
- **CS_DCOCLK_SELECT** - [Not available for ACLK]
- **CS_LFXTCLK_SELECT**
- **CS_HFXTCLK_SELECT** - [Not available for ACLK]
- **CS_LFMODEOSC_SELECT**
- **CS_MODEOSC_SELECT** - [Not available for ACLK]

clockSourceDivider is the selected clock divider to calculate clock signal from clock source. Valid values are:

- **CS_CLOCK_DIVIDER_1** - [Default for ACLK]
- **CS_CLOCK_DIVIDER_2**
- **CS_CLOCK_DIVIDER_4**
- **CS_CLOCK_DIVIDER_8** - [Default for SMCLK and MCLK]
- **CS_CLOCK_DIVIDER_16**
- **CS_CLOCK_DIVIDER_32**

Modified bits of **CSCTL0** register, bits of **CSCTL3** register and bits of **CSCTL2** register.

Returns:

None

10.2.2.8 void CS_disableClockRequest (uint8_t *selectClock*)

Disables conditional module requests.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	16
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

selectClock selects specific request enables. Valid values are:

- **CS_ACLK**
- **CS_MCLK**
- **CS_SMCLK**
- **CS_MODEOSC**

Modified bits of **CSCTL6** register.

Returns:

None

10.2.2.9 void CS_enableClockRequest (uint8_t *selectClock*)

Enables conditional module requests.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	16
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	50
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

selectClock selects specific request enables. Valid values are:

- **CS_ACLK**
- **CS_MCLK**
- **CS_SMCLK**
- **CS_MODOSC**

Modified bits of **CSCTL6** register.

Returns:

None

10.2.2.10 uint8_t CS_faultFlagStatus (uint8_t *mask*)

Gets the current CS fault flag status.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

mask is the masked interrupt flag status to be returned. Mask parameter can be either any of the following selection.

Mask value is the logical OR of any of the following:

- **CS_LFXTOFFG** - LFXT oscillator fault flag
- **CS_HFXTOFFG** - HFXT oscillator fault flag

Returns:

Logical OR of any of the following:

- **CS_LFXTOFFG** LFXT oscillator fault flag
- **CS_HFXTOFFG** HFXT oscillator fault flag
indicating the status of the masked interrupts

10.2.2.11 uint32_t CS_getACLK (void)

Get the current ACLK frequency.

If a oscillator fault is set, the frequency returned will be based on the fail safe mechanism of CS module. The user of this API must ensure that CS_externalClockSourceInit API was invoked before in case LFXT or HFXT is being used.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	50
TI Compiler 4.2.1	Size	28
TI Compiler 4.2.1	Speed	28
IAR 5.51.6	None	42
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	70
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	640

Returns:

Current ACLK frequency in Hz

10.2.2.12 uint32_t CS_getMCLK (void)

Get the current MCLK frequency.

If a oscillator fault is set, the frequency returned will be based on the fail safe mechanism of CS module. The user of this API must ensure that CS_externalClockSourceInit API was invoked before in case LFXT or HFXT is being used.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	30
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	632

Returns:

Current MCLK frequency in Hz

10.2.2.13 uint32_t CS_getSMCLK (void)

Get the current SMCLK frequency.

If a oscillator fault is set, the frequency returned will be based on the fail safe mechanism of CS module. The user of this API must ensure that CS_externalClockSourceInit API was invoked before in case LFXT or HFXT is being used.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	46
TI Compiler 4.2.1	Size	24
TI Compiler 4.2.1	Speed	24
IAR 5.51.6	None	36
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	70
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	640

Returns:

Current SMCLK frequency in Hz

10.2.2.14 void CS_HFXTOff (void)

Stops the HFXT oscillator using the HFXTOFF bit.

Modified bits of **CSCTL4** register.**Code Metrics:**

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Returns:

None

10.2.2.15 void CS_HFXTStart (uint16_t *hfxtdrive*)

Starts the HFXT crystal.

Initializes the HFXT crystal oscillator, which supports crystal frequencies between 0 MHz and 24 MHz, depending on the selected drive strength. Loops until all oscillator fault flags are cleared, with no timeout. See the device-specific data sheet for appropriate drive settings. NOTE: User must call CS_setExternalClockSource to set frequency of external clocks before calling this function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	144
TI Compiler 4.2.1	Size	136
TI Compiler 4.2.1	Speed	138
IAR 5.51.6	None	136
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	142
MSPGCC 4.8.0	None	290
MSPGCC 4.8.0	Size	162
MSPGCC 4.8.0	Speed	170

Parameters:

hfxtdrive is the target drive strength for the HFXT crystal oscillator. Valid values are:

- **CS_HFXTDRIVE_4MHZ_8MHZ**
- **CS_HFXTDRIVE_8MHZ_16MHZ**
- **CS_HFXTDRIVE_16MHZ_24MHZ**
- **CS_HFXTDRIVE_24MHZ_32MHZ** [Default]

Modified bits of **CSCTL5** register, bits of **CSCTL4** register and bits of **SFRIFG1** register.

Returns:

None

10.2.2.16 bool CS_HFXTStartWithTimeout (uint16_t hfxtdrive, uint32_t timeout)

Initializes the HFXT crystal oscillator with timeout.

Initializes the HFXT crystal oscillator, which supports crystal frequencies between 0 MHz and 24 MHz, depending on the selected drive strength. Loops until all oscillator fault flags are cleared or until a timeout counter is decremented and equals to zero. See the device-specific data sheet for appropriate drive settings. NOTE: User must call CS_setExternalClockSource to set frequency of external clocks before calling this function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	194
TI Compiler 4.2.1	Size	162
TI Compiler 4.2.1	Speed	164
IAR 5.51.6	None	168
IAR 5.51.6	Size	134
IAR 5.51.6	Speed	162
MSPGCC 4.8.0	None	350
MSPGCC 4.8.0	Size	196
MSPGCC 4.8.0	Speed	228

Parameters:

hfxtdrive is the target drive strength for the HFXT crystal oscillator. Valid values are:

- **CS_HFXTDRIVE_4MHZ_8MHZ**
- **CS_HFXTDRIVE_8MHZ_16MHZ**
- **CS_HFXTDRIVE_16MHZ_24MHZ**
- **CS_HFXTDRIVE_24MHZ_32MHZ** [Default]

timeout is the count value that gets decremented every time the loop that clears oscillator fault flags gets executed.

Modified bits of **CSCTL5** register, bits of **CSCTL4** register and bits of **SFRIFG1** register.

Returns:

STATUS_SUCCESS or STATUS_FAIL

10.2.2.17 void CS_LFXTOff (void)

Stops the LFXT oscillator using the LFXTOFF bit.

Modified bits of **CSCTL4** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	16
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Returns:

None

10.2.2.18 void CS_LFXTStart (uint16_t *lfxtdrive*)

Initializes the LFXT crystal in low frequency mode.

Initializes the LFXT crystal oscillator in low frequency mode. Loops until all oscillator fault flags are cleared, with no timeout. See the device- specific data sheet for appropriate drive settings. **IMPORTANT:** User must call CS_setExternalClockSource function to set frequency of external clocks before calling this function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	80
TI Compiler 4.2.1	Size	72
TI Compiler 4.2.1	Speed	74
IAR 5.51.6	None	72
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	86
MSPGCC 4.8.0	None	172
MSPGCC 4.8.0	Size	78
MSPGCC 4.8.0	Speed	76

Parameters:

lfxtdrive is the target drive strength for the LFXT crystal oscillator. Valid values are:

- CS_LFXT_DRIVE0
- CS_LFXT_DRIVE1
- CS_LFXT_DRIVE2
- CS_LFXT_DRIVE3 [Default]

Modified bits of **CSCTL0** register, bits of **CSCTL5** register, bits of **CSCTL4** register and bits of **SFRIFG1** register.

Returns:

None

10.2.2.19 bool CS_LFXTStartWithTimeout (uint16_t *lfxtdrive*, uint32_t *timeout*)

Initializes the LFXT crystal oscillator in low frequency mode with timeout.

Initializes the LFXT crystal oscillator in low frequency mode with timeout. Loops until all oscillator fault flags are cleared or until a timeout counter is decremented and equals to zero. See the device-specific datasheet for appropriate drive settings. **IMPORTANT:** User must call CS_setExternalClockSource to set frequency of external clocks before calling this function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	130
TI Compiler 4.2.1	Size	98
TI Compiler 4.2.1	Speed	100
IAR 5.51.6	None	104
IAR 5.51.6	Size	72
IAR 5.51.6	Speed	102
MSPGCC 4.8.0	None	232
MSPGCC 4.8.0	Size	108
MSPGCC 4.8.0	Speed	114

Parameters:

lfxtdrive is the target drive strength for the LFXT crystal oscillator. Valid values are:

- **CS_LFXT_DRIVE0**
- **CS_LFXT_DRIVE1**
- **CS_LFXT_DRIVE2**
- **CS_LFXT_DRIVE3** [Default]

timeout is the count value that gets decremented every time the loop that clears oscillator fault flags gets executed.

Modified bits of **CSCTL0** register, bits of **CSCTL5** register, bits of **CSCTL4** register and bits of **SFRIFG1** register.

Returns:

STATUS_SUCCESS or STATUS_FAIL indicating if the LFXT crystal oscillator was initialized successfully

10.2.2.20 void CS_setDCOFreq (uint16_t *dcorsel*, uint16_t *dcofsel*)

Set DCO frequency.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	20
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:

dcorsel selects frequency range option. Valid values are:

- **CS_DCORSEL_0** [Default] - Low Frequency Option
- **CS_DCORSEL_1** - High Frequency Option

dcofsel selects valid frequency options based on dco frequency range selection (dcorsel) Valid values are:

- **CS_DCOFSEL_0** - Low frequency option 1MHz. High frequency option 1MHz.
- **CS_DCOFSEL_1** - Low frequency option 2.67MHz. High frequency option 5.33MHz.
- **CS_DCOFSEL_2** - Low frequency option 3.33MHz. High frequency option 6.67MHz.
- **CS_DCOFSEL_3** - Low frequency option 4MHz. High frequency option 8MHz.
- **CS_DCOFSEL_4** - Low frequency option 5.33MHz. High frequency option 16MHz.
- **CS_DCOFSEL_5** - Low frequency option 6.67MHz. High frequency option 20MHz.
- **CS_DCOFSEL_6** - Low frequency option 8MHz. High frequency option 24MHz.

Returns:

None

10.2.2.21 void CS_setExternalClockSource (uint32_t *LFXTCLK_frequency*, uint32_t *HFXTCLK_frequency*)

Sets the external clock source.

This function sets the external clock sources LFXT and HFXT crystal oscillator frequency values. This function must be called if an external crystal LFXT or HFXT is used and the user intends to call CS_getMCLK, CS_getSMCLK, CS_getACLK and CS_LFXTStart, CS_LFXTByPass, CS_LFXTStartWithTimeout, CS_LFXTByPassWithTimeout, CS_HFXTStart, CS_HFXTByPass, CS_HFXTStartWithTimeout, CS_HFXTByPassWithTimeout.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	44
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:

LFXTCLK_frequency is the LFXT crystal frequencies in Hz

HFXTCLK_frequency is the HFXT crystal frequencies in Hz

Returns:

None

10.2.2.22 void CS_VLOoff (void)

Turns off VLO.

Modified bits of **CSCTL4** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	16
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Returns:

None

10.3 Programming Example

The following example shows the configuration of the CS module that sets SMCLK = MCLK = 8MHz

```
//Set DCO Frequency to 8MHz
CS_setDCOFreq(CS_BASE,CS_DCORSEL_0,CS_DCOFSEL_6);

//configure MCLK, SMCLK to be source by DCOCLK
CS_clockSignalInit(CS_BASE,CS_SMCLK,CS_DCOCLK_SELECT,CS_CLOCK_DIVIDER_1);
CS_clockSignalInit(CS_BASE,CS_MCLK,CS_DCOCLK_SELECT,CS_CLOCK_DIVIDER_1);
```

11 Direct Memory Access (DMA)

Introduction	98
API Functions	98
Programming Example	111

11.1 Introduction

The Direct Memory Access (DMA) API provides a set of functions for using the MSP430Ware DMA modules. Functions are provided to initialize and setup each DMA channel with the source and destination addresses, manage the interrupts for each channel, and set bits that affect all DMA channels.

The DMA module provides the ability to move data from one address in the device to another, and that includes other peripheral addresses to RAM or vice-versa, all without the actual use of the CPU. Please be advised, that the DMA module does halt the CPU for 2 cycles while transferring, but does not have to edit any registers or anything. The DMA can transfer by bytes or words at a time, and will automatically increment or decrement the source or destination address if desired. There are also 6 different modes to transfer by, including single-transfer, block-transfer, and burst-block-transfer, as well as repeated versions of those three different kinds which allows transfers to be repeated without having re-enable transfers.

The DMA settings that affect all DMA channels include prioritization, from a fixed priority to dynamic round-robin priority. Another setting that can be changed is when transfers occur, the CPU may be in a read-modify-write operation which can be disastrous to time sensitive material, so this can be disabled. And Non-Maskable-Interrupts can indeed be maskable to the DMA module if not enabled.

The DMA module can generate one interrupt per channel. The interrupt is only asserted when the specified amount of transfers has been completed. With single-transfer, this occurs when that many single transfers have occurred, while with block or burst-block transfers, once the block is completely transferred the interrupt is asserted.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	612
TI Compiler 4.2.1	Size	370
TI Compiler 4.2.1	Speed	376
IAR 5.51.6	None	466
IAR 5.51.6	Size	362
IAR 5.51.6	Speed	382
MSPGCC 4.8.0	None	980
MSPGCC 4.8.0	Size	406
MSPGCC 4.8.0	Speed	430

11.2 API Functions

Functions

- void [DMA_clearInterrupt](#) (uint8_t channelSelect)
- void [DMA_clearNMIAbort](#) (uint8_t channelSelect)
- void [DMA_disableInterrupt](#) (uint8_t channelSelect)
- void [DMA_disableNMIAbort](#) (void)
- void [DMA_disableRoundRobinPriority](#) (void)
- void [DMA_disableTransferDuringReadModifyWrite](#) (void)

- void [DMA_disableTransfers](#) (uint8_t channelSelect)
- void [DMA_enableInterrupt](#) (uint8_t channelSelect)
- void [DMA_enableNMIAbort](#) (void)
- void [DMA_enableRoundRobinPriority](#) (void)
- void [DMA_enableTransferDuringReadModifyWrite](#) (void)
- void [DMA_enableTransfers](#) (uint8_t channelSelect)
- uint16_t [DMA_getInterruptStatus](#) (uint8_t channelSelect)
- bool [DMA_init](#) (uint8_t channelSelect, uint16_t transferModeSelect, uint16_t transferSize, uint8_t triggerSourceSelect, uint8_t transferUnitSelect, uint8_t triggerTypeSelect)
- bool [DMA_initialize](#) (DMA_initializeParam *param)
- uint16_t [DMA_NMIAbortStatus](#) (uint8_t channelSelect)
- void [DMA_setDstAddress](#) (uint8_t channelSelect, uint32_t dstAddress, uint16_t directionSelect)
- void [DMA_setSrcAddress](#) (uint8_t channelSelect, uint32_t srcAddress, uint16_t directionSelect)
- void [DMA_setTransferSize](#) (uint8_t channelSelect, uint16_t transferSize)
- void [DMA_startTransfer](#) (uint8_t channelSelect)

11.2.1 Detailed Description

The DMA API is broken into three groups of functions: those that deal with initialization and transfers, those that handle interrupts, and those that affect all DMA channels.

The DMA initialization and transfer functions are: [DMA_init\(\)](#) [DMA_setSrcAddress\(\)](#) [DMA_setDstAddress\(\)](#) [DMA_enableTransfers\(\)](#) [DMA_disableTransfers\(\)](#) [DMA_startTransfer\(\)](#) [DMA_setTransferSize\(\)](#)

The DMA interrupts are handled by: [DMA_enableInterrupt\(\)](#) [DMA_disableInterrupt\(\)](#) [DMA_getInterruptStatus\(\)](#) [DMA_clearInterrupt\(\)](#) [DMA_NMIAbortStatus\(\)](#) [DMA_clearNMIAbort\(\)](#)

Features of the DMA that affect all channels are handled by: [DMA_disableTransferDuringReadModifyWrite\(\)](#) [DMA_enableTransferDuringReadModifyWrite\(\)](#) [DMA_enableRoundRobinPriority\(\)](#) [DMA_disableRoundRobinPriority\(\)](#) [DMA_enableNMIAbort\(\)](#) [DMA_disableNMIAbort\(\)](#)

11.2.2 Function Documentation

11.2.2.1 void DMA_clearInterrupt (uint8_t *channelSelect*)

Clears the interrupt flag for the selected channel.

This function clears the DMA interrupt flag is cleared, so that it no longer asserts.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

channelSelect is the specified channel to clear the interrupt flag for. Valid values are:

- [DMA_CHANNEL_0](#)
- [DMA_CHANNEL_1](#)
- [DMA_CHANNEL_2](#)

- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

Returns:
None

11.2.2.2 void DMA_clearNMIAbort (uint8_t *channelSelect*)

Clears the status of the NMIAbort to proceed with transfers for the selected channel.

This function clears the status of the NMI Abort flag for the selected channel to allow for transfers on the channel to continue.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

channelSelect is the specified channel to clear the NMI Abort flag for. Valid values are:

- DMA_CHANNEL_0
- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

Returns:
None

11.2.2.3 void DMA_disableInterrupt (uint8_t *channelSelect*)

Disables the DMA interrupt for the selected channel.

Disables the DMA interrupt source. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

channelSelect is the specified channel to disable the interrupt for. Valid values are:

- DMA_CHANNEL_0
- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

Returns:

None

11.2.2.4 void DMA_disableNMIAbort (void)

Disables any NMI from interrupting a DMA transfer.

This function disables NMI's from interrupting any DMA transfer currently in progress.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

11.2.2.5 void DMA_disableRoundRobinPriority (void)

Disables Round Robin prioritization.

This function disables Round Robin Prioritization, enabling static prioritization of the DMA channels. In static prioritization, the DMA channels are prioritized with the lowest DMA channel index having the highest priority (i.e. DMA Channel 0 has the highest priority).

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

11.2.2.6 void DMA_disableTransferDuringReadModifyWrite (void)

Disables the DMA from stopping the CPU during a Read-Modify-Write Operation to start a transfer.

This function allows the CPU to finish any read-modify-write operations it may be in the middle of before transfers of and DMA channel stop the CPU.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

11.2.2.7 void DMA_disableTransfers (uint8_t channelSelect)

Disables transfers from being triggered.

This function disables transfer from being triggered for the selected channel. This function should be called before any re-initialization of the selected DMA channel.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	14
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

channelSelect is the specified channel to disable transfers for. Valid values are:

- DMA_CHANNEL_0
- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

Returns:

None

11.2.2.8 void DMA_enableInterrupt (uint8_t *channelSelect*)

Enables the DMA interrupt for the selected channel.

Enables the DMA interrupt source. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. Does not clear interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

channelSelect is the specified channel to enable the interrupt for. Valid values are:

- DMA_CHANNEL_0
- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

Returns:

None

11.2.2.9 void DMA_enableNMIAbort (void)

Enables a NMI to interrupt a DMA transfer.

This function allow NMI's to interrupting any DMA transfer currently in progress and stops any future transfers to begin before the NMI is done processing.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

11.2.2.10 void DMA_enableRoundRobinPriority (void)

Enables Round Robin prioritization.

This function enables Round Robin Prioritization of DMA channels. In the case of Round Robin Prioritization, the last DMA channel to have transferred data then has the last priority, which comes into play when multiple DMA channels are ready to transfer at the same time.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

11.2.2.11 void DMA_enableTransferDuringReadModifyWrite (void)

Enables the DMA to stop the CPU during a Read-Modify-Write Operation to start a transfer.

This function allows the DMA to stop the CPU in the middle of a read- modify-write operation to transfer data.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:
None

11.2.2.12 void DMA_enableTransfers (uint8_t *channelSelect*)

Enables transfers to be triggered.

This function enables transfers upon appropriate trigger of the selected trigger source for the selected channel.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	14
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

channelSelect is the specified channel to enable transfer for. Valid values are:

- DMA_CHANNEL_0
- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

Returns:
None

11.2.2.13 uint16_t DMA_getInterruptStatus (uint8_t *channelSelect*)

Returns the status of the interrupt flag for the selected channel.

Returns the status of the interrupt flag for the selected channel.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

channelSelect is the specified channel to return the interrupt flag status from. Valid values are:

- **DMA_CHANNEL_0**
- **DMA_CHANNEL_1**
- **DMA_CHANNEL_2**
- **DMA_CHANNEL_3**
- **DMA_CHANNEL_4**
- **DMA_CHANNEL_5**
- **DMA_CHANNEL_6**
- **DMA_CHANNEL_7**

Returns:

One of the following:

- **DMA_INT_INACTIVE**
- **DMA_INT_ACTIVE**
indicating the status of the current interrupt flag

11.2.2.14 **bool DMA_init** (uint8_t *channelSelect*, uint16_t *transferModeSelect*, uint16_t *transferSize*, uint8_t *triggerSourceSelect*, uint8_t *transferUnitSelect*, uint8_t *triggerTypeSelect*)

DEPRECATED - Initializes the specified DMA channel.

This function initializes the specified DMA channel. Upon successful completion of initialization of the selected channel the control registers will be cleared and the given variables will be set. Please note, if transfers have been enabled with the `enableTransfers()` function, then a call to `disableTransfers()` is necessary before re-initialization. Also note, that the trigger sources are device dependent and can be found in the device family data sheet. The amount of DMA channels available are also device specific.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	84
TI Compiler 4.2.1	Size	74
TI Compiler 4.2.1	Speed	74
IAR 5.51.6	None	74
IAR 5.51.6	Size	70
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	76
MSPGCC 4.8.0	Speed	92

Parameters:

channelSelect is the specified channel to initialize. Valid values are:

- **DMA_CHANNEL_0**
- **DMA_CHANNEL_1**
- **DMA_CHANNEL_2**
- **DMA_CHANNEL_3**
- **DMA_CHANNEL_4**
- **DMA_CHANNEL_5**
- **DMA_CHANNEL_6**
- **DMA_CHANNEL_7**

transferModeSelect is the transfer mode of the selected channel. Valid values are:

- **DMA_TRANSFER_SINGLE** [Default] - Single transfer, transfers disabled after transferAmount of transfers.
- **DMA_TRANSFER_BLOCK** - Multiple transfers of transferAmount, transfers disabled once finished.
- **DMA_TRANSFER_BURSTBLOCK** - Multiple transfers of transferAmount interleaved with CPU activity, transfers disabled once finished.
- **DMA_TRANSFER_REPEATED_SINGLE** - Repeated single transfer by trigger.

- **DMA_TRANSFER_REPEATED_BLOCK** - Multiple transfers of transferAmount by trigger.
 - **DMA_TRANSFER_REPEATED_BURSTBLOCK** - Multiple transfers of transferAmount by trigger interleaved with CPU activity.
- Modified bits are **DMADT** of **DMAxCTL** register.

transferSize is the amount of transfers to complete in a block transfer mode, as well as how many transfers to complete before the interrupt flag is set. Valid value is between 1-65535, if 0, no transfers will occur.

Modified bits are **DMAxSZ** of **DMAxSZ** register.

triggerSourceSelect is the source that will trigger the start of each transfer, note that the sources are device specific.

Valid values are:

- **DMA_TRIGGERSOURCE_0** [Default]
- **DMA_TRIGGERSOURCE_1**
- **DMA_TRIGGERSOURCE_2**
- **DMA_TRIGGERSOURCE_3**
- **DMA_TRIGGERSOURCE_4**
- **DMA_TRIGGERSOURCE_5**
- **DMA_TRIGGERSOURCE_6**
- **DMA_TRIGGERSOURCE_7**
- **DMA_TRIGGERSOURCE_8**
- **DMA_TRIGGERSOURCE_9**
- **DMA_TRIGGERSOURCE_10**
- **DMA_TRIGGERSOURCE_11**
- **DMA_TRIGGERSOURCE_12**
- **DMA_TRIGGERSOURCE_13**
- **DMA_TRIGGERSOURCE_14**
- **DMA_TRIGGERSOURCE_15**
- **DMA_TRIGGERSOURCE_16**
- **DMA_TRIGGERSOURCE_17**
- **DMA_TRIGGERSOURCE_18**
- **DMA_TRIGGERSOURCE_19**
- **DMA_TRIGGERSOURCE_20**
- **DMA_TRIGGERSOURCE_21**
- **DMA_TRIGGERSOURCE_22**
- **DMA_TRIGGERSOURCE_23**
- **DMA_TRIGGERSOURCE_24**
- **DMA_TRIGGERSOURCE_25**
- **DMA_TRIGGERSOURCE_26**
- **DMA_TRIGGERSOURCE_27**
- **DMA_TRIGGERSOURCE_28**
- **DMA_TRIGGERSOURCE_29**
- **DMA_TRIGGERSOURCE_30**
- **DMA_TRIGGERSOURCE_31**

Modified bits are **DMAxTSEL** of **DMACTLx** register.

transferUnitSelect is the specified size of transfers. Valid values are:

- **DMA_SIZE_SRCWORD_DSTWORD** [Default]
- **DMA_SIZE_SRCBYTE_DSTWORD**
- **DMA_SIZE_SRCWORD_DSTBYTE**
- **DMA_SIZE_SRCBYTE_DSTBYTE**

Modified bits are **DMASRCBYTE** and **DMADSTBYTE** of **DMAxCTL** register.

triggerTypeSelect is the type of trigger that the trigger signal needs to be to start a transfer. Valid values are:

- **DMA_TRIGGER_RISINGEDGE** [Default]
- **DMA_TRIGGER_HIGH** - A trigger would be a high signal from the trigger source, to be held high through the length of the transfer(s).

Modified bits are **DMALEVEL** of **DMAxCTL** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the initialization process.

11.2.2.15 bool DMA_initialize (DMA_initializeParam * *param*)

Initializes the specified DMA channel.

This function initializes the specified DMA channel. Upon successful completion of initialization of the selected channel the control registers will be cleared and the given variables will be set. Please note, if transfers have been enabled with the enableTransfers() function, then a call to disableTransfers() is necessary before re-initialization. Also note, that the trigger sources are device dependent and can be found in the device family data sheet. The amount of DMA channels available are also device specific.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	154
TI Compiler 4.2.1	Size	94
TI Compiler 4.2.1	Speed	100
IAR 5.51.6	None	144
IAR 5.51.6	Size	94
IAR 5.51.6	Speed	92
MSPGCC 4.8.0	None	244
MSPGCC 4.8.0	Size	94
MSPGCC 4.8.0	Speed	102

Parameters:

param is the pointer to struct for initialization.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the initialization process.

11.2.2.16 uint16_t DMA_NMIAbortStatus (uint8_t *channelSelect*)

Returns the status of the NMIAbort for the selected channel.

This function returns the status of the NMI Abort flag for the selected channel. If this flag has been set, it is because a transfer on this channel was aborted due to a interrupt from an NMI.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

channelSelect is the specified channel to return the status of the NMI Abort flag for. Valid values are:

- DMA_CHANNEL_0
- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6

■ DMA_CHANNEL_7

Returns:

One of the following:

■ DMA_NOTABORTED

■ DMA_ABORTED

indicating the status of the NMIAbort for the selected channel

11.2.2.17 void DMA_setDstAddress (uint8_t *channelSelect*, uint32_t *dstAddress*, uint16_t *directionSelect*)

Sets the destination address and the direction that the destination address will move after a transfer.

This function sets the destination address and the direction that the destination address will move after a transfer is complete. It may be incremented, decremented, or unchanged.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	70
TI Compiler 4.2.1	Size	38
TI Compiler 4.2.1	Speed	38
IAR 5.51.6	None	46
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	116
MSPGCC 4.8.0	Size	48
MSPGCC 4.8.0	Speed	48

Parameters:

channelSelect is the specified channel to set the destination address direction for. Valid values are:

- DMA_CHANNEL_0
- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

dstAddress is the address of where the data will be transferred to.

Modified bits are **DMAxDA** of **DMAxDA** register.

directionSelect is the specified direction of the destination address after a transfer. Valid values are:

- DMA_DIRECTION_UNCHANGED
- DMA_DIRECTION_DECREMENT
- DMA_DIRECTION_INCREMENT

Modified bits are **DMADSTINCR** of **DMAxCTL** register.

Returns:

None

11.2.2.18 void DMA_setSrcAddress (uint8_t *channelSelect*, uint32_t *srcAddress*, uint16_t *directionSelect*)

Sets source address and the direction that the source address will move after a transfer.

This function sets the source address and the direction that the source address will move after a transfer is complete. It may be incremented, decremented or unchanged.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	66
TI Compiler 4.2.1	Size	36
TI Compiler 4.2.1	Speed	36
IAR 5.51.6	None	42
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	44
MSPGCC 4.8.0	Speed	44

Parameters:

channelSelect is the specified channel to set source address direction for. Valid values are:

- DMA_CHANNEL_0
- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

srcAddress is the address of where the data will be transferred from.

Modified bits are **DMASA** of **DMASAR** register.

directionSelect is the specified direction of the source address after a transfer. Valid values are:

- DMA_DIRECTION_UNCHANGED
- DMA_DIRECTION_DECREMENT
- DMA_DIRECTION_INCREMENT

Modified bits are **DMASRCINCR** of **DMASCTL** register.

Returns:

None

11.2.2.19 void DMA_setTransferSize (uint8_t channelSelect, uint16_t transferSize)

Sets the specified amount of transfers for the selected DMA channel.

This function sets the specified amount of transfers for the selected DMA channel without having to reinitialize the DMA channel.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

channelSelect is the specified channel to set source address direction for. Valid values are:

- DMA_CHANNEL_0

- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

transferSize is the amount of transfers to complete in a block transfer mode, as well as how many transfers to complete before the interrupt flag is set. Valid value is between 1-65535, if 0, no transfers will occur. Modified bits are **DMAxSZ** of **DMAxSZ** register.

Returns:
None

11.2.2.20 void DMA_startTransfer (uint8_t channelSelect)

Starts a transfer if using the default trigger source selected in initialization.

This functions triggers a transfer of data from source to destination if the trigger source chosen from initialization is the DMA_TRIGGERSOURCE_0. Please note, this function needs to be called for each (repeated-)single transfer, and when transferAmount of transfers have been complete in (repeated-)block transfers.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

channelSelect is the specified channel to start transfers for. Valid values are:

- DMA_CHANNEL_0
- DMA_CHANNEL_1
- DMA_CHANNEL_2
- DMA_CHANNEL_3
- DMA_CHANNEL_4
- DMA_CHANNEL_5
- DMA_CHANNEL_6
- DMA_CHANNEL_7

Returns:
None

11.3 Programming Example

The following example shows how to initialize and use the DMA API to transfer words from one spot in RAM to another.

```
// Initialize and Setup DMA Channel 0
/*
Base Address of the DMA Module
Configure DMA channel 0
Configure channel for repeated block transfers
DMA interrupt flag will be set after every 16 transfers
Use DMA_startTransfer() function to trigger transfers
Transfer Word-to-Word
Trigger upon Rising Edge of Trigger Source Signal
*/
DMA_init(DMA_BASE,
         DMA_CHANNEL_0,
         DMA_TRANSFER_REPEATED_BLOCK,
         16,
         DMA_TRIGGERSOURCE_0,
         DMA_SIZE_SRCWORD_DSTWORD,
         DMA_TRIGGER_RISINGEDGE);

/*
Base Address of the DMA Module
Configure DMA channel 0
Use 0x1C00 as source
Increment source address after every transfer
*/
DMA_setSrcAddress(DMA_BASE,
                  DMA_CHANNEL_0,
                  0x1C00,
                  DMA_DIRECTION_INCREMENT);

/*
Base Address of the DMA Module
Configure DMA channel 0
Use 0x1C20 as destination
Increment destination address after every transfer
*/
DMA_setDstAddress(DMA_BASE,
                  DMA_CHANNEL_0,
                  0x1C20,
                  DMA_DIRECTION_INCREMENT);

// Enable transfers on DMA channel 0
DMA_enableTransfers(DMA_BASE,
                    DMA_CHANNEL_0);

while(1)
{
    // Start block transfer on DMA channel 0
    DMA_startTransfer(DMA_BASE,
                     DMA_CHANNEL_0);
}
```

12 EUSCI Universal Asynchronous Receiver/Transmitter (EUSCI_A_UART)

Introduction	113
API Functions	113
Programming Example	125

12.1 Introduction

The MSP430Ware library for UART mode features include:

- Odd, even, or non-parity
- Independent transmit and receive shift registers
- Separate transmit and receive buffer registers
- LSB-first or MSB-first data transmit and receive
- Built-in idle-line and address-bit communication protocols for multiprocessor systems
- Receiver start-edge detection for auto wake up from LPMx modes
- Status flags for error detection and suppression
- Status flags for address detection
- Independent interrupt capability for receive and transmit

In UART mode, the USCI transmits and receives characters at a bit rate asynchronous to another device. Timing for each character is based on the selected baud rate of the USCI. The transmit and receive functions use the same baud-rate frequency.

This driver is contained in `eusci_a_uart.c`, with `eusci_a_uart.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	820
TI Compiler 4.2.1	Size	480
TI Compiler 4.2.1	Speed	480
IAR 5.51.6	None	626
IAR 5.51.6	Size	426
IAR 5.51.6	Speed	486
MSPGCC 4.8.0	None	1284
MSPGCC 4.8.0	Size	496
MSPGCC 4.8.0	Speed	738

12.2 API Functions

Functions

- void [EUSCI_A_UART_clearInterruptFlag](#) (uint16_t baseAddress, uint8_t mask)

- void [EUSCI_A_UART_disable](#) (uint16_t baseAddress)
- void [EUSCI_A_UART_disableInterrupt](#) (uint16_t baseAddress, uint8_t mask)
- void [EUSCI_A_UART_enable](#) (uint16_t baseAddress)
- void [EUSCI_A_UART_enableInterrupt](#) (uint16_t baseAddress, uint8_t mask)
- uint8_t [EUSCI_A_UART_getInterruptStatus](#) (uint16_t baseAddress, uint8_t mask)
- uint32_t [EUSCI_A_UART_getReceiveBufferAddress](#) (uint16_t baseAddress)
- uint32_t [EUSCI_A_UART_getTransmitBufferAddress](#) (uint16_t baseAddress)
- bool [EUSCI_A_UART_init](#) (uint16_t baseAddress, EUSCI_A_UART_initParam *param)
- bool [EUSCI_A_UART_initAdvance](#) (uint16_t baseAddress, uint8_t selectClockSource, uint16_t clockPrescaler, uint8_t firstModReg, uint8_t secondModReg, uint8_t parity, uint16_t msborLsbFirst, uint16_t numberOfStopBits, uint16_t uartMode, uint8_t overSampling)
- uint8_t [EUSCI_A_UART_queryStatusFlags](#) (uint16_t baseAddress, uint8_t mask)
- uint8_t [EUSCI_A_UART_receiveData](#) (uint16_t baseAddress)
- void [EUSCI_A_UART_resetDormant](#) (uint16_t baseAddress)
- void [EUSCI_A_UART_selectDeglitchTime](#) (uint16_t baseAddress, uint16_t deglitchTime)
- void [EUSCI_A_UART_setDormant](#) (uint16_t baseAddress)
- void [EUSCI_A_UART_transmitAddress](#) (uint16_t baseAddress, uint8_t transmitAddress)
- void [EUSCI_A_UART_transmitBreak](#) (uint16_t baseAddress)
- void [EUSCI_A_UART_transmitData](#) (uint16_t baseAddress, uint8_t transmitData)

12.2.1 Detailed Description

The EUSCI_A_UART API provides the set of functions required to implement an interrupt driven EUSCI_A_UART driver. The EUSCI_A_UART initialization with the various modes and features is done by the [EUSCI_A_UART_init\(\)](#). At the end of this function EUSCI_A_UART is initialized and stays disabled. [EUSCI_A_UART_enable\(\)](#) enables the EUSCI_A_UART and the module is now ready for transmit and receive. It is recommended to initialize the EUSCI_A_UART via [EUSCI_A_UART_init\(\)](#), enable the required interrupts and then enable EUSCI_A_UART via [EUSCI_A_UART_enable\(\)](#).

The EUSCI_A_UART API is broken into three groups of functions: those that deal with configuration and control of the EUSCI_A_UART modules, those used to send and receive data, and those that deal with interrupt handling and those dealing with DMA.

Configuration and control of the EUSCI_UART are handled by the

- [EUSCI_A_UART_init\(\)](#)
- [EUSCI_A_UART_initAdvance\(\)](#)
- [EUSCI_A_UART_enable\(\)](#)
- [EUSCI_A_UART_disable\(\)](#)
- [EUSCI_A_UART_setDormant\(\)](#)
- [EUSCI_A_UART_resetDormant\(\)](#)
- [EUSCI_A_UART_selectDeglitchTime\(\)](#)

Sending and receiving data via the EUSCI_UART is handled by the

- [EUSCI_A_UART_transmitData\(\)](#)
- [EUSCI_A_UART_receiveData\(\)](#)
- [EUSCI_A_UART_transmitAddress\(\)](#)
- [EUSCI_A_UART_transmitBreak\(\)](#)

Managing the EUSCI_UART interrupts and status are handled by the

- [EUSCI_A_UART_enableInterrupt\(\)](#)
- [EUSCI_A_UART_disableInterrupt\(\)](#)
- [EUSCI_A_UART_getInterruptStatus\(\)](#)
- [EUSCI_A_UART_clearInterruptFlag\(\)](#)
- [EUSCI_A_UART_queryStatusFlags\(\)](#)

DMA related

- EUSCI_A_UART_getReceiveBufferAddressForDMA()
- EUSCI_A_UART_getTransmitBufferAddressForDMA()

12.2.2 Function Documentation

12.2.2.1 void EUSCI_A_UART_clearInterruptFlag (uint16_t *baseAddress*, uint8_t *mask*)

Clears UART interrupt sources.

The UART interrupt source is cleared, so that it no longer asserts. The highest interrupt flag is automatically cleared when an interrupt vector generator is used.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

mask is a bit mask of the interrupt sources to be cleared. Mask value is the logical OR of any of the following:

- EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG
- EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG
- EUSCI_A_UART_STARTBIT_INTERRUPT_FLAG
- EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT_FLAG

Modified bits of **UCAxIFG** register.

Returns:

None

12.2.2.2 void EUSCI_A_UART_disable (uint16_t *baseAddress*)

Disables the UART block.

This will disable operation of the UART block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Modified bits are **UCSWRST** of **UCAxCTL1** register.

Returns:

None

12.2.2.3 void EUSCI_A_UART_disableInterrupt (uint16_t baseAddress, uint8_t mask)

Disables individual UART interrupt sources.

Disables the indicated UART interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	62
TI Compiler 4.2.1	Size	26
TI Compiler 4.2.1	Speed	26
IAR 5.51.6	None	36
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

mask is the bit mask of the interrupt sources to be disabled. Mask value is the logical OR of any of the following:

- **EUSCI_A_UART_RECEIVE_INTERRUPT** - Receive interrupt
- **EUSCI_A_UART_TRANSMIT_INTERRUPT** - Transmit interrupt
- **EUSCI_A_UART_RECEIVE_ERRONEOUSCHAR_INTERRUPT** - Receive erroneous-character interrupt enable
- **EUSCI_A_UART_BREAKCHAR_INTERRUPT** - Receive break character interrupt enable
- **EUSCI_A_UART_STARTBIT_INTERRUPT** - Start bit received interrupt enable
- **EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT** - Transmit complete interrupt enable

Modified bits of **UCAxCTL1** register and bits of **UCAxIE** register.

Returns:

None

12.2.2.4 void EUSCI_A_UART_enable (uint16_t baseAddress)

Enables the UART block.

This will enable operation of the UART block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Modified bits are **UCSWRST** of **UCAxCTL1** register.

Returns:

None

12.2.2.5 void EUSCI_A_UART_enableInterrupt (uint16_t *baseAddress*, uint8_t *mask*)

Enables individual UART interrupt sources.

Enables the indicated UART interrupt sources. The interrupt flag is first and then the corresponding interrupt is enabled. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. Does not clear interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	62
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	36
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	90
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

mask is the bit mask of the interrupt sources to be enabled. Mask value is the logical OR of any of the following:

- **EUSCI_A_UART_RECEIVE_INTERRUPT** - Receive interrupt
- **EUSCI_A_UART_TRANSMIT_INTERRUPT** - Transmit interrupt
- **EUSCI_A_UART_RECEIVE_ERRONEOUSCHAR_INTERRUPT** - Receive erroneous-character interrupt enable
- **EUSCI_A_UART_BREAKCHAR_INTERRUPT** - Receive break character interrupt enable
- **EUSCI_A_UART_STARTBIT_INTERRUPT** - Start bit received interrupt enable
- **EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT** - Transmit complete interrupt enable

Modified bits of **UCAxCTL1** register and bits of **UCAxIE** register.

Returns:

None

12.2.2.6 uint8_t EUSCI_A_UART_getInterruptStatus (uint16_t *baseAddress*, uint8_t *mask*)

Gets the current UART interrupt status.

This returns the interrupt status for the UART module based on which flag is passed.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

mask is the masked interrupt flag status to be returned. Mask value is the logical OR of any of the following:

- EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG
- EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG
- EUSCI_A_UART_STARTBIT_INTERRUPT_FLAG
- EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT_FLAG

Modified bits of **UCAxIFG** register.

Returns:

Logical OR of any of the following:

- EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG
 - EUSCI_A_UART_TRANSMIT_INTERRUPT_FLAG
 - EUSCI_A_UART_STARTBIT_INTERRUPT_FLAG
 - EUSCI_A_UART_TRANSMIT_COMPLETE_INTERRUPT_FLAG
- indicating the status of the masked flags

12.2.2.7 uint32_t EUSCI_A_UART_getReceiveBufferAddress (uint16_t *baseAddress*)

Returns the address of the RX Buffer of the UART for the DMA module.

Returns the address of the UART RX Buffer. This can be used in conjunction with the DMA to store the received data directly to memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Returns:

Address of RX Buffer

12.2.2.8 uint32_t EUSCI_A_UART_getTransmitBufferAddress (uint16_t baseAddress)

Returns the address of the TX Buffer of the UART for the DMA module.

Returns the address of the UART TX Buffer. This can be used in conjunction with the DMA to obtain transmitted data directly from memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Returns:

Address of TX Buffer

12.2.2.9 bool EUSCI_A_UART_init (uint16_t baseAddress, EUSCI_A_UART_initParam * param)

Advanced initialization routine for the UART block. The values to be written into the clockPrescaler, firstModReg, secondModReg and overSampling parameters should be pre-computed and passed into the initialization function.

Upon successful initialization of the UART block, this function will have initialized the module, but the UART block still remains disabled and must be enabled with [EUSCI_A_UART_enable\(\)](#). To calculate values for clockPrescaler, firstModReg, secondModReg and overSampling please use the link below.

http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	232
TI Compiler 4.2.1	Size	142
TI Compiler 4.2.1	Speed	142
IAR 5.51.6	None	188
IAR 5.51.6	Size	160
IAR 5.51.6	Speed	158
MSPGCC 4.8.0	None	398
MSPGCC 4.8.0	Size	150
MSPGCC 4.8.0	Speed	254

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

param is the pointer to struct for initialization.

Modified bits are **UCPEN**, **UCPAR**, **UCMSB**, **UC7BIT**, **UCSPB**, **UCMODEx** and **UCSYNC** of **UCAxCTL0** register; bits **UCSSELx** and **UCSWRST** of **UCAxCTL1** register.

Returns:

STATUS_SUCCESS or STATUS_FAIL of the initialization process

12.2.2.10 bool EUSCI_A_UART_initAdvance (uint16_t *baseAddress*, uint8_t *selectClockSource*, uint16_t *clockPrescalar*, uint8_t *firstModReg*, uint8_t *secondModReg*, uint8_t *parity*, uint16_t *msborLsbFirst*, uint16_t *numberOfStopBits*, uint16_t *uartMode*, uint8_t *overSampling*)

DEPRECATED - Advanced initialization routine for the UART block. The values to be written into the clockPrescalar, firstModReg, secondModReg and overSampling parameters should be pre-computed and passed into the initialization function.

Upon successful initialization of the UART block, this function will have initialized the module, but the UART block still remains disabled and must be enabled with [EUSCI_A_UART_enable\(\)](#). To calculate values for clockPrescalar, firstModReg, secondModReg and overSampling please use the link below.

http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	106
TI Compiler 4.2.1	Size	118
TI Compiler 4.2.1	Speed	118
IAR 5.51.6	None	106
IAR 5.51.6	Size	98
IAR 5.51.6	Speed	98
MSPGCC 4.8.0	None	130
MSPGCC 4.8.0	Size	98
MSPGCC 4.8.0	Speed	242

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

selectClockSource selects Clock source. Valid values are:

- EUSCI_A_UART_CLOCKSOURCE_SMCLK
- EUSCI_A_UART_CLOCKSOURCE_ACLK

clockPrescalar is the value to be written into UCBRx bits

firstModReg is First modulation stage register setting. This value is a pre-calculated value which can be obtained from the Device Users Guide. This value is written into UCBRFx bits of UCAxMCTLW.

secondModReg is Second modulation stage register setting. This value is a pre-calculated value which can be obtained from the Device Users Guide. This value is written into UCBRSx bits of UCAxMCTLW.

parity is the desired parity. Valid values are:

- EUSCI_A_UART_NO_PARITY [Default]
- EUSCI_A_UART_ODD_PARITY
- EUSCI_A_UART_EVEN_PARITY

msborLsbFirst controls direction of receive and transmit shift register. Valid values are:

- EUSCI_A_UART_MSB_FIRST
- EUSCI_A_UART_LSB_FIRST [Default]

numberOfStopBits indicates one/two STOP bits Valid values are:

- EUSCI_A_UART_ONE_STOP_BIT [Default]
- EUSCI_A_UART_TWO_STOP_BITS

uartMode selects the mode of operation Valid values are:

- EUSCI_A_UART_MODE [Default]
- EUSCI_A_UART_IDLE_LINE_MULTI_PROCESSOR_MODE
- EUSCI_A_UART_ADDRESS_BIT_MULTI_PROCESSOR_MODE
- EUSCI_A_UART_AUTOMATIC_BAUDRATE_DETECTION_MODE

overSampling indicates low frequency or oversampling baud generation Valid values are:

- EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION
- EUSCI_A_UART_LOW_FREQUENCY_BAUDRATE_GENERATION

Modified bits are **UCPEN**, **UCPAR**, **UCMSB**, **UC7BIT**, **UCSPB**, **UCMODEx** and **UCSYNC** of **UCAxCTL0** register; bits **UCSSELx** and **UCSWRST** of **UCAxCTL1** register.

Returns:

STATUS_SUCCESS or STATUS_FAIL of the initialization process

12.2.2.11 uint8_t EUSCI_A_UART_queryStatusFlags (uint16_t baseAddress, uint8_t mask)

Gets the current UART status flags.

This returns the status for the UART module based on which flag is passed.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

mask is the masked interrupt flag status to be returned. Mask value is the logical OR of any of the following:

- EUSCI_A_UART_LISTEN_ENABLE
- EUSCI_A_UART_FRAMING_ERROR
- EUSCI_A_UART_OVERRUN_ERROR
- EUSCI_A_UART_PARITY_ERROR
- EUSCI_A_UART_BREAK_DETECT
- EUSCI_A_UART_RECEIVE_ERROR
- EUSCI_A_UART_ADDRESS_RECEIVED
- EUSCI_A_UART_IDLELINE
- EUSCI_A_UART_BUSY

Modified bits of **UCAxSTAT** register.

Returns:

Logical OR of any of the following:

- EUSCI_A_UART_LISTEN_ENABLE
- EUSCI_A_UART_FRAMING_ERROR
- EUSCI_A_UART_OVERRUN_ERROR
- EUSCI_A_UART_PARITY_ERROR
- EUSCI_A_UART_BREAK_DETECT

- EUSCI_A_UART_RECEIVE_ERROR
- EUSCI_A_UART_ADDRESS_RECEIVED
- EUSCI_A_UART_IDLELINE
- EUSCI_A_UART_BUSY

indicating the status of the masked interrupt flags

12.2.2.12 uint8_t EUSCI_A_UART_receiveData (uint16_t *baseAddress*)

Receives a byte that has been sent to the UART Module.

This function reads a byte of data from the UART receive data Register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	32
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	54
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Modified bits of **UCAxRXBUF** register.

Returns:

Returns the byte received from by the UART module, cast as an uint8_t.

12.2.2.13 void EUSCI_A_UART_resetDormant (uint16_t *baseAddress*)

Re-enables UART module from dormant mode.

Not dormant. All received characters set UCRXIFG.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Modified bits are **UCDORM** of **UCAxCTL1** register.

Returns:

None

12.2.2.14 void EUSCI_A_UART_selectDeglitchTime (uint16_t *baseAddress*, uint16_t *deglitchTime*)

Sets the deglitch time.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

deglitchTime is the selected deglitch time Valid values are:

- EUSCI_A_UART_DEGLITCH_TIME_2ns
- EUSCI_A_UART_DEGLITCH_TIME_50ns
- EUSCI_A_UART_DEGLITCH_TIME_100ns
- EUSCI_A_UART_DEGLITCH_TIME_200ns

Returns:

None

12.2.2.15 void EUSCI_A_UART_setDormant (uint16_t *baseAddress*)

Sets the UART module in dormant mode.

Puts USCI in sleep mode Only characters that are preceded by an idle-line or with address bit set UCRXIFG. In UART mode with automatic baud-rate detection, only the combination of a break and sync field sets UCRXIFG.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Modified bits of **UCAxCTL1** register.

Returns:

None

12.2.2.16 void EUSCI_A_UART_transmitAddress (uint16_t *baseAddress*, uint8_t *transmitAddress*)

Transmits the next byte to be transmitted marked as address depending on selected multiprocessor mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	20
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

transmitAddress is the next byte to be transmitted

Modified bits of **UCAxTXBUF** register and bits of **UCAxCTL1** register.

Returns:

None

12.2.2.17 void EUSCI_A_UART_transmitBreak (uint16_t *baseAddress*)

Transmit break.

Transmits a break with the next write to the transmit buffer. In UART mode with automatic baud-rate detection, EUSCI_A_UART_AUTOMATICBAUDRATE_SYNC(0x55) must be written into UCAxTXBUF to generate the required break/sync fields. Otherwise, DEFAULT_SYNC(0x00) must be written into the transmit buffer. Also ensures module is ready for transmitting the next data.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	60
TI Compiler 4.2.1	Size	42
TI Compiler 4.2.1	Speed	42
IAR 5.51.6	None	68
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	60
MSPGCC 4.8.0	None	92
MSPGCC 4.8.0	Size	54
MSPGCC 4.8.0	Speed	48

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Modified bits of **UCAxTXBUF** register and bits of **UCAxCTL1** register.

Returns:

None

12.2.2.18 void EUSCI_A_UART_transmitData (uint16_t *baseAddress*, uint8_t *transmitData*)

Transmits a byte from the UART Module.

This function will place the supplied data into UART transmit data register to start transmission

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	38
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

transmitData data to be transmitted from the UART module

Modified bits of **UCAxTXBUF** register.

Returns:

None

12.3 Programming Example

The following example shows how to use the EUSCI_UART API to initialize the EUSCI_UART, transmit characters, and receive characters.

```
// Configure UART
if ( STATUS_FAIL == EUSCI_A_UART_init(EUSCI_A0_BASE,
    EUSCI_A_UART_CLOCKSOURCE_ACLK,
    CLOCK_VALUE,
    32768,
    EUSCI_A_UART_NO_PARITY,
    EUSCI_A_UART_LSB_FIRST,
    EUSCI_A_UART_ONE_STOP_BIT,
    EUSCI_A_UART_MODE,
    EUSCI_A_UART_LOW_FREQUENCY_BAUDRATE_GENERATION ) ) {
    return;
}

EUSCI_A_UART_enable(EUSCI_A0_BASE);

// Enable USCI_A0 RX interrupt
EUSCI_A_UART_enableInterrupt(EUSCI_A0_BASE,
    EUSCI_A_UART_RECEIVE_INTERRUPT);
```


13 EUSCI Synchronous Peripheral Interface (EUSCI_A_SPI)

Introduction	126
API Functions	126
Programming Example	138

13.1 Introduction

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard named by Motorola that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame.

This library provides the API for handling a SPI communication using EUSCI.

The SPI module can be configured as either a master or a slave device.

The SPI module also includes a programmable bit rate clock divider and prescaler to generate the output serial clock derived from the module's input clock.

This driver is contained in `eusci_a_spi.c`, with `eusci_a_spi.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	818
TI Compiler 4.2.1	Size	462
TI Compiler 4.2.1	Speed	462
IAR 5.51.6	None	578
IAR 5.51.6	Size	462
IAR 5.51.6	Speed	470
MSPGCC 4.8.0	None	1150
MSPGCC 4.8.0	Size	470
MSPGCC 4.8.0	Speed	472

13.2 Functions

Functions

- void [EUSCI_A_SPI_changeClockPhasePolarity](#) (uint16_t baseAddress, uint16_t clockPhase, uint16_t clockPolarity)
- void [EUSCI_A_SPI_changeMasterClock](#) (uint16_t baseAddress, EUSCI_A_SPI_changeMasterClockParam *param)
- void [EUSCI_A_SPI_clearInterruptFlag](#) (uint16_t baseAddress, uint8_t mask)
- void [EUSCI_A_SPI_disable](#) (uint16_t baseAddress)
- void [EUSCI_A_SPI_disableInterrupt](#) (uint16_t baseAddress, uint8_t mask)
- void [EUSCI_A_SPI_enable](#) (uint16_t baseAddress)
- void [EUSCI_A_SPI_enableInterrupt](#) (uint16_t baseAddress, uint8_t mask)
- uint8_t [EUSCI_A_SPI_getInterruptStatus](#) (uint16_t baseAddress, uint8_t mask)
- uint32_t [EUSCI_A_SPI_getReceiveBufferAddress](#) (uint16_t baseAddress)
- uint32_t [EUSCI_A_SPI_getTransmitBufferAddress](#) (uint16_t baseAddress)
- void [EUSCI_A_SPI_initMaster](#) (uint16_t baseAddress, EUSCI_A_SPI_initMasterParam *param)

- void [EUSCI_A_SPI_initSlave](#) (uint16_t baseAddress, EUSCI_A_SPI_initSlaveParam *param)
- uint16_t [EUSCI_A_SPI_isBusy](#) (uint16_t baseAddress)
- void [EUSCI_A_SPI_masterChangeClock](#) (uint16_t baseAddress, uint32_t clockSourceFrequency, uint32_t desiredSpiClock)
- void [EUSCI_A_SPI_masterInit](#) (uint16_t baseAddress, uint8_t selectClockSource, uint32_t clockSourceFrequency, uint32_t desiredSpiClock, uint16_t msbFirst, uint16_t clockPhase, uint16_t clockPolarity, uint16_t spiMode)
- uint8_t [EUSCI_A_SPI_receiveData](#) (uint16_t baseAddress)
- void [EUSCI_A_SPI_select4PinFunctionality](#) (uint16_t baseAddress, uint8_t select4PinFunctionality)
- void [EUSCI_A_SPI_slaveInit](#) (uint16_t baseAddress, uint16_t msbFirst, uint16_t clockPhase, uint16_t clockPolarity, uint16_t spiMode)
- void [EUSCI_A_SPI_transmitData](#) (uint16_t baseAddress, uint8_t transmitData)

13.2.1 Detailed Description

To use the module as a master, the user must call [EUSCI_A_SPI_masterInit\(\)](#) to configure the SPI Master. This is followed by enabling the SPI module using [EUSCI_A_SPI_enable\(\)](#). The interrupts are then enabled (if needed). It is recommended to enable the SPI module before enabling the interrupts. A data transmit is then initiated using [EUSCI_A_SPI_transmitData\(\)](#) and then when the receive flag is set, the received data is read using [EUSCI_A_SPI_receiveData\(\)](#) and this indicates that an RX/TX operation is complete.

To use the module as a slave, initialization is done using [EUSCI_A_SPI_slaveInit\(\)](#) and this is followed by enabling the module using [EUSCI_A_SPI_enable\(\)](#). Following this, the interrupts may be enabled as needed. When the receive flag is set, data is first transmitted using [EUSCI_A_SPI_transmitData\(\)](#) and this is followed by a data reception by [EUSCI_A_SPI_receiveData\(\)](#)

The SPI API is broken into 3 groups of functions: those that deal with status and initialization, those that handle data, and those that manage interrupts.

The status and initialization of the SPI module are managed by

- [EUSCI_A_SPI_masterInit\(\)](#)
- [EUSCI_A_SPI_slaveInit\(\)](#)
- [EUSCI_A_SPI_disable\(\)](#)
- [EUSCI_A_SPI_enable\(\)](#)
- [EUSCI_A_SPI_masterChangeClock\(\)](#)
- [EUSCI_A_SPI_isBusy\(\)](#)
- [EUSCI_A_SPI_select4PinFunctionality\(\)](#)
- [EUSCI_A_SPI_changeClockPhasePolarity\(\)](#)

Data handling is done by

- [EUSCI_A_SPI_transmitData\(\)](#)
- [EUSCI_A_SPI_receiveData\(\)](#)

Interrupts from the SPI module are managed using

- [EUSCI_A_SPI_disableInterrupt\(\)](#)
- [EUSCI_A_SPI_enableInterrupt\(\)](#)
- [EUSCI_A_SPI_getInterruptStatus\(\)](#)
- [EUSCI_A_SPI_clearInterruptFlag\(\)](#)

DMA related

- [EUSCI_A_SPI_getReceiveBufferAddressForDMA\(\)](#)
- [EUSCI_A_SPI_getTransmitBufferAddressForDMA\(\)](#)

13.2.2 Function Documentation

13.2.2.1 void EUSCI_A_SPI_changeClockPhasePolarity (uint16_t *baseAddress*, uint16_t *clockPhase*, uint16_t *clockPolarity*)

Changes the SPI clock phase and polarity. At the end of this function call, SPI module is left enabled.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	56
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	28
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	94
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

clockPhase is clock phase select. Valid values are:

- EUSCI_A_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT [Default]
- EUSCI_A_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT

clockPolarity is clock polarity select Valid values are:

- EUSCI_A_SPI_CLOCKPOLARITY_INACTIVITY_HIGH
- EUSCI_A_SPI_CLOCKPOLARITY_INACTIVITY_LOW [Default]

Modified bits are **UCCKPL**, **UCCKPH** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

13.2.2.2 void EUSCI_A_SPI_changeMasterClock (uint16_t *baseAddress*, EUSCI_A_SPI_changeMasterClockParam * *param*)

Initializes the SPI Master clock. At the end of this function call, SPI module is left enabled.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	54
TI Compiler 4.2.1	Size	40
TI Compiler 4.2.1	Speed	40
IAR 5.51.6	None	50
IAR 5.51.6	Size	46
IAR 5.51.6	Speed	46
MSPGCC 4.8.0	None	88
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

param is the pointer to struct for master clock setting.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

13.2.2.3 void EUSCI_A_SPI_clearInterruptFlag (uint16_t *baseAddress*, uint8_t *mask*)

Clears the selected SPI interrupt status flag.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

mask is the masked interrupt flag to be cleared. Mask value is the logical OR of any of the following:

- **EUSCI_A_SPI_TRANSMIT_INTERRUPT**
- **EUSCI_A_SPI_RECEIVE_INTERRUPT**

Modified bits of **UCAxIFG** register.

Returns:

None

13.2.2.4 void EUSCI_A_SPI_disable (uint16_t *baseAddress*)

Disables the SPI block.

This will disable operation of the SPI block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:
None

13.2.2.5 void EUSCI_A_SPI_disableInterrupt (uint16_t *baseAddress*, uint8_t *mask*)

Disables individual SPI interrupt sources.

Disables the indicated SPI interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

mask is the bit mask of the interrupt sources to be disabled. Mask value is the logical OR of any of the following:

- EUSCI_A_SPI_TRANSMIT_INTERRUPT
- EUSCI_A_SPI_RECEIVE_INTERRUPT

Modified bits of **UCAxIE** register.

Returns:
None

13.2.2.6 void EUSCI_A_SPI_enable (uint16_t *baseAddress*)

Enables the SPI block.

This will enable operation of the SPI block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:
None

13.2.2.7 void EUSCI_A_SPI_enableInterrupt (uint16_t *baseAddress*, uint8_t *mask*)

Enables individual SPI interrupt sources.

Enables the indicated SPI interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. Does not clear interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

mask is the bit mask of the interrupt sources to be enabled. Mask value is the logical OR of any of the following:

- EUSCI_A_SPI_TRANSMIT_INTERRUPT
- EUSCI_A_SPI_RECEIVE_INTERRUPT

Modified bits of **UCAxIFG** register and bits of **UCAxIE** register.

Returns:
None

13.2.2.8 uint8_t EUSCI_A_SPI_getInterruptStatus (uint16_t *baseAddress*, uint8_t *mask*)

Gets the current SPI interrupt status.

This returns the interrupt status for the SPI module based on which flag is passed.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

mask is the masked interrupt flag status to be returned. Mask value is the logical OR of any of the following:

- EUSCI_A_SPI_TRANSMIT_INTERRUPT

■ EUSCI_A_SPI_RECEIVE_INTERRUPT

Returns:

Logical OR of any of the following:

- EUSCI_A_SPI_TRANSMIT_INTERRUPT
- EUSCI_A_SPI_RECEIVE_INTERRUPT
indicating the status of the masked interrupts

13.2.2.9 uint32_t EUSCI_A_SPI_getReceiveBufferAddress (uint16_t *baseAddress*)

Returns the address of the RX Buffer of the SPI for the DMA module.

Returns the address of the SPI RX Buffer. This can be used in conjunction with the DMA to store the received data directly to memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Returns:

the address of the RX Buffer

13.2.2.10 uint32_t EUSCI_A_SPI_getTransmitBufferAddress (uint16_t *baseAddress*)

Returns the address of the TX Buffer of the SPI for the DMA module.

Returns the address of the SPI TX Buffer. This can be used in conjunction with the DMA to obtain transmitted data directly from memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Returns:

the address of the TX Buffer

13.2.2.11 void EUSCI_A_SPI_initMaster (uint16_t *baseAddress*, EUSCI_A_SPI_initMasterParam * *param*)

Initializes the SPI Master block.

Upon successful initialization of the SPI master block, this function will have set the bus speed for the master, but the SPI Master block still remains disabled and must be enabled with [EUSCI_A_SPI_enable\(\)](#)

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	114
TI Compiler 4.2.1	Size	84
TI Compiler 4.2.1	Speed	84
IAR 5.51.6	None	98
IAR 5.51.6	Size	92
IAR 5.51.6	Speed	92
MSPGCC 4.8.0	None	190
MSPGCC 4.8.0	Size	90
MSPGCC 4.8.0	Speed	90

Parameters:***baseAddress*** is the base address of the EUSCI_A_SPI Master module.***param*** is the pointer to struct for master initialization.

Modified bits are **UCCKPH**, **UCCKPL**, **UC7BIT**, **UCMSB**, **UCSSELx** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

STATUS_SUCCESS

13.2.2.12 void EUSCI_A_SPI_initSlave (uint16_t *baseAddress*, EU- SCI_A_SPI_initSlaveParam * *param*)

Initializes the SPI Slave block.

Upon successful initialization of the SPI slave block, this function will have initialized the slave block, but the SPI Slave block still remains disabled and must be enabled with [EUSCI_A_SPI_enable\(\)](#)

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	56
TI Compiler 4.2.1	Size	34
TI Compiler 4.2.1	Speed	34
IAR 5.51.6	None	36
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	34
MSPGCC 4.8.0	Speed	34

Parameters:***baseAddress*** is the base address of the EUSCI_A_SPI Slave module.

param is the pointer to struct for slave initialization.

Modified bits are **UCMSB**, **UCMST**, **UC7BIT**, **UCCKPL**, **UCCKPH**, **UCMODE** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

STATUS_SUCCESS

13.2.2.13 uint16_t EUSCI_A_SPI_isBusy (uint16_t *baseAddress*)

Indicates whether or not the SPI bus is busy.

This function returns an indication of whether or not the SPI bus is busy. This function checks the status of the bus via UCBBUSY bit

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	20
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Returns:

One of the following:

- **EUSCI_A_SPI_BUSY**
- **EUSCI_A_SPI_NOT_BUSY**
indicating if the EUSCI_A_SPI is busy

13.2.2.14 void EUSCI_A_SPI_masterChangeClock (uint16_t *baseAddress*, uint32_t *clockSourceFrequency*, uint32_t *desiredSpiClock*)

DEPRECATED - Initializes the SPI Master clock. At the end of this function call, SPI module is left enabled.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	90
TI Compiler 4.2.1	Size	36
TI Compiler 4.2.1	Speed	36
IAR 5.51.6	None	66
IAR 5.51.6	Size	46
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	92
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	60

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

clockSourceFrequency is the frequency of the selected clock source

desiredSpiClock is the desired clock rate for SPI communication

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

13.2.2.15 void EUSCI_A_SPI_masterInit (uint16_t *baseAddress*, uint8_t *selectClockSource*, uint32_t *clockSourceFrequency*, uint32_t *desiredSpiClock*, uint16_t *msbFirst*, uint16_t *clockPhase*, uint16_t *clockPolarity*, uint16_t *spiMode*)

DEPRECATED - Initializes the SPI Master block.

Upon successful initialization of the SPI master block, this function will have set the bus speed for the master, but the SPI Master block still remains disabled and must be enabled with [EUSCI_A_SPI_enable\(\)](#)

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	106
TI Compiler 4.2.1	Size	118
TI Compiler 4.2.1	Speed	118
IAR 5.51.6	None	108
IAR 5.51.6	Size	102
IAR 5.51.6	Speed	102
MSPGCC 4.8.0	None	126
MSPGCC 4.8.0	Size	94
MSPGCC 4.8.0	Speed	86

Parameters:

baseAddress is the base address of the EUSCI_A_SPI Master module.

selectClockSource selects Clock source. Valid values are:

- EUSCI_A_SPI_CLOCKSOURCE_ACLK
- EUSCI_A_SPI_CLOCKSOURCE_SMCLK

clockSourceFrequency is the frequency of the selected clock source

desiredSpiClock is the desired clock rate for SPI communication

msbFirst controls the direction of the receive and transmit shift register. Valid values are:

- EUSCI_A_SPI_MSB_FIRST
- EUSCI_A_SPI_LSB_FIRST [Default]

clockPhase is clock phase select. Valid values are:

- EUSCI_A_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT [Default]
- EUSCI_A_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT

clockPolarity is clock polarity select Valid values are:

- EUSCI_A_SPI_CLOCKPOLARITY_INACTIVITY_HIGH
- EUSCI_A_SPI_CLOCKPOLARITY_INACTIVITY_LOW [Default]

spiMode is SPI mode select Valid values are:

- EUSCI_A_SPI_3PIN
- EUSCI_A_SPI_4PIN_UCxSTE_ACTIVE_HIGH
- EUSCI_A_SPI_4PIN_UCxSTE_ACTIVE_LOW

Modified bits are **UCCKPH**, **UCCKPL**, **UC7BIT**, **UCMSB**, **UCSSELx** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

STATUS_SUCCESS

13.2.2.16 uint8_t EUSCI_A_SPI_receiveData (uint16_t *baseAddress*)

Receives a byte that has been sent to the SPI Module.

This function reads a byte of data from the SPI receive data Register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Returns:

Returns the byte received from by the SPI module, cast as an uint8_t.

13.2.2.17 void EUSCI_A_SPI_select4PinFunctionality (uint16_t *baseAddress*, uint8_t *select4PinFunctionality*)

Selects 4Pin Functionality.

This function should be invoked only in 4-wire mode. Invoking this function has no effect in 3-wire mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	14
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	50
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

select4PinFunctionality selects 4 pin functionality Valid values are:

- EUSCI_A_SPI_PREVENT_CONFLICTS_WITH_OTHER_MASTERS
- EUSCI_A_SPI_ENABLE_SIGNAL_FOR_4WIRE_SLAVE

Modified bits are **UCSTEM** of **UCAxCTLW0** register.

Returns:

None

13.2.2.18 void EUSCI_A_SPI_slaveInit (uint16_t *baseAddress*, uint16_t *msbFirst*, uint16_t *clockPhase*, uint16_t *clockPolarity*, uint16_t *spiMode*)

DEPRECATED - Initializes the SPI Slave block.

Upon successful initialization of the SPI slave block, this function will have initialized the slave block, but the SPI Slave block still remains disabled and must be enabled with [EUSCI_A_SPI_enable\(\)](#)

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	82
TI Compiler 4.2.1	Size	30
TI Compiler 4.2.1	Speed	30
IAR 5.51.6	None	62
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

Parameters:

baseAddress is the base address of the EUSCI_A_SPI Slave module.

msbFirst controls the direction of the receive and transmit shift register. Valid values are:

- EUSCI_A_SPI_MSB_FIRST
- EUSCI_A_SPI_LSB_FIRST [Default]

clockPhase is clock phase select. Valid values are:

- EUSCI_A_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT [Default]
- EUSCI_A_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT

clockPolarity is clock polarity select Valid values are:

- EUSCI_A_SPI_CLOCKPOLARITY_INACTIVITY_HIGH
- EUSCI_A_SPI_CLOCKPOLARITY_INACTIVITY_LOW [Default]

spiMode is SPI mode select Valid values are:

- EUSCI_A_SPI_3PIN
- EUSCI_A_SPI_4PIN_UCxSTE_ACTIVE_HIGH
- EUSCI_A_SPI_4PIN_UCxSTE_ACTIVE_LOW

Modified bits are **UCMSB**, **UCMST**, **UC7BIT**, **UCCKPL**, **UCCKPH**, **UCMODE** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

STATUS_SUCCESS

13.2.2.19 void EUSCI_A_SPI_transmitData (uint16_t *baseAddress*, uint8_t *transmitData*)

Transmits a byte from the SPI Module.

This function will place the supplied data into SPI transmit data register to start transmission.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	30
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:**baseAddress** is the base address of the EUSCI_A_SPI module.**transmitData** data to be transmitted from the SPI module**Returns:**

None

13.3 Programming Example

The following example shows how to use the SPI API to configure the SPI module as a master device, and how to do a simple send of data.

```
//Initialize slave to MSB first, inactive high clock polarity and 3 wire SPI
returnValue = EUSCI_A_SPI_slaveInit(EUSCI_A0_BASE,
    EUSCI_A_SPI_MSB_FIRST,
    EUSCI_A_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT,
    EUSCI_A_SPI_CLOCKPOLARITY_INACTIVITY_HIGH
);

if (STATUS_FAIL == returnValue){
    return;
}

//Enable SPI Module
EUSCI_A_SPI_enable(EUSCI_A0_BASE);

//Enable Receive interrupt
EUSCI_A_SPI_enableInterrupt(EUSCI_A0_BASE,
    EUSCI_A_SPI_RECEIVE_INTERRUPT
);
```

14 EUSCI Synchronous Peripheral Interface (EUSCI_B_SPI)

Introduction	139
API Functions	139
Programming Example	151

14.1 Introduction

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard named by Motorola that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame.

This library provides the API for handling a SPI communication using EUSCI.

The SPI module can be configured as either a master or a slave device.

The SPI module also includes a programmable bit rate clock divider and prescaler to generate the output serial clock derived from the module's input clock.

This driver is contained in `eusci_b_spi.c`, with `eusci_b_spi.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	812
TI Compiler 4.2.1	Size	458
TI Compiler 4.2.1	Speed	458
IAR 5.51.6	None	568
IAR 5.51.6	Size	438
IAR 5.51.6	Speed	462
MSPGCC 4.8.0	None	1138
MSPGCC 4.8.0	Size	466
MSPGCC 4.8.0	Speed	464

14.2 Functions

Functions

- void [EUSCI_B_SPI_changeClockPhasePolarity](#) (uint16_t baseAddress, uint16_t clockPhase, uint16_t clockPolarity)
- void [EUSCI_B_SPI_changeMasterClock](#) (uint16_t baseAddress, EUSCI_B_SPI_changeMasterClockParam *param)
- void [EUSCI_B_SPI_clearInterruptFlag](#) (uint16_t baseAddress, uint8_t mask)
- void [EUSCI_B_SPI_disable](#) (uint16_t baseAddress)
- void [EUSCI_B_SPI_disableInterrupt](#) (uint16_t baseAddress, uint8_t mask)
- void [EUSCI_B_SPI_enable](#) (uint16_t baseAddress)
- void [EUSCI_B_SPI_enableInterrupt](#) (uint16_t baseAddress, uint8_t mask)
- uint8_t [EUSCI_B_SPI_getInterruptStatus](#) (uint16_t baseAddress, uint8_t mask)
- uint32_t [EUSCI_B_SPI_getReceiveBufferAddress](#) (uint16_t baseAddress)
- uint32_t [EUSCI_B_SPI_getTransmitBufferAddress](#) (uint16_t baseAddress)
- void [EUSCI_B_SPI_initMaster](#) (uint16_t baseAddress, EUSCI_B_SPI_initMasterParam *param)

- void [EUSCI_B_SPI_initSlave](#) (uint16_t baseAddress, EUSCI_B_SPI_initSlaveParam *param)
- uint16_t [EUSCI_B_SPI_isBusy](#) (uint16_t baseAddress)
- void [EUSCI_B_SPI_masterChangeClock](#) (uint16_t baseAddress, uint32_t clockSourceFrequency, uint32_t desiredSpiClock)
- void [EUSCI_B_SPI_masterInit](#) (uint16_t baseAddress, uint8_t selectClockSource, uint32_t clockSourceFrequency, uint32_t desiredSpiClock, uint16_t msbFirst, uint16_t clockPhase, uint16_t clockPolarity, uint16_t spiMode)
- uint8_t [EUSCI_B_SPI_receiveData](#) (uint16_t baseAddress)
- void [EUSCI_B_SPI_select4PinFunctionality](#) (uint16_t baseAddress, uint8_t select4PinFunctionality)
- void [EUSCI_B_SPI_slaveInit](#) (uint16_t baseAddress, uint16_t msbFirst, uint16_t clockPhase, uint16_t clockPolarity, uint16_t spiMode)
- void [EUSCI_B_SPI_transmitData](#) (uint16_t baseAddress, uint8_t transmitData)

14.2.1 Detailed Description

To use the module as a master, the user must call [EUSCI_B_SPI_masterInit\(\)](#) to configure the SPI Master. This is followed by enabling the SPI module using [EUSCI_B_SPI_enable\(\)](#). The interrupts are then enabled (if needed). It is recommended to enable the SPI module before enabling the interrupts. A data transmit is then initiated using [EUSCI_B_SPI_transmitData\(\)](#) and then when the receive flag is set, the received data is read using [EUSCI_B_SPI_receiveData\(\)](#) and this indicates that an RX/TX operation is complete.

To use the module as a slave, initialization is done using [EUSCI_B_SPI_slaveInit\(\)](#) and this is followed by enabling the module using [EUSCI_B_SPI_enable\(\)](#). Following this, the interrupts may be enabled as needed. When the receive flag is set, data is first transmitted using [EUSCI_B_SPI_transmitData\(\)](#) and this is followed by a data reception by [EUSCI_B_SPI_receiveData\(\)](#)

The SPI API is broken into 3 groups of functions: those that deal with status and initialization, those that handle data, and those that manage interrupts.

The status and initialization of the SPI module are managed by

- [EUSCI_B_SPI_masterInit\(\)](#)
- [EUSCI_B_SPI_slaveInit\(\)](#)
- [EUSCI_B_SPI_disable\(\)](#)
- [EUSCI_B_SPI_enable\(\)](#)
- [EUSCI_B_SPI_masterChangeClock\(\)](#)
- [EUSCI_B_SPI_isBusy\(\)](#)
- [EUSCI_B_SPI_select4PinFunctionality\(\)](#)
- [EUSCI_B_SPI_changeClockPhasePolarity\(\)](#)

Data handling is done by

- [EUSCI_B_SPI_transmitData\(\)](#)
- [EUSCI_B_SPI_receiveData\(\)](#)

Interrupts from the SPI module are managed using

- [EUSCI_B_SPI_disableInterrupt\(\)](#)
- [EUSCI_B_SPI_enableInterrupt\(\)](#)
- [EUSCI_B_SPI_getInterruptStatus\(\)](#)
- [EUSCI_B_SPI_clearInterruptFlag\(\)](#)

DMA related

- [EUSCI_B_SPI_getReceiveBufferAddressForDMA\(\)](#)
- [EUSCI_B_SPI_getTransmitBufferAddressForDMA\(\)](#)

14.2.2 Function Documentation

14.2.2.1 void EUSCI_B_SPI_changeClockPhasePolarity (uint16_t *baseAddress*, uint16_t *clockPhase*, uint16_t *clockPolarity*)

Changes the SPI clock phase and polarity. At the end of this function call, SPI module is left enabled.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	56
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	28
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	94
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

clockPhase is clock phase select. Valid values are:

- EUSCI_B_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT [Default]
- EUSCI_B_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT

clockPolarity is clock polarity select Valid values are:

- EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH
- EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_LOW [Default]

Modified bits are **UCCKPL**, **UCCKPH** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

14.2.2.2 void EUSCI_B_SPI_changeMasterClock (uint16_t *baseAddress*, EUSCI_B_SPI_changeMasterClockParam * *param*)

Initializes the SPI Master clock. At the end of this function call, SPI module is left enabled.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	54
TI Compiler 4.2.1	Size	40
TI Compiler 4.2.1	Speed	40
IAR 5.51.6	None	50
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	46
MSPGCC 4.8.0	None	88
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

param is the pointer to struct for master clock setting.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

14.2.2.3 void EUSCI_B_SPI_clearInterruptFlag (uint16_t *baseAddress*, uint8_t *mask*)

Clears the selected SPI interrupt status flag.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

mask is the masked interrupt flag to be cleared. Mask value is the logical OR of any of the following:

- **EUSCI_B_SPI_TRANSMIT_INTERRUPT**
- **EUSCI_B_SPI_RECEIVE_INTERRUPT**

Modified bits of **UCAxIFG** register.

Returns:

None

14.2.2.4 void EUSCI_B_SPI_disable (uint16_t *baseAddress*)

Disables the SPI block.

This will disable operation of the SPI block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:
None

14.2.2.5 void EUSCI_B_SPI_disableInterrupt (uint16_t *baseAddress*, uint8_t *mask*)

Disables individual SPI interrupt sources.

Disables the indicated SPI interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

mask is the bit mask of the interrupt sources to be disabled. Mask value is the logical OR of any of the following:

- EUSCI_B_SPI_TRANSMIT_INTERRUPT
- EUSCI_B_SPI_RECEIVE_INTERRUPT

Modified bits of **UCAxIE** register.

Returns:
None

14.2.2.6 void EUSCI_B_SPI_enable (uint16_t *baseAddress*)

Enables the SPI block.

This will enable operation of the SPI block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:
None

14.2.2.7 void EUSCI_B_SPI_enableInterrupt (uint16_t *baseAddress*, uint8_t *mask*)

Enables individual SPI interrupt sources.

Enables the indicated SPI interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. Does not clear interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

mask is the bit mask of the interrupt sources to be enabled. Mask value is the logical OR of any of the following:

- EUSCI_B_SPI_TRANSMIT_INTERRUPT
- EUSCI_B_SPI_RECEIVE_INTERRUPT

Modified bits of **UCAxIFG** register and bits of **UCAxIE** register.

Returns:
None

14.2.2.8 uint8_t EUSCI_B_SPI_getInterruptStatus (uint16_t *baseAddress*, uint8_t *mask*)

Gets the current SPI interrupt status.

This returns the interrupt status for the SPI module based on which flag is passed.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

mask is the masked interrupt flag status to be returned. Mask value is the logical OR of any of the following:

- EUSCI_B_SPI_TRANSMIT_INTERRUPT

■ EUSCI_B_SPI_RECEIVE_INTERRUPT

Returns:

Logical OR of any of the following:

- EUSCI_B_SPI_TRANSMIT_INTERRUPT
 - EUSCI_B_SPI_RECEIVE_INTERRUPT
- indicating the status of the masked interrupts

14.2.2.9 uint32_t EUSCI_B_SPI_getReceiveBufferAddress (uint16_t *baseAddress*)

Returns the address of the RX Buffer of the SPI for the DMA module.

Returns the address of the SPI RX Buffer. This can be used in conjunction with the DMA to store the received data directly to memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Returns:

the address of the RX Buffer

14.2.2.10 uint32_t EUSCI_B_SPI_getTransmitBufferAddress (uint16_t *baseAddress*)

Returns the address of the TX Buffer of the SPI for the DMA module.

Returns the address of the SPI TX Buffer. This can be used in conjunction with the DMA to obtain transmitted data directly from memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Returns:

the address of the TX Buffer

14.2.2.11 void EUSCI_B_SPI_initMaster (uint16_t *baseAddress*, EUSCI_B_SPI_initMasterParam * *param*)

Initializes the SPI Master block.

Upon successful initialization of the SPI master block, this function will have set the bus speed for the master, but the SPI Master block still remains disabled and must be enabled with [EUSCI_B_SPI_enable\(\)](#)

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	108
TI Compiler 4.2.1	Size	80
TI Compiler 4.2.1	Speed	80
IAR 5.51.6	None	90
IAR 5.51.6	Size	78
IAR 5.51.6	Speed	86
MSPGCC 4.8.0	None	180
MSPGCC 4.8.0	Size	86
MSPGCC 4.8.0	Speed	86

Parameters:***baseAddress*** is the base address of the EUSCI_B_SPI Master module.***param*** is the pointer to struct for master initialization.

Modified bits are **UCCKPH**, **UCCKPL**, **UC7BIT**, **UCMSB**, **UCSSELx** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

STATUS_SUCCESS

14.2.2.12 void EUSCI_B_SPI_initSlave (uint16_t *baseAddress*, EUSCI_B_SPI_initSlaveParam * *param*)

Initializes the SPI Slave block.

Upon successful initialization of the SPI slave block, this function will have initialized the slave block, but the SPI Slave block still remains disabled and must be enabled with [EUSCI_B_SPI_enable\(\)](#)

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	56
TI Compiler 4.2.1	Size	34
TI Compiler 4.2.1	Speed	34
IAR 5.51.6	None	36
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	34
MSPGCC 4.8.0	Speed	34

Parameters:***baseAddress*** is the base address of the EUSCI_B_SPI Slave module.

param is the pointer to struct for slave initialization.

Modified bits are **UCMSB**, **UCMST**, **UC7BIT**, **UCCKPL**, **UCCKPH**, **UCMODE** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

STATUS_SUCCESS

14.2.2.13 uint16_t EUSCI_B_SPI_isBusy (uint16_t *baseAddress*)

Indicates whether or not the SPI bus is busy.

This function returns an indication of whether or not the SPI bus is busy. This function checks the status of the bus via UCBBUSY bit

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Returns:

One of the following:

- **EUSCI_B_SPI_BUSY**
- **EUSCI_B_SPI_NOT_BUSY**
indicating if the EUSCI_B_SPI is busy

14.2.2.14 void EUSCI_B_SPI_masterChangeClock (uint16_t *baseAddress*, uint32_t *clockSourceFrequency*, uint32_t *desiredSpiClock*)

DEPRECATED - Initializes the SPI Master clock. At the end of this function call, SPI module is left enabled.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	90
TI Compiler 4.2.1	Size	36
TI Compiler 4.2.1	Speed	36
IAR 5.51.6	None	66
IAR 5.51.6	Size	46
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	92
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	60

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

clockSourceFrequency is the frequency of the selected clock source

desiredSpiClock is the desired clock rate for SPI communication

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

14.2.2.15 void EUSCI_B_SPI_masterInit (uint16_t *baseAddress*, uint8_t *selectClockSource*, uint32_t *clockSourceFrequency*, uint32_t *desiredSpiClock*, uint16_t *msbFirst*, uint16_t *clockPhase*, uint16_t *clockPolarity*, uint16_t *spiMode*)

DEPRECATED - Initializes the SPI Master block.

Upon successful initialization of the SPI master block, this function will have set the bus speed for the master, but the SPI Master block still remains disabled and must be enabled with [EUSCI_B_SPI_enable\(\)](#)

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	106
TI Compiler 4.2.1	Size	118
TI Compiler 4.2.1	Speed	118
IAR 5.51.6	None	108
IAR 5.51.6	Size	102
IAR 5.51.6	Speed	102
MSPGCC 4.8.0	None	126
MSPGCC 4.8.0	Size	94
MSPGCC 4.8.0	Speed	82

Parameters:

baseAddress is the base address of the EUSCI_B_SPI Master module.

selectClockSource selects Clock source. Valid values are:

- EUSCI_B_SPI_CLOCKSOURCE_ACLK
- EUSCI_B_SPI_CLOCKSOURCE_SMCLK

clockSourceFrequency is the frequency of the selected clock source

desiredSpiClock is the desired clock rate for SPI communication

msbFirst controls the direction of the receive and transmit shift register. Valid values are:

- EUSCI_B_SPI_MSB_FIRST
- EUSCI_B_SPI_LSB_FIRST [Default]

clockPhase is clock phase select. Valid values are:

- EUSCI_B_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT [Default]
- EUSCI_B_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT

clockPolarity is clock polarity select Valid values are:

- EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH
- EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_LOW [Default]

spiMode is SPI mode select Valid values are:

- EUSCI_B_SPI_3PIN
- EUSCI_B_SPI_4PIN_UCxSTE_ACTIVE_HIGH
- EUSCI_B_SPI_4PIN_UCxSTE_ACTIVE_LOW

Modified bits are **UCCKPH**, **UCCKPL**, **UC7BIT**, **UCMSB**, **UCSSELx** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

STATUS_SUCCESS

14.2.2.16 uint8_t EUSCI_B_SPI_receiveData (uint16_t *baseAddress*)

Receives a byte that has been sent to the SPI Module.

This function reads a byte of data from the SPI receive data Register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Returns:

Returns the byte received from by the SPI module, cast as an uint8_t.

14.2.2.17 void EUSCI_B_SPI_select4PinFunctionality (uint16_t *baseAddress*, uint8_t *select4PinFunctionality*)

Selects 4Pin Functionality.

This function should be invoked only in 4-wire mode. Invoking this function has no effect in 3-wire mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	14
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	50
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

select4PinFunctionality selects 4 pin functionality Valid values are:

- EUSCI_B_SPI_PREVENT_CONFLICTS_WITH_OTHER_MASTERS
- EUSCI_B_SPI_ENABLE_SIGNAL_FOR_4WIRE_SLAVE

Modified bits are **UCSTEM** of **UCAxCTLW0** register.

Returns:

None

14.2.2.18 void EUSCI_B_SPI_slaveInit (uint16_t *baseAddress*, uint16_t *msbFirst*, uint16_t *clockPhase*, uint16_t *clockPolarity*, uint16_t *spiMode*)

DEPRECATED - Initializes the SPI Slave block.

Upon successful initialization of the SPI slave block, this function will have initialized the slave block, but the SPI Slave block still remains disabled and must be enabled with [EUSCI_B_SPI_enable\(\)](#)

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	82
TI Compiler 4.2.1	Size	30
TI Compiler 4.2.1	Speed	30
IAR 5.51.6	None	62
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

Parameters:

baseAddress is the base address of the EUSCI_B_SPI Slave module.

msbFirst controls the direction of the receive and transmit shift register. Valid values are:

- EUSCI_B_SPI_MSB_FIRST
- EUSCI_B_SPI_LSB_FIRST [Default]

clockPhase is clock phase select. Valid values are:

- EUSCI_B_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT [Default]
- EUSCI_B_SPI_PHASE_DATA_CAPTURED_ONFIRST_CHANGED_ON_NEXT

clockPolarity is clock polarity select Valid values are:

- EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH
- EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_LOW [Default]

spiMode is SPI mode select Valid values are:

- EUSCI_B_SPI_3PIN
- EUSCI_B_SPI_4PIN_UCxSTE_ACTIVE_HIGH
- EUSCI_B_SPI_4PIN_UCxSTE_ACTIVE_LOW

Modified bits are **UCMSB**, **UCMST**, **UC7BIT**, **UCCKPL**, **UCCKPH**, **UCMODE** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

STATUS_SUCCESS

14.2.2.19 void EUSCI_B_SPI_transmitData (uint16_t *baseAddress*, uint8_t *transmitData*)

Transmits a byte from the SPI Module.

This function will place the supplied data into SPI transmit data register to start transmission.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	30
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:**baseAddress** is the base address of the EUSCI_B_SPI module.**transmitData** data to be transmitted from the SPI module**Returns:**

None

14.3 Programming Example

The following example shows how to use the SPI API to configure the SPI module as a master device, and how to do a simple send of data.

```
//Initialize slave to MSB first, inactive high clock polarity and 3 wire SPI
returnValue = EUSCI_B_SPI_slaveInit(EUSCI_B0_BASE,
    EUSCI_B_SPI_MSB_FIRST,
    EUSCI_B_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT,
    EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH
);

if (STATUS_FAIL == returnValue){
    return;
}

//Enable SPI Module
EUSCI_B_SPI_enable(EUSCI_A0_BASE);

//Enable Receive interrupt
EUSCI_B_SPI_enableInterrupt(EUSCI_A0_BASE,
    EUSCI_B_SPI_RECEIVE_INTERRUPT
);
```

15 EUSCI Inter-Integrated Circuit (EUSCI_B_I2C)

Introduction	152
API Functions	154
Programming Example	177

15.1 Introduction

In I2C mode, the eUSCI_B module provides an interface between the device and I2C-compatible devices connected by the two-wire I2C serial bus. External components attached to the I2C bus serially transmit and/or receive serial data to/from the eUSCI_B module through the 2-wire I2C interface. The Inter-Integrated Circuit (I2C) API provides a set of functions for using the MSP430Ware I2C modules. Functions are provided to initialize the I2C modules, to send and receive data, obtain status, and to manage interrupts for the I2C modules.

The I2C module provide the ability to communicate to other IC devices over an I2C bus. The I2C bus is specified to support devices that can both transmit and receive (write and read) data. Also, devices on the I2C bus can be designated as either a master or a slave. The MSP430Ware I2C modules support both sending and receiving data as either a master or a slave, and also support the simultaneous operation as both a master and a slave.

I2C module can generate interrupts. The I2C module configured as a master will generate interrupts when a transmit or receive operation is completed (or aborted due to an error). The I2C module configured as a slave will generate interrupts when data has been sent or requested by a master.

15.1.1 Master Operations

To drive the master module, the APIs need to be invoked in the following order

- **EUSCI_B_I2C_masterInit**
- **EUSCI_B_I2C_setSlaveAddress**
- **EUSCI_B_I2C_setMode**
- **EUSCI_B_I2C_enable**
- **EUSCI_B_I2C_enableInterrupt** (if interrupts are being used) This may be followed by the APIs for transmit or receive as required

The user must first initialize the I2C module and configure it as a master with a call to [EUSCI_B_I2C_masterInit\(\)](#). That function will set the clock and data rates. This is followed by a call to set the slave address with which the master intends to communicate with using [EUSCI_B_I2C_setSlaveAddress](#). Then the mode of operation (transmit or receive) is chosen using [EUSCI_B_I2C_setMode](#). The I2C module may now be enabled using [EUSCI_B_I2C_enable](#). It is recommended to enable the EUSCI_B_I2C module before enabling the interrupts. Any transmission or reception of data may be initiated at this point after interrupts are enabled (if any).

The transaction can then be initiated on the bus by calling the transmit or receive related APIs as listed below.

Master Single Byte Transmission

- [EUSCI_B_I2C_masterSendSingleByte\(\)](#)

Master Multiple Byte Transmission

- [EUSCI_B_I2C_masterMultiByteSendStart\(\)](#)
- [EUSCI_B_I2C_masterMultiByteSendNext\(\)](#)
- [EUSCI_B_I2C_masterMultiByteSendStop\(\)](#)

Master Single Byte Reception

- [EUSCI_B_I2C_masterReceiveSingleByte\(\)](#)

Master Multiple Byte Reception

- EUSCI_B_I2C_masterMultiByteReceiveStart()
- EUSCI_B_I2C_masterMultiByteReceiveNext()
- EUSCI_B_I2C_masterMultiByteReceiveFinish()
- EUSCI_B_I2C_masterMultiByteReceiveStop()

For the interrupt-driven transaction, the user must register an interrupt handler for the I2C devices and enable the I2C interrupt.

15.1.2 Slave Operations

To drive the slave module, the APIs need to be invoked in the following order

- EUSCI_B_I2C_slaveInit()
- EUSCI_B_I2C_setMode()
- EUSCI_B_I2C_enable()
- EUSCI_B_I2C_enableInterrupt() (if interrupts are being used) This may be followed by the APIs for transmit or receive as required

The user must first call the EUSCI_B_I2C_slaveInit to initialize the slave module in I2C mode and set the slave address. This is followed by a call to set the mode of operation (transmit or receive).The I2C module may now be enabled using EUSCI_B_I2C_enable. It is recommended to enable the I2C module before enabling the interrupts. Any transmission or reception of data may be initiated at this point after interrupts are enabled (if any).

The transaction can then be initiated on the bus by calling the transmit or receive related APIs as listed below.

Slave Transmission API

- EUSCI_B_I2C_slaveDataPut()

Slave Reception API

- EUSCI_B_I2C_slaveDataGet()

For the interrupt-driven transaction, the user must register an interrupt handler for the I2C devices and enable the I2C interrupt.

This driver is contained in `eusci_b_i2c.c`, with `eusci_b_i2c.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	1918
TI Compiler 4.2.1	Size	1080
TI Compiler 4.2.1	Speed	1078
IAR 5.51.6	None	1508
IAR 5.51.6	Size	1090
IAR 5.51.6	Speed	1228
MSPGCC 4.8.0	None	2904
MSPGCC 4.8.0	Size	1224
MSPGCC 4.8.0	Speed	1214

15.2 API Functions

Functions

- void [EUSCI_B_I2C_clearInterruptFlag](#) (uint16_t baseAddress, uint16_t mask)
- void [EUSCI_B_I2C_disable](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_disableInterrupt](#) (uint16_t baseAddress, uint16_t mask)
- void [EUSCI_B_I2C_disableMultiMasterMode](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_enable](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_enableInterrupt](#) (uint16_t baseAddress, uint16_t mask)
- void [EUSCI_B_I2C_enableMultiMasterMode](#) (uint16_t baseAddress)
- uint16_t [EUSCI_B_I2C_getInterruptStatus](#) (uint16_t baseAddress, uint16_t mask)
- uint8_t [EUSCI_B_I2C_getMode](#) (uint16_t baseAddress)
- uint32_t [EUSCI_B_I2C_getReceiveBufferAddress](#) (uint16_t baseAddress)
- uint32_t [EUSCI_B_I2C_getTransmitBufferAddress](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_initMaster](#) (uint16_t baseAddress, EUSCI_B_I2C_initMasterParam *param)
- void [EUSCI_B_I2C_initSlave](#) (uint16_t baseAddress, EUSCI_B_I2C_initSlaveParam *param)
- uint16_t [EUSCI_B_I2C_isBusBusy](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_masterInit](#) (uint16_t baseAddress, uint8_t selectClockSource, uint32_t i2cClk, uint32_t dataRate, uint8_t byteCounterThreshold, uint8_t autoSTOPGeneration)
- uint16_t [EUSCI_B_I2C_masterIsStartSent](#) (uint16_t baseAddress)
- uint16_t [EUSCI_B_I2C_masterIsStopSent](#) (uint16_t baseAddress)
- uint8_t [EUSCI_B_I2C_masterMultiByteReceiveFinish](#) (uint16_t baseAddress)
- bool [EUSCI_B_I2C_masterMultiByteReceiveFinishWithTimeout](#) (uint16_t baseAddress, uint8_t *txData, uint32_t timeout)
- uint8_t [EUSCI_B_I2C_masterMultiByteReceiveNext](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_masterMultiByteReceiveStop](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_masterMultiByteSendFinish](#) (uint16_t baseAddress, uint8_t txData)
- bool [EUSCI_B_I2C_masterMultiByteSendFinishWithTimeout](#) (uint16_t baseAddress, uint8_t txData, uint32_t timeout)
- void [EUSCI_B_I2C_masterMultiByteSendNext](#) (uint16_t baseAddress, uint8_t txData)
- bool [EUSCI_B_I2C_masterMultiByteSendNextWithTimeout](#) (uint16_t baseAddress, uint8_t txData, uint32_t timeout)
- void [EUSCI_B_I2C_masterMultiByteSendStart](#) (uint16_t baseAddress, uint8_t txData)
- bool [EUSCI_B_I2C_masterMultiByteSendStartWithTimeout](#) (uint16_t baseAddress, uint8_t txData, uint32_t timeout)
- void [EUSCI_B_I2C_masterMultiByteSendStop](#) (uint16_t baseAddress)
- bool [EUSCI_B_I2C_masterMultiByteSendStopWithTimeout](#) (uint16_t baseAddress, uint32_t timeout)
- uint8_t [EUSCI_B_I2C_masterReceiveSingleByte](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_masterReceiveStart](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_masterSendSingleByte](#) (uint16_t baseAddress, uint8_t txData)
- bool [EUSCI_B_I2C_masterSendSingleByteWithTimeout](#) (uint16_t baseAddress, uint8_t txData, uint32_t timeout)
- void [EUSCI_B_I2C_masterSendStart](#) (uint16_t baseAddress)
- uint8_t [EUSCI_B_I2C_masterSingleReceive](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_setMode](#) (uint16_t baseAddress, uint8_t mode)
- void [EUSCI_B_I2C_setSlaveAddress](#) (uint16_t baseAddress, uint8_t slaveAddress)
- uint8_t [EUSCI_B_I2C_slaveDataGet](#) (uint16_t baseAddress)
- void [EUSCI_B_I2C_slaveDataPut](#) (uint16_t baseAddress, uint8_t transmitData)
- void [EUSCI_B_I2C_slaveInit](#) (uint16_t baseAddress, uint8_t slaveAddress, uint8_t slaveAddressOffset, uint32_t slaveOwnAddressEnable)

15.2.1 Detailed Description

The eUSCI I2C API is broken into three groups of functions: those that deal with interrupts, those that handle status and initialization, and those that deal with sending and receiving data.

The I2C master and slave interrupts are handled by

- EUSCI_B_I2C_enableInterrupt
- EUSCI_B_I2C_disableInterrupt
- EUSCI_B_I2C_clearInterruptFlag
- EUSCI_B_I2C_getInterruptStatus

Status and initialization functions for the I2C modules are

- EUSCI_B_I2C_masterInit
- EUSCI_B_I2C_enable
- EUSCI_B_I2C_disable
- EUSCI_B_I2C_isBusBusy
- EUSCI_B_I2C_isBusy
- EUSCI_B_I2C_slaveInit
- EUSCI_B_I2C_interruptStatus
- EUSCI_B_I2C_setSlaveAddress
- EUSCI_B_I2C_setMode
- EUSCI_B_I2C_masterIsStopSent
- EUSCI_B_I2C_masterIsStartSent
- EUSCI_B_I2C_selectMasterEnvironmentSelect

Sending and receiving data from the I2C slave module is handled by

- EUSCI_B_I2C_slaveDataPut
- EUSCI_B_I2C_slaveDataGet

Sending and receiving data from the I2C slave module is handled by

- EUSCI_B_I2C_masterSendSingleByte
- EUSCI_B_I2C_masterSendStart
- EUSCI_B_I2C_masterMultiByteSendStart
- EUSCI_B_I2C_masterMultiByteSendNext
- EUSCI_B_I2C_masterMultiByteSendFinish
- EUSCI_B_I2C_masterMultiByteSendStop
- EUSCI_B_I2C_masterMultiByteReceiveNext
- EUSCI_B_I2C_masterMultiByteReceiveFinish
- EUSCI_B_I2C_masterMultiByteReceiveStop
- EUSCI_B_I2C_masterReceiveStart
- EUSCI_B_I2C_masterSingleReceive
- EUSCI_B_I2C_getReceiveBufferAddressForDMA
- EUSCI_B_I2C_getTransmitBufferAddressForDMA

DMA related

- EUSCI_B_I2C_getReceiveBufferAddressForDMA
- EUSCI_B_I2C_getTransmitBufferAddressForDMA

15.2.2 Function Documentation

15.2.2.1 void EUSCI_B_I2C_clearInterruptFlag (uint16_t *baseAddress*, uint16_t *mask*)

Clears I2C interrupt sources.

The I2C interrupt source is cleared, so that it no longer asserts. The highest interrupt flag is automatically cleared when an interrupt vector generator is used.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	12
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the I2C module.

mask is a bit mask of the interrupt sources to be cleared. Mask value is the logical OR of any of the following:

- **EUSCI_B_I2C_NAK_INTERRUPT** - Not-acknowledge interrupt
- **EUSCI_B_I2C_ARBITRATIONLOST_INTERRUPT** - Arbitration lost interrupt
- **EUSCI_B_I2C_STOP_INTERRUPT** - STOP condition interrupt
- **EUSCI_B_I2C_START_INTERRUPT** - START condition interrupt
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT0** - Transmit interrupt0
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT1** - Transmit interrupt1
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT2** - Transmit interrupt2
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT3** - Transmit interrupt3
- **EUSCI_B_I2C_RECEIVE_INTERRUPT0** - Receive interrupt0
- **EUSCI_B_I2C_RECEIVE_INTERRUPT1** - Receive interrupt1
- **EUSCI_B_I2C_RECEIVE_INTERRUPT2** - Receive interrupt2
- **EUSCI_B_I2C_RECEIVE_INTERRUPT3** - Receive interrupt3
- **EUSCI_B_I2C_BIT9_POSITION_INTERRUPT** - Bit position 9 interrupt
- **EUSCI_B_I2C_CLOCK_LOW_TIMEOUT_INTERRUPT** - Clock low timeout interrupt enable
- **EUSCI_B_I2C_BYTE_COUNTER_INTERRUPT** - Byte counter interrupt enable

Modified bits of **UCBxIFG** register.

Returns:

None

15.2.2.2 void EUSCI_B_I2C_disable (uint16_t *baseAddress*)

Disables the I2C block.

This will disable operation of the I2C block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the USCI I2C module.

Modified bits are **UCSWRST** of **UCBxCTLW0** register.

Returns:

None

15.2.2.3 void EUSCI_B_I2C_disableInterrupt (uint16_t baseAddress, uint16_t mask)

Disables individual I2C interrupt sources.

Disables the indicated I2C interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	12
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the I2C module.

mask is the bit mask of the interrupt sources to be disabled. Mask value is the logical OR of any of the following:

- **EUSCI_B_I2C_NAK_INTERRUPT** - Not-acknowledge interrupt
- **EUSCI_B_I2C_ARBITRATIONLOST_INTERRUPT** - Arbitration lost interrupt
- **EUSCI_B_I2C_STOP_INTERRUPT** - STOP condition interrupt
- **EUSCI_B_I2C_START_INTERRUPT** - START condition interrupt
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT0** - Transmit interrupt0
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT1** - Transmit interrupt1
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT2** - Transmit interrupt2
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT3** - Transmit interrupt3
- **EUSCI_B_I2C_RECEIVE_INTERRUPT0** - Receive interrupt0
- **EUSCI_B_I2C_RECEIVE_INTERRUPT1** - Receive interrupt1
- **EUSCI_B_I2C_RECEIVE_INTERRUPT2** - Receive interrupt2
- **EUSCI_B_I2C_RECEIVE_INTERRUPT3** - Receive interrupt3
- **EUSCI_B_I2C_BIT9_POSITION_INTERRUPT** - Bit position 9 interrupt
- **EUSCI_B_I2C_CLOCK_LOW_TIMEOUT_INTERRUPT** - Clock low timeout interrupt enable

■ EUSCI_B_I2C_BYTE_COUNTER_INTERRUPT - Byte counter interrupt enable

Modified bits of **UCBxIE** register.

Returns:
None

15.2.2.4 void EUSCI_B_I2C_disableMultiMasterMode (uint16_t *baseAddress*)

Disables Multi Master Mode.

At the end of this function, the I2C module is still disabled till EUSCI_B_I2C_enable is invoked

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	16
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the I2C module.

Modified bits are **UCSWRST** and **UCMM** of **UCBxCTLW0** register.

Returns:
None

15.2.2.5 void EUSCI_B_I2C_enable (uint16_t *baseAddress*)

Enables the I2C block.

This will enable operation of the I2C block.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the USCI I2C module.

Modified bits are **UCSWRST** of **UCBxCTLW0** register.

Returns:
None

15.2.2.6 void EUSCI_B_I2C_enableInterrupt (uint16_t *baseAddress*, uint16_t *mask*)

Enables individual I2C interrupt sources.

Enables the indicated I2C interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	12
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the I2C module.

mask is the bit mask of the interrupt sources to be enabled. Mask value is the logical OR of any of the following:

- **EUSCI_B_I2C_NAK_INTERRUPT** - Not-acknowledge interrupt
- **EUSCI_B_I2C_ARBITRATIONLOST_INTERRUPT** - Arbitration lost interrupt
- **EUSCI_B_I2C_STOP_INTERRUPT** - STOP condition interrupt
- **EUSCI_B_I2C_START_INTERRUPT** - START condition interrupt
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT0** - Transmit interrupt0
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT1** - Transmit interrupt1
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT2** - Transmit interrupt2
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT3** - Transmit interrupt3
- **EUSCI_B_I2C_RECEIVE_INTERRUPT0** - Receive interrupt0
- **EUSCI_B_I2C_RECEIVE_INTERRUPT1** - Receive interrupt1
- **EUSCI_B_I2C_RECEIVE_INTERRUPT2** - Receive interrupt2
- **EUSCI_B_I2C_RECEIVE_INTERRUPT3** - Receive interrupt3
- **EUSCI_B_I2C_BIT9_POSITION_INTERRUPT** - Bit position 9 interrupt
- **EUSCI_B_I2C_CLOCK_LOW_TIMEOUT_INTERRUPT** - Clock low timeout interrupt enable
- **EUSCI_B_I2C_BYTE_COUNTER_INTERRUPT** - Byte counter interrupt enable

Modified bits of **UCBxIE** register.

Returns:
None

15.2.2.7 void EUSCI_B_I2C_enableMultiMasterMode (uint16_t *baseAddress*)

Enables Multi Master Mode.

At the end of this function, the I2C module is still disabled till EUSCI_B_I2C_enable is invoked

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	16
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the I2C module.

Modified bits are **UCSWRST** and **UCMM** of **UCBxCTLW0** register.

Returns:

None

15.2.2.8 uint16_t EUSCI_B_I2C_getInterruptStatus (uint16_t baseAddress, uint16_t mask)

Gets the current I2C interrupt status.

This returns the interrupt status for the I2C module based on which flag is passed.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the I2C module.

mask is the masked interrupt flag status to be returned. Mask value is the logical OR of any of the following:

- **EUSCI_B_I2C_NAK_INTERRUPT** - Not-acknowledge interrupt
- **EUSCI_B_I2C_ARBITRATIONLOST_INTERRUPT** - Arbitration lost interrupt
- **EUSCI_B_I2C_STOP_INTERRUPT** - STOP condition interrupt
- **EUSCI_B_I2C_START_INTERRUPT** - START condition interrupt
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT0** - Transmit interrupt0
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT1** - Transmit interrupt1
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT2** - Transmit interrupt2
- **EUSCI_B_I2C_TRANSMIT_INTERRUPT3** - Transmit interrupt3
- **EUSCI_B_I2C_RECEIVE_INTERRUPT0** - Receive interrupt0
- **EUSCI_B_I2C_RECEIVE_INTERRUPT1** - Receive interrupt1
- **EUSCI_B_I2C_RECEIVE_INTERRUPT2** - Receive interrupt2
- **EUSCI_B_I2C_RECEIVE_INTERRUPT3** - Receive interrupt3
- **EUSCI_B_I2C_BIT9_POSITION_INTERRUPT** - Bit position 9 interrupt
- **EUSCI_B_I2C_CLOCK_LOW_TIMEOUT_INTERRUPT** - Clock low timeout interrupt enable
- **EUSCI_B_I2C_BYTE_COUNTER_INTERRUPT** - Byte counter interrupt enable

Returns:

Logical OR of any of the following:

- **EUSCI_B_I2C_NAK_INTERRUPT** Not-acknowledge interrupt
 - **EUSCI_B_I2C_ARBITRATIONLOST_INTERRUPT** Arbitration lost interrupt
 - **EUSCI_B_I2C_STOP_INTERRUPT** STOP condition interrupt
 - **EUSCI_B_I2C_START_INTERRUPT** START condition interrupt
 - **EUSCI_B_I2C_TRANSMIT_INTERRUPT0** Transmit interrupt0
 - **EUSCI_B_I2C_TRANSMIT_INTERRUPT1** Transmit interrupt1
 - **EUSCI_B_I2C_TRANSMIT_INTERRUPT2** Transmit interrupt2
 - **EUSCI_B_I2C_TRANSMIT_INTERRUPT3** Transmit interrupt3
 - **EUSCI_B_I2C_RECEIVE_INTERRUPT0** Receive interrupt0
 - **EUSCI_B_I2C_RECEIVE_INTERRUPT1** Receive interrupt1
 - **EUSCI_B_I2C_RECEIVE_INTERRUPT2** Receive interrupt2
 - **EUSCI_B_I2C_RECEIVE_INTERRUPT3** Receive interrupt3
 - **EUSCI_B_I2C_BIT9_POSITION_INTERRUPT** Bit position 9 interrupt
 - **EUSCI_B_I2C_CLOCK_LOW_TIMEOUT_INTERRUPT** Clock low timeout interrupt enable
 - **EUSCI_B_I2C_BYTE_COUNTER_INTERRUPT** Byte counter interrupt enable
- indicating the status of the masked interrupts

15.2.2.9 uint8_t EUSCI_B_I2C_getMode (uint16_t baseAddress)

Gets the mode of the I2C device.

Current I2C transmit/receive mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the I2C module.

Modified bits are **UCTR** of **UCBxCTLW0** register.

Returns:

None Return one of the following:

- **EUSCI_B_I2C_TRANSMIT_MODE**
 - **EUSCI_B_I2C_RECEIVE_MODE**
- indicating the current mode

15.2.2.10 uint32_t EUSCI_B_I2C_getReceiveBufferAddress (uint16_t baseAddress)

Returns the address of the RX Buffer of the I2C for the DMA module.

Returns the address of the I2C RX Buffer. This can be used in conjunction with the DMA to store the received data directly to memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the I2C module.

Returns:

The address of the I2C RX Buffer

15.2.2.11 uint32_t EUSCI_B_I2C_getTransmitBufferAddress (uint16_t *baseAddress*)

Returns the address of the TX Buffer of the I2C for the DMA module.

Returns the address of the I2C TX Buffer. This can be used in conjunction with the DMA to obtain transmitted data directly from memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the I2C module.

Returns:

The address of the I2C TX Buffer

15.2.2.12 void EUSCI_B_I2C_initMaster (uint16_t *baseAddress*, EUSCI_B_I2C_initMasterParam * *param*)

Initializes the I2C Master block.

This function initializes operation of the I2C Master block. Upon successful initialization of the I2C block, this function will have set the bus speed for the master; however I2C module is still disabled till EUSCI_B_I2C_enable is invoked.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	122
TI Compiler 4.2.1	Size	78
TI Compiler 4.2.1	Speed	78
IAR 5.51.6	None	102
IAR 5.51.6	Size	88
IAR 5.51.6	Speed	88
MSPGCC 4.8.0	None	174
MSPGCC 4.8.0	Size	76
MSPGCC 4.8.0	Speed	76

Parameters:

baseAddress is the base address of the I2C Master module.

param is the pointer to the struct for master initialization.

Returns:

None

15.2.2.13 void EUSCI_B_I2C_initSlave (uint16_t baseAddress, EUSCI_B_I2C_initSlaveParam * param)

Initializes the I2C Slave block.

This function initializes operation of the I2C as a Slave mode. Upon successful initialization of the I2C blocks, this function will have set the slave address but the I2C module is still disabled till EUSCI_B_I2C_enable is invoked.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	56
TI Compiler 4.2.1	Size	36
TI Compiler 4.2.1	Speed	34
IAR 5.51.6	None	52
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	106
MSPGCC 4.8.0	Size	34
MSPGCC 4.8.0	Speed	34

Parameters:

baseAddress is the base address of the I2C Slave module.

param is the pointer to the struct for slave initialization.

Returns:

None

15.2.2.14 uint16_t EUSCI_B_I2C_isBusBusy (uint16_t baseAddress)

Indicates whether or not the I2C bus is busy.

This function returns an indication of whether or not the I2C bus is busy. This function checks the status of the bus via UCBBUSY bit in UCBxSTAT register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	20
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the I2C module.

Returns:

One of the following:

- **EUSCI_B_I2C_BUS_BUSY**
- **EUSCI_B_I2C_BUS_NOT_BUSY**
indicating whether the bus is busy

15.2.2.15 void EUSCI_B_I2C_masterInit (uint16_t *baseAddress*, uint8_t *selectClockSource*, uint32_t *i2cClk*, uint32_t *dataRate*, uint8_t *byteCounterThreshold*, uint8_t *autoSTOPGeneration*)

DEPRECATED - Initializes the I2C Master block.

This function initializes operation of the I2C Master block. Upon successful initialization of the I2C block, this function will have set the bus speed for the master; however I2C module is still disabled till EUSCI_B_I2C_enable is invoked.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	94
TI Compiler 4.2.1	Size	92
TI Compiler 4.2.1	Speed	92
IAR 5.51.6	None	88
IAR 5.51.6	Size	86
IAR 5.51.6	Speed	86
MSPGCC 4.8.0	None	102
MSPGCC 4.8.0	Size	82
MSPGCC 4.8.0	Speed	72

Parameters:

baseAddress is the base address of the I2C Master module.

selectClockSource is the clocksource. Valid values are:

- **EUSCI_B_I2C_CLOCKSOURCE_ACLK**
- **EUSCI_B_I2C_CLOCKSOURCE_SMCLK**

i2cClk is the rate of the clock supplied to the I2C module (the frequency in Hz of the clock source specified in selectClockSource).

dataRate setup for selecting data transfer rate. Valid values are:

- **EUSCI_B_I2C_SET_DATA_RATE_400KBPS**
- **EUSCI_B_I2C_SET_DATA_RATE_100KBPS**

byteCounterThreshold sets threshold for automatic STOP or UCSTPIFG

autoSTOPGeneration sets up the STOP condition generation. Valid values are:

- **EUSCI_B_I2C_NO_AUTO_STOP**
- **EUSCI_B_I2C_SET_BYTECOUNT_THRESHOLD_FLAG**

■ EUSCI_B_I2C_SEND_STOP_AUTOMATICALLY_ON_BYTECOUNT_THRESHOLD

Returns:

None

15.2.2.16 uint16_t EUSCI_B_I2C_masterIsStartSent (uint16_t *baseAddress*)

Indicates whether Start got sent.

This function returns an indication of whether or not Start got sent This function checks the status of the bus via UCTXSTT bit in UCBxCTL1 register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	16
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the I2C Master module.

Returns:

One of the following:

- EUSCI_B_I2C_START_SEND_COMPLETE
 - EUSCI_B_I2C_SENDING_START
- indicating whether the start was sent

15.2.2.17 uint16_t EUSCI_B_I2C_masterIsStopSent (uint16_t *baseAddress*)

Indicates whether STOP got sent.

This function returns an indication of whether or not STOP got sent This function checks the status of the bus via UCTXSTP bit in UCBxCTL1 register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	16
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the I2C Master module.

Returns:

One of the following:

- **EUSCI_B_I2C_STOP_SEND_COMPLETE**
- **EUSCI_B_I2C_SENDING_STOP**
indicating whether the stop was sent

15.2.2.18 uint8_t EUSCI_B_I2C_masterMultiByteReceiveFinish (uint16_t *baseAddress*)

Finishes multi-byte reception at the Master end.

This function is used by the Master module to initiate completion of a multi-byte reception. This function receives the current byte and initiates the STOP from master to slave.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	42
TI Compiler 4.2.1	Size	26
TI Compiler 4.2.1	Speed	26
IAR 5.51.6	None	34
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	38

Parameters:

baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTP** of **UCBxCTLW0** register.

Returns:

Received byte at Master end.

15.2.2.19 bool EUSCI_B_I2C_masterMultiByteReceiveFinishWithTimeout (uint16_t *baseAddress*, uint8_t * *txData*, uint32_t *timeout*)

Finishes multi-byte reception at the Master end with timeout.

This function is used by the Master module to initiate completion of a multi-byte reception. This function receives the current byte and initiates the STOP from master to slave.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	136
TI Compiler 4.2.1	Size	76
TI Compiler 4.2.1	Speed	76
IAR 5.51.6	None	96
IAR 5.51.6	Size	94
IAR 5.51.6	Speed	94
MSPGCC 4.8.0	None	178
MSPGCC 4.8.0	Size	100
MSPGCC 4.8.0	Speed	100

Parameters:

baseAddress is the base address of the I2C Master module.

txData is a pointer to the location to store the received byte at master end

timeout is the amount of time to wait until giving up

Modified bits are **UCTXSTP** of **UCBxCTLW0** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the reception process

15.2.2.20 uint8_t EUSCI_B_I2C_masterMultiByteReceiveNext (uint16_t baseAddress)

Starts multi-byte reception at the Master end one byte at a time.

This function is used by the Master module to receive each byte of a multi- byte reception. This function reads currently received byte.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the I2C Master module.

Returns:

Received byte at Master end.

15.2.2.21 void EUSCI_B_I2C_masterMultiByteReceiveStop (uint16_t baseAddress)

Sends the STOP at the end of a multi-byte reception at the Master end.

This function is used by the Master module to initiate STOP

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTP** of **UCBxCTLW0** register.

Returns:
None

15.2.2.22 void EUSCI_B_I2C_masterMultiByteSendFinish (uint16_t *baseAddress*, uint8_t *txData*)

Finishes multi-byte transmission from Master to Slave.

This function is used by the Master module to send the last byte and STOP. This function transmits the last data byte of a multi-byte transmission to the slave and then sends a stop.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	48
TI Compiler 4.2.1	Size	30
TI Compiler 4.2.1	Speed	30
IAR 5.51.6	None	56
IAR 5.51.6	Size	44
IAR 5.51.6	Speed	44
MSPGCC 4.8.0	None	98
MSPGCC 4.8.0	Size	38
MSPGCC 4.8.0	Speed	38

Parameters:

baseAddress is the base address of the I2C Master module.

txData is the last data byte to be transmitted in a multi-byte transmission

Modified bits of **UCBxTXBUF** register and bits of **UCBxCTLW0** register.

Returns:
None

15.2.2.23 bool EUSCI_B_I2C_masterMultiByteSendFinishWithTimeout (uint16_t *baseAddress*, uint8_t *txData*, uint32_t *timeout*)

Finishes multi-byte transmission from Master to Slave with timeout.

This function is used by the Master module to send the last byte and STOP. This function transmits the last data byte of a multi-byte transmission to the slave and then sends a stop.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	138
TI Compiler 4.2.1	Size	84
TI Compiler 4.2.1	Speed	84
IAR 5.51.6	None	116
IAR 5.51.6	Size	106
IAR 5.51.6	Speed	106
MSPGCC 4.8.0	None	196
MSPGCC 4.8.0	Size	94
MSPGCC 4.8.0	Speed	96

Parameters:

baseAddress is the base address of the I2C Master module.

txData is the last data byte to be transmitted in a multi-byte transmission

timeout is the amount of time to wait until giving up

Modified bits of **UCBxTXBUF** register and bits of **UCBxCTLW0** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the transmission process.

15.2.2.24 void EUSCI_B_I2C_masterMultiByteSendNext (uint16_t *baseAddress*, uint8_t *txData*)

Continues multi-byte transmission from Master to Slave.

This function is used by the Master module continue each byte of a multi- byte transmission. This function transmits each data byte of a multi-byte transmission to the slave.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	38
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

Parameters:

baseAddress is the base address of the I2C Master module.

txData is the next data byte to be transmitted

Modified bits of **UCBxTXBUF** register.

Returns:

None

15.2.2.25 bool EUSCI_B_I2C_masterMultiByteSendNextWithTimeout (uint16_t *baseAddress*, uint8_t *txData*, uint32_t *timeout*)

Continues multi-byte transmission from Master to Slave with timeout.

This function is used by the Master module continue each byte of a multi- byte transmission. This function transmits each data byte of a multi-byte transmission to the slave.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	76
TI Compiler 4.2.1	Size	46
TI Compiler 4.2.1	Speed	46
IAR 5.51.6	None	70
IAR 5.51.6	Size	60
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	110
MSPGCC 4.8.0	Size	64
MSPGCC 4.8.0	Speed	64

Parameters:

baseAddress is the base address of the I2C Master module.

txData is the next data byte to be transmitted

timeout is the amount of time to wait until giving up

Modified bits of **UCBxTXBUF** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the transmission process.

15.2.2.26 void EUSCI_B_I2C_masterMultiByteSendStart (uint16_t baseAddress, uint8_t txData)

Starts multi-byte transmission from Master to Slave.

This function is used by the master module to start a multi byte transaction.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	74
TI Compiler 4.2.1	Size	38
TI Compiler 4.2.1	Speed	38
IAR 5.51.6	None	62
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	52
MSPGCC 4.8.0	None	138
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

Parameters:

baseAddress is the base address of the I2C Master module.

txData is the first data byte to be transmitted

Modified bits of **UCBxTXBUF** register, bits of **UCBxCTLW0** register, bits of **UCBxIE** register and bits of **UCBxIFG** register.

Returns:

None

15.2.2.27 bool EUSCI_B_I2C_masterMultiByteSendStartWithTimeout (uint16_t baseAddress, uint8_t txData, uint32_t timeout)

Starts multi-byte transmission from Master to Slave with timeout.

This function is used by the master module to start a multi byte transaction.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	116
TI Compiler 4.2.1	Size	68
TI Compiler 4.2.1	Speed	68
IAR 5.51.6	None	94
IAR 5.51.6	Size	68
IAR 5.51.6	Speed	80
MSPGCC 4.8.0	None	184
MSPGCC 4.8.0	Size	84
MSPGCC 4.8.0	Speed	82

Parameters:

baseAddress is the base address of the I2C Master module.

txData is the first data byte to be transmitted

timeout is the amount of time to wait until giving up

Modified bits of **UCBxTXBUF** register, bits of **UCBxCTLW0** register, bits of **UCBxIE** register and bits of **UCBxIFG** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the transmission process.

15.2.2.28 void EUSCI_B_I2C_masterMultiByteSendStop (uint16_t baseAddress)

Send STOP byte at the end of a multi-byte transmission from Master to Slave.

This function is used by the Master module send STOP at the end of a multi- byte transmission. This function sends a stop after current transmission is complete.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	32
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	52
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Parameters:

baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTP** of **UCBxCTLW0** register.

Returns:

None

15.2.2.29 bool EUSCI_B_I2C_masterMultiByteSendStopWithTimeout (uint16_t baseAddress, uint32_t timeout)

Send STOP byte at the end of a multi-byte transmission from Master to Slave with timeout.

This function is used by the Master module send STOP at the end of a multi- byte transmission. This function sends a stop after current transmission is complete.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	74
TI Compiler 4.2.1	Size	44
TI Compiler 4.2.1	Speed	44
IAR 5.51.6	None	64
IAR 5.51.6	Size	46
IAR 5.51.6	Speed	58
MSPGCC 4.8.0	None	110
MSPGCC 4.8.0	Size	56
MSPGCC 4.8.0	Speed	52

Parameters:

baseAddress is the base address of the I2C Master module.

timeout is the amount of time to wait until giving up

Modified bits are **UCTXSTP** of **UCBxCTLW0** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the transmission process.

15.2.2.30 uint8_t EUSCI_B_I2C_masterReceiveSingleByte (uint16_t baseAddress)

Does single byte reception from Slave.

This function is used by the Master module to receive a single byte. This function sends start and stop, waits for data reception and then receives the data from the slave

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	40
TI Compiler 4.2.1	Size	26
TI Compiler 4.2.1	Speed	26
IAR 5.51.6	None	34
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	68
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

Parameters:

baseAddress is the base address of the I2C Master module.

Modified bits of **UCBxTXBUF** register, bits of **UCBxCTLW0** register, bits of **UCBxIE** register and bits of **UCBxIFG** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the transmission process.

15.2.2.31 void EUSCI_B_I2C_masterReceiveStart (uint16_t baseAddress)

Starts reception at the Master end.

This function is used by the Master module initiate reception of a single byte. This function sends a start.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	16
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTT** of **UCBxCTLW0** register.

Returns:
None

15.2.2.32 void EUSCI_B_I2C_masterSendSingleByte (uint16_t *baseAddress*, uint8_t *txData*)

Does single byte transmission from Master to Slave.

This function is used by the Master module to send a single byte. This function sends a start, then transmits the byte to the slave and then sends a stop.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	98
TI Compiler 4.2.1	Size	52
TI Compiler 4.2.1	Speed	52
IAR 5.51.6	None	90
IAR 5.51.6	Size	52
IAR 5.51.6	Speed	68
MSPGCC 4.8.0	None	196
MSPGCC 4.8.0	Size	60
MSPGCC 4.8.0	Speed	60

Parameters:

baseAddress is the base address of the I2C Master module.

txData is the data byte to be transmitted

Modified bits of **UCBxTXBUF** register, bits of **UCBxCTLW0** register, bits of **UCBxIE** register and bits of **UCBxIFG** register.

Returns:
None

15.2.2.33 bool EUSCI_B_I2C_masterSendSingleByteWithTimeout (uint16_t *baseAddress*, uint8_t *txData*, uint32_t *timeout*)

Does single byte transmission from Master to Slave with timeout.

This function is used by the Master module to send a single byte. This function sends a start, then transmits the byte to the slave and then sends a stop.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	184
TI Compiler 4.2.1	Size	106
TI Compiler 4.2.1	Speed	106
IAR 5.51.6	None	150
IAR 5.51.6	Size	116
IAR 5.51.6	Speed	122
MSPGCC 4.8.0	None	290
MSPGCC 4.8.0	Size	118
MSPGCC 4.8.0	Speed	118

Parameters:

baseAddress is the base address of the I2C Master module.

txData is the data byte to be transmitted

timeout is the amount of time to wait until giving up

Modified bits of **UCBxTXBUF** register, bits of **UCBxCTLW0** register, bits of **UCBxIE** register and bits of **UCBxIFG** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the transmission process.

15.2.2.34 void EUSCI_B_I2C_masterSendStart (uint16_t baseAddress)

This function is used by the Master module to initiate START.

This function is used by the Master module to initiate START

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTT** of **UCBxCTLW0** register.

Returns:

None

15.2.2.35 uint8_t EUSCI_B_I2C_masterSingleReceive (uint16_t baseAddress)

receives a byte that has been sent to the I2C Master Module.

This function reads a byte of data from the I2C receive data Register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	32
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	54
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Parameters:

baseAddress is the base address of the I2C Master module.

Returns:

Returns the byte received from by the I2C module, cast as an uint8_t.

15.2.2.36 void EUSCI_B_I2C_setMode (uint16_t baseAddress, uint8_t mode)

Sets the mode of the I2C device.

When the receive parameter is set to EUSCI_B_I2C_TRANSMIT_MODE, the address will indicate that the I2C module is in receive mode; otherwise, the I2C module is in send mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	32
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	16
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	52
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

baseAddress is the base address of the USCI I2C module.

mode Mode for the EUSCI_B_I2C module Valid values are:

- EUSCI_B_I2C_TRANSMIT_MODE [Default]
- EUSCI_B_I2C_RECEIVE_MODE

Modified bits are **UCTR** of **UCBxCTLW0** register.

Returns:

None

15.2.2.37 void EUSCI_B_I2C_setSlaveAddress (uint16_t baseAddress, uint8_t slaveAddress)

Sets the address that the I2C Master will place on the bus.

This function will set the address that the I2C Master will place on the bus when initiating a transaction.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	30
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:**baseAddress** is the base address of the USCI I2C module.**slaveAddress** 7-bit slave addressModified bits of **UCBxI2CSA** register.**Returns:**

None

15.2.2.38 uint8_t EUSCI_B_I2C_slaveDataGet (uint16_t *baseAddress*)

Receives a byte that has been sent to the I2C Module.

This function reads a byte of data from the I2C receive data Register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:**baseAddress** is the base address of the I2C Slave module.**Returns:**

Returns the byte received from by the I2C module, cast as an uint8_t.

15.2.2.39 void EUSCI_B_I2C_slaveDataPut (uint16_t *baseAddress*, uint8_t *transmitData*)

Transmits a byte from the I2C Module.

This function will place the supplied data into I2C transmit data register to start transmission.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	30
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:**baseAddress** is the base address of the I2C Slave module.**transmitData** data to be transmitted from the I2C module

Modified bits of **UCBxTXBUF** register.

Returns:
None

15.2.2.40 void EUSCI_B_I2C_slaveInit (uint16_t *baseAddress*, uint8_t *slaveAddress*,
uint8_t *slaveAddressOffset*, uint32_t *slaveOwnAddressEnable*)

DEPRECATED - Initializes the I2C Slave block.

This function initializes operation of the I2C as a Slave mode. Upon successful initialization of the I2C blocks, this function will have set the slave address but the I2C module is still disabled till EUSCI_B_I2C_enable is invoked.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	82
TI Compiler 4.2.1	Size	36
TI Compiler 4.2.1	Speed	36
IAR 5.51.6	None	66
IAR 5.51.6	Size	58
IAR 5.51.6	Speed	58
MSPGCC 4.8.0	None	86
MSPGCC 4.8.0	Size	54
MSPGCC 4.8.0	Speed	52

Parameters:

baseAddress is the base address of the I2C Slave module.

slaveAddress 7-bit slave address

slaveAddressOffset Own address Offset referred to- 'x' value of UCBxI2COAx. Valid values are:

- EUSCI_B_I2C_OWN_ADDRESS_OFFSET0
- EUSCI_B_I2C_OWN_ADDRESS_OFFSET1
- EUSCI_B_I2C_OWN_ADDRESS_OFFSET2
- EUSCI_B_I2C_OWN_ADDRESS_OFFSET3

slaveOwnAddressEnable selects if the specified address is enabled or disabled. Valid values are:

- EUSCI_B_I2C_OWN_ADDRESS_DISABLE
- EUSCI_B_I2C_OWN_ADDRESS_ENABLE

Returns:
None

15.3 Programming Example

The following example shows how to use the I2C API to send data as a master.

```
//Initialize Slave
EUSCI_B_I2C_slaveInit(EUSCI_B_B0_BASE,
    0x48,
    EUSCI_B_I2C_OWN_ADDRESS_OFFSET0,
    EUSCI_B_I2C_OWN_ADDRESS_ENABLE);

EUSCI_B_I2C_enable(EUSCI_B0_BASE);

EUSCI_B_I2C_enableInterrupt(EUSCI_B0_BASE,
    EUSCI_B_I2C_TRANSMIT_INTERRUPT0 +
    EUSCI_B_I2C_STOP_INTERRUPT);
```

16 FRAM Controller

Introduction	178
API Functions	178
Programming Example	184

16.1 Introduction

FRAM memory is a non-volatile memory that reads and writes like standard SRAM. The MSP430 FRAM memory features include:

- Byte or word write access
- Automatic and programmable wait state control with independent wait state settings for access and cycle times
- Error Correction Code with bit error correction, extended bit error detection and flag indicators
- Cache for fast read
- Power control for disabling FRAM on non-usage

This driver is contained in `fram.c`, with `fram.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	396
TI Compiler 4.2.1	Size	164
TI Compiler 4.2.1	Speed	166
IAR 5.51.6	None	174
IAR 5.51.6	Size	134
IAR 5.51.6	Speed	370
MSPGCC 4.8.0	None	574
MSPGCC 4.8.0	Size	218
MSPGCC 4.8.0	Speed	320

16.2 API Functions

Functions

- void [FRAM_configureWaitStateControl](#) (uint8_t waitState)
- void [FRAM_delayPowerUpFromLPM](#) (uint8_t delayStatus)
- void [FRAM_disableInterrupt](#) (uint16_t interruptMask)
- void [FRAM_enableInterrupt](#) (uint8_t interruptMask)
- uint8_t [FRAM_getInterruptStatus](#) (uint16_t interruptFlagMask)
- void [FRAM_memoryFill32](#) (uint32_t value, uint32_t *framPtr, uint16_t count)
- void [FRAM_write16](#) (uint16_t *dataPtr, uint16_t *framPtr, uint16_t numberOfWords)
- void [FRAM_write32](#) (uint32_t *dataPtr, uint32_t *framPtr, uint16_t count)
- void [FRAM_write8](#) (uint8_t *dataPtr, uint8_t *framPtr, uint16_t numberOfBytes)

16.2.1 Detailed Description

FRAM_enableInterrupt enables selected FRAM interrupt sources.

FRAM_getInterruptStatus returns the status of the selected FRAM interrupt flags.

FRAM_disableInterrupt disables selected FRAM interrupt sources.

Depending on the kind of writes being performed to the FRAM, this library provides APIs for FRAM writes.

FRAM_write8 facilitates writing into the FRAM memory in byte format. FRAM_write16 facilitates writing into the FRAM memory in word format. FRAM_write32 facilitates writing into the FRAM memory in long format, pass by reference. FRAM_memoryFill32 facilitates writing into the FRAM memory in long format, pass by value.

The FRAM API is broken into 3 groups of functions: those that write into FRAM, those that handle interrupts, and those that configure the wait state and power-up delay after LPM.

FRAM writes are managed by

- [FRAM_write8\(\)](#)
- [FRAM_write16\(\)](#)
- [FRAM_write32\(\)](#)
- [FRAM_memoryFill32\(\)](#)

The FRAM interrupts are handled by

- [FRAM_enableInterrupt\(\)](#)
- [FRAM_getInterruptStatus\(\)](#)
- [FRAM_disableInterrupt\(\)](#)

The FRAM wait state and power-up delay after LPM are handled by

- [FRAM_configureWaitStateControl\(\)](#)
- [FRAM_delayPowerUpFromLPM\(\)](#)

16.2.2 Function Documentation

16.2.2.1 void FRAM_configureWaitStateControl (uint8_t *waitState*)

Configures the access time of the FRAM module.

Configures the access time of the FRAM module.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Parameters:

waitState defines the number of CPU cycles required for access time defined in the datasheet Valid values are:

- **FRAM_ACCESS_TIME_CYCLES_0**
- **FRAM_ACCESS_TIME_CYCLES_1**

- **FRAM_ACCESS_TIME_CYCLES_2**
- **FRAM_ACCESS_TIME_CYCLES_3**
- **FRAM_ACCESS_TIME_CYCLES_4**
- **FRAM_ACCESS_TIME_CYCLES_5**
- **FRAM_ACCESS_TIME_CYCLES_6**
- **FRAM_ACCESS_TIME_CYCLES_7**

Modified bits are **NWAITS** of **GCCTL0** register.

Returns:
None

16.2.2.2 void FRAM_delayPowerUpFromLPM (uint8_t *delayStatus*)

Configures when the FRAM module will power up after LPM exit.

Configures when the FRAM module will power up after LPM exit. The module can either wait until the first FRAM access to power up or power up immediately after leaving LPM. If FRAM power is disabled, a memory access will automatically insert wait states to ensure sufficient timing for the FRAM power-up and access.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	36
TI Compiler 4.2.1	Size	24
TI Compiler 4.2.1	Speed	24
IAR 5.51.6	None	24
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Parameters:

delayStatus chooses if FRAM should power up instantly with LPM exit or to wait until first FRAM access after LPM exit Valid values are:

- **FRAM_DELAY_FROM_LPM_ENABLE**
- **FRAM_DELAY_FROM_LPM_DISABLE**

Returns:
None

16.2.2.3 void FRAM_disableInterrupt (uint16_t *interruptMask*)

Disables selected FRAM interrupt sources.

Disables the indicated FRAM interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	32
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	20
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	54
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:

interruptMask is the bit mask of the memory buffer interrupt sources to be disabled. Mask value is the logical OR of any of the following:

- **FRAM_PUC_ON_UNCORRECTABLE_BIT** - Enable PUC reset if FRAM uncorrectable bit error detected.
- **FRAM_UNCORRECTABLE_BIT_INTERRUPT** - Interrupts when an uncorrectable bit error is detected.
- **FRAM_CORRECTABLE_BIT_INTERRUPT** - Interrupts when a correctable bit error is detected.
- **FRAM_ACCESS_TIME_ERROR_INTERRUPT** - Interrupts when an access time error occurs.

Returns:

None

16.2.2.4 void FRAM_enableInterrupt (uint8_t *interruptMask*)

Enables selected FRAM interrupt sources.

Enables the indicated FRAM interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. Does not clear interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	22
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	54
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Parameters:

interruptMask is the bit mask of the memory buffer interrupt sources to be disabled. Mask value is the logical OR of any of the following:

- **FRAM_PUC_ON_UNCORRECTABLE_BIT** - Enable PUC reset if FRAM uncorrectable bit error detected.
- **FRAM_UNCORRECTABLE_BIT_INTERRUPT** - Interrupts when an uncorrectable bit error is detected.
- **FRAM_CORRECTABLE_BIT_INTERRUPT** - Interrupts when a correctable bit error is detected.
- **FRAM_ACCESS_TIME_ERROR_INTERRUPT** - Interrupts when an access time error occurs.

Modified bits of **GCCTL0** register and bits of **FRCTL0** register.

Returns:

None

16.2.2.5 uint8_t FRAM_getInterruptStatus (uint16_t *interruptFlagMask*)

Returns the status of the selected FRAM interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

interruptFlagMask is a bit mask of the interrupt flags status to be returned. Mask value is the logical OR of any of the following:

- **FRAM_ACCESS_TIME_ERROR_FLAG** - Interrupt flag is set if a wrong setting for NPRECHG and NACCESS is set and FRAM access time is not hold.
- **FRAM_UNCORRECTABLE_BIT_FLAG** - Interrupt flag is set if an uncorrectable bit error has been detected in the FRAM memory error detection logic.
- **FRAM_CORRECTABLE_BIT_FLAG** - Interrupt flag is set if a correctable bit error has been detected and corrected in the FRAM memory error detection logic.

Returns:

Logical OR of any of the following:

- **FRAM_ACCESS_TIME_ERROR_FLAG** Interrupt flag is set if a wrong setting for NPRECHG and NACCESS is set and FRAM access time is not hold.
 - **FRAM_UNCORRECTABLE_BIT_FLAG** Interrupt flag is set if an uncorrectable bit error has been detected in the FRAM memory error detection logic.
 - **FRAM_CORRECTABLE_BIT_FLAG** Interrupt flag is set if a correctable bit error has been detected and corrected in the FRAM memory error detection logic.
- indicating the status of the masked flags

16.2.2.6 void FRAM_memoryFill32 (uint32_t *value*, uint32_t * *framPtr*, uint16_t *count*)

Write data into the fram memory in long format, pass by value.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	60
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	20
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	84
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Parameters:

value is the value to written to FRAM memory

framPtr is the pointer into which to write the data

count is the number of 32 bit addresses to fill

Returns:

None

16.2.2.7 void FRAM_write16 (uint16_t * *dataPtr*, uint16_t * *framPtr*, uint16_t *numberOfWords*)

Write data into the fram memory in word format.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	62
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	18
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	60
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	20

Parameters:

dataPtr is the pointer to the data to be written

framPtr is the pointer into which to write the data

numberOfWords is the number of words to be written

Returns:

None

16.2.2.8 void FRAM_write32 (uint32_t * *dataPtr*, uint32_t * *framPtr*, uint16_t *count*)

Write data into the fram memory in long format, pass by reference.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	68
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	24
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	80
MSPGCC 4.8.0	None	80
MSPGCC 4.8.0	Size	38
MSPGCC 4.8.0	Speed	26

Parameters:

dataPtr is the pointer to the data to be written

framPtr is the pointer into which to write the data

count is the number of 32 bit words to be written

Returns:

None

16.2.2.9 void FRAM_write8 (uint8_t * *dataPtr*, uint8_t * *framPtr*, uint16_t *numberOfBytes*)

Write data into the fram memory in byte format.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	62
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	18
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	60
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	156

Parameters:

dataPtr is the pointer to the data to be written

framPtr is the pointer into which to write the data

numberOfBytes is the number of bytes to be written

Returns:

None

16.3 Programming Example

The following example shows some FRAM operations using the APIs

```
//Writes the value of "data", 128 times to FRAM
FRAM_memoryFill32(FRAM_BASE,data,
                  (unsigned long *)FRAM_TEST_START,128);
```

17 GPIO

Introduction	185
API Functions	186
Programming Example	203

17.1 Introduction

The Digital I/O (GPIO) API provides a set of functions for using the MSP430Ware GPIO modules. Functions are provided to setup and enable use of input/output pins, setting them up with or without interrupts and those that access the pin value.

The digital I/O features include:

- Independently programmable individual I/Os
- Any combination of input or output
- Individually configurable P1 and P2 interrupts. Some devices may include additional port interrupts.
- Independent input and output data registers
- Individually configurable pullup or pulldown resistors

Devices within the family may have up to twelve digital I/O ports implemented (P1 to P11 and PJ). Most ports contain eight I/O lines; however, some ports may contain less (see the device-specific data sheet for ports available). Each I/O line is individually configurable for input or output direction, and each can be individually read or written. Each I/O line is individually configurable for pullup or pulldown resistors. PJ contains only four I/O lines.

Ports P1 and P2 always have interrupt capability. Each interrupt for the P1 and P2 I/O lines can be individually enabled and configured to provide an interrupt on a rising or falling edge of an input signal. All P1 I/O lines source a single interrupt vector P1IV, and all P2 I/O lines source a different, single interrupt vector P2IV. On some devices, additional ports with interrupt capability may be available (see the device-specific data sheet for details) and contain their own respective interrupt vectors. Individual ports can be accessed as byte-wide ports or can be combined into word-wide ports and accessed via word formats. Port pairs P1/P2, P3/P4, P5/P6, P7/P8, etc., are associated with the names PA, PB, PC, PD, etc., respectively. All port registers are handled in this manner with this naming convention except for the interrupt vector registers, P1IV and P2IV; that is, PAIV does not exist. When writing to port PA with word operations, all 16 bits are written to the port. When writing to the lower byte of the PA port using byte operations, the upper byte remains unchanged. Similarly, writing to the upper byte of the PA port using byte instructions leaves the lower byte unchanged. When writing to a port that contains less than the maximum number of bits possible, the unused bits are a "don't care". Ports PB, PC, PD, PE, and PF behave similarly.

Reading of the PA port using word operations causes all 16 bits to be transferred to the destination. Reading the lower or upper byte of the PA port (P1 or P2) and storing to memory using byte operations causes only the lower or upper byte to be transferred to the destination, respectively. Reading of the PA port and storing to a general-purpose register using byte operations causes the byte transferred to be written to the least significant byte of the register. The upper significant byte of the destination register is cleared automatically. Ports PB, PC, PD, PE, and PF behave similarly. When reading from ports that contain less than the maximum bits possible, unused bits are read as zeros (similarly for port PJ).

The GPIO pin may be configured as an I/O pin with [GPIO_setAsOutputPin\(\)](#), [GPIO_setAsInputPin\(\)](#), [GPIO_setAsInputPinWithPullDownresistor\(\)](#) or [GPIO_setAsInputPinWithPullUpresistor\(\)](#). The GPIO pin may instead be configured to operate in the Peripheral Module assigned function by configuring the GPIO using [GPIO_setAsPeripheralModuleFunctionOutputPin\(\)](#) or [GPIO_setAsPeripheralModuleFunctionInputPin\(\)](#).

This driver is contained in `gpio.c`, with `gpio.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	1304
TI Compiler 4.2.1	Size	492
TI Compiler 4.2.1	Speed	500
IAR 5.51.6	None	874
IAR 5.51.6	Size	380
IAR 5.51.6	Speed	704
MSPGCC 4.8.0	None	1878
MSPGCC 4.8.0	Size	586
MSPGCC 4.8.0	Speed	582

17.2 API Functions

Functions

- void [GPIO_clearInterruptFlag](#) (uint8_t selectedPort, uint16_t selectedPins)
- void [GPIO_disableInterrupt](#) (uint8_t selectedPort, uint16_t selectedPins)
- void [GPIO_enableInterrupt](#) (uint8_t selectedPort, uint16_t selectedPins)
- uint8_t [GPIO_getInputPinValue](#) (uint8_t selectedPort, uint16_t selectedPins)
- uint16_t [GPIO_getInterruptStatus](#) (uint8_t selectedPort, uint16_t selectedPins)
- void [GPIO_interruptEdgeSelect](#) (uint8_t selectedPort, uint16_t selectedPins, uint8_t edgeSelect)
- void [GPIO_setAsInputPin](#) (uint8_t selectedPort, uint16_t selectedPins)
- void [GPIO_setAsInputPinWithPullDownResistor](#) (uint8_t selectedPort, uint16_t selectedPins)
- void [GPIO_setAsInputPinWithPullUpResistor](#) (uint8_t selectedPort, uint16_t selectedPins)
- void [GPIO_setAsOutputPin](#) (uint8_t selectedPort, uint16_t selectedPins)
- void [GPIO_setAsPeripheralModuleFunctionInputPin](#) (uint8_t selectedPort, uint16_t selectedPins, uint8_t mode)
- void [GPIO_setAsPeripheralModuleFunctionOutputPin](#) (uint8_t selectedPort, uint16_t selectedPins, uint8_t mode)
- void [GPIO_setOutputHighOnPin](#) (uint8_t selectedPort, uint16_t selectedPins)
- void [GPIO_setOutputLowOnPin](#) (uint8_t selectedPort, uint16_t selectedPins)
- void [GPIO_toggleOutputOnPin](#) (uint8_t selectedPort, uint16_t selectedPins)

17.2.1 Detailed Description

The GPIO API is broken into three groups of functions: those that deal with configuring the GPIO pins, those that deal with interrupts, and those that access the pin value.

The GPIO pins are configured with

- [GPIO_setAsOutputPin\(\)](#)
- [GPIO_setAsInputPin\(\)](#)
- [GPIO_setAsInputPinWithPullDownresistor\(\)](#)
- [GPIO_setAsInputPinWithPullUpresistor\(\)](#)
- [GPIO_setAsPeripheralModuleFunctionOutputPin\(\)](#)
- [GPIO_setAsPeripheralModuleFunctionInputPin\(\)](#)

The GPIO interrupts are handled with

- [GPIO_enableInterrupt\(\)](#)
- [GPIO_disableInterrupt\(\)](#)
- [GPIO_clearInterruptFlag\(\)](#)
- [GPIO_getInterruptStatus\(\)](#)
- [GPIO_interruptEdgeSelect\(\)](#)

The GPIO pin state is accessed with

- [GPIO_setOutputHighOnPin\(\)](#)
- [GPIO_setOutputLowOnPin\(\)](#)
- [GPIO_toggleOutputOnPin\(\)](#)
- [GPIO_getInputPinValue\(\)](#)

17.2.2 Function Documentation

17.2.2.1 void GPIO_clearInterruptFlag (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function clears the interrupt flag on the selected pin.

This function clears the interrupt flag on the selected pin. Note that only Port 1, 2, A have this capability.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	60
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	36
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	80
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Parameters:

selectedPort is the selected port. Valid values are:

- **GPIO_PORT_P1**
- **GPIO_PORT_P2**
- **GPIO_PORT_PA**

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- **GPIO_PIN0**
- **GPIO_PIN1**
- **GPIO_PIN2**
- **GPIO_PIN3**
- **GPIO_PIN4**
- **GPIO_PIN5**
- **GPIO_PIN6**
- **GPIO_PIN7**
- **GPIO_PIN8**
- **GPIO_PIN9**
- **GPIO_PIN10**
- **GPIO_PIN11**
- **GPIO_PIN12**
- **GPIO_PIN13**
- **GPIO_PIN14**
- **GPIO_PIN15**

Modified bits of **PxIFG** register.

Returns:

None

17.2.2.2 void GPIO_disableInterrupt (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function disables the port interrupt on the selected pin.

This function disables the port interrupt on the selected pin. Note that only Port 1, 2, A have this capability.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	60
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	36
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	80
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_PA

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Modified bits of **PxIE** register.

Returns:

None

17.2.2.3 void GPIO_enableInterrupt (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function enables the port interrupt on the selected pin.

This function enables the port interrupt on the selected pin. Note that only Port 1, 2, A have this capability.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	60
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	36
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_PA

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Modified bits of **PxIE** register.

Returns:

None

17.2.2.4 uint8_t GPIO_getInputPinValue (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function gets the input value on the selected pin.

This function gets the input value on the selected pin.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	66
TI Compiler 4.2.1	Size	26
TI Compiler 4.2.1	Speed	26
IAR 5.51.6	None	46
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	78
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	36

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Valid values are:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Returns:

One of the following:

- GPIO_INPUT_PIN_HIGH
 - GPIO_INPUT_PIN_LOW
- indicating the status of the pin

17.2.2.5 uint16_t GPIO_getInterruptStatus (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function gets the interrupt status of the selected pin.

This function gets the interrupt status of the selected pin. Note that only Port 1, 2, A have this capability.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	56
TI Compiler 4.2.1	Size	24
TI Compiler 4.2.1	Speed	24
IAR 5.51.6	None	36
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_PA

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Returns:

Logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

indicating the interrupt status of the selected pins [Default: 0]

17.2.2.6 void GPIO_interruptEdgeSelect (uint8_t *selectedPort*, uint16_t *selectedPins*, uint8_t *edgeSelect*)

This function selects on what edge the port interrupt flag should be set for a transition.

This function selects on what edge the port interrupt flag should be set for a transition. Values for *edgeSelect* should be GPIO_LOW_TO_HIGH_TRANSITION or GPIO_HIGH_TO_LOW_TRANSITION.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	82
TI Compiler 4.2.1	Size	32
TI Compiler 4.2.1	Speed	32
IAR 5.51.6	None	56
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	118
MSPGCC 4.8.0	Size	48
MSPGCC 4.8.0	Speed	40

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14

- GPIO_PIN15

edgeSelect specifies what transition sets the interrupt flag Valid values are:

- GPIO_HIGH_TO_LOW_TRANSITION
- GPIO_LOW_TO_HIGH_TRANSITION

Modified bits of **PxIES** register.

Returns:

None

17.2.2.7 void GPIO_setAsInputPin (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function configures the selected Pin as input pin.

This function selected pins on a selected port as input pins.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	100
TI Compiler 4.2.1	Size	36
TI Compiler 4.2.1	Speed	36
IAR 5.51.6	None	64
IAR 5.51.6	Size	44
IAR 5.51.6	Speed	60
MSPGCC 4.8.0	None	160
MSPGCC 4.8.0	Size	40
MSPGCC 4.8.0	Speed	40

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5

- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Modified bits of **PxDIR** register, bits of **PxREN** register and bits of **PxSEL** register.

Returns:

None

17.2.2.8 void GPIO_setAsInputPinWithPullDownResistor (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function sets the selected Pin in input Mode with Pull Down resistor.

This function sets the selected Pin in input Mode with Pull Down resistor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	112
TI Compiler 4.2.1	Size	42
TI Compiler 4.2.1	Speed	42
IAR 5.51.6	None	72
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	180
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Modified bits of **PxDIR** register, bits of **PxOUT** register and bits of **PxREN** register.

Returns:

None

17.2.2.9 void GPIO_setAsInputPinWithPullUpResistor (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function sets the selected Pin in input Mode with Pull Up resistor.

This function sets the selected Pin in input Mode with Pull Up resistor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	112
TI Compiler 4.2.1	Size	42
TI Compiler 4.2.1	Speed	42
IAR 5.51.6	None	72
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	176
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11

- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Modified bits of **PxDIR** register, bits of **PxOUT** register and bits of **PxREN** register.

Returns:

None

17.2.2.10 void GPIO_setAsOutputPin (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function configures the selected Pin as output pin.

This function selected pins on a selected port as output pins.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	86
TI Compiler 4.2.1	Size	34
TI Compiler 4.2.1	Speed	34
IAR 5.51.6	None	54
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	130
MSPGCC 4.8.0	Size	38
MSPGCC 4.8.0	Speed	38

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5

- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Modified bits of **PxDIR** register and bits of **PxSEL** register.

Returns:

None

17.2.2.11 void GPIO_setAsPeripheralModuleFunctionInputPin (uint8_t *selectedPort*, uint16_t *selectedPins*, uint8_t *mode*)

This function configures the peripheral module function in the input direction for the selected pin for either primary, secondary or ternary module function modes.

This function configures the peripheral module function in the input direction for the selected pin for either primary, secondary or ternary module function modes. Accepted values for mode are GPIO_PRIMARY_MODULE_FUNCTION, GPIO_SECONDARY_MODULE_FUNCTION, and GPIO_TERNARY_MODULE_FUNCTION

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	168
TI Compiler 4.2.1	Size	64
TI Compiler 4.2.1	Speed	68
IAR 5.51.6	None	118
IAR 5.51.6	Size	74
IAR 5.51.6	Speed	102
MSPGCC 4.8.0	None	260
MSPGCC 4.8.0	Size	76
MSPGCC 4.8.0	Speed	78

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

mode is the specified mode that the pin should be configured for the module function. Valid values are:

- GPIO_PRIMARY_MODULE_FUNCTION
- GPIO_SECONDARY_MODULE_FUNCTION
- GPIO_TERNARY_MODULE_FUNCTION

Modified bits of **PxDIR** register and bits of **PxSEL** register.

Returns:

None

17.2.2.12 void GPIO_setAsPeripheralModuleFunctionOutputPin (uint8_t *selectedPort*, uint16_t *selectedPins*, uint8_t *mode*)

This function configures the peripheral module function in the output direction for the selected pin for either primary, secondary or ternary module function modes.

This function configures the peripheral module function in the output direction for the selected pin for either primary, secondary or ternary module function modes. Accepted values for mode are GPIO_PRIMARY_MODULE_FUNCTION, GPIO_SECONDARY_MODULE_FUNCTION, and GPIO_TERNARY_MODULE_FUNCTION

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	168
TI Compiler 4.2.1	Size	60
TI Compiler 4.2.1	Speed	64
IAR 5.51.6	None	118
IAR 5.51.6	Size	74
IAR 5.51.6	Speed	86
MSPGCC 4.8.0	None	256
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	74

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

mode is the specified mode that the pin should be configured for the module function. Valid values are:

- GPIO_PRIMARY_MODULE_FUNCTION
- GPIO_SECONDARY_MODULE_FUNCTION
- GPIO_TERNARY_MODULE_FUNCTION

Modified bits of **PxDIR** register and bits of **PxSEL** register.

Returns:

None

17.2.2.13 void GPIO_setOutputHighOnPin (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function sets output HIGH on the selected Pin.

This function sets output HIGH on the selected port's pin.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	58
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	34
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Modified bits of **PxOUT** register.

Returns:

None

17.2.2.14 void GPIO_setOutputLowOnPin (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function sets output LOW on the selected Pin.

This function sets output LOW on the selected port's pin.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	58
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	34
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4
- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10

- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Modified bits of **PxOUT** register.

Returns:
None

17.2.2.15 void GPIO_toggleOutputOnPin (uint8_t *selectedPort*, uint16_t *selectedPins*)

This function toggles the output on the selected Pin.

This function toggles the output on the selected port's pin.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	58
TI Compiler 4.2.1	Size	22
TI Compiler 4.2.1	Speed	22
IAR 5.51.6	None	34
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	28
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Parameters:

selectedPort is the selected port. Valid values are:

- GPIO_PORT_P1
- GPIO_PORT_P2
- GPIO_PORT_P3
- GPIO_PORT_P4
- GPIO_PORT_P5
- GPIO_PORT_P6
- GPIO_PORT_P7
- GPIO_PORT_P8
- GPIO_PORT_P9
- GPIO_PORT_P10
- GPIO_PORT_P11
- GPIO_PORT_PA
- GPIO_PORT_PB
- GPIO_PORT_PC
- GPIO_PORT_PD
- GPIO_PORT_PE
- GPIO_PORT_PF
- GPIO_PORT_PJ

selectedPins is the specified pin in the selected port. Mask value is the logical OR of any of the following:

- GPIO_PIN0
- GPIO_PIN1
- GPIO_PIN2
- GPIO_PIN3
- GPIO_PIN4

- GPIO_PIN5
- GPIO_PIN6
- GPIO_PIN7
- GPIO_PIN8
- GPIO_PIN9
- GPIO_PIN10
- GPIO_PIN11
- GPIO_PIN12
- GPIO_PIN13
- GPIO_PIN14
- GPIO_PIN15

Modified bits of **PxOUT** register.

Returns:
None

17.3 Programming Example

The following example shows how to use the GPIO API. A trigger is generated on a hi "TO" low transition on P1.4 (pulled-up input pin), which will generate P1_ISR. In the ISR, we toggle P1.0 (output pin).

```
//Set P1.0 to output direction
GPIO_setAsOutputPin(
    GPIO_PORT_P1,
    GPIO_PIN0
);

//Enable P1.4 internal resistance as pull-Up resistance
GPIO_setAsInputPinWithPullUpresistor(
    GPIO_PORT_P1,
    GPIO_PIN4
);

//P1.4 interrupt enabled
GPIO_enableInterrupt(
    GPIO_PORT_P1,
    GPIO_PIN4
);

//P1.4 Hi/Lo edge
GPIO_interruptEdgeSelect(
    GPIO_PORT_P1,
    GPIO_PIN4,
    GPIO_HIGH_TO_LOW_TRANSITION
);

//P1.4 IFG cleared
GPIO_clearInterruptFlag(
    GPIO_PORT_P1,
    GPIO_PIN4
);

//Enter LPM4 w/interrupt
__bis_SR_register(LPM4_bits + GIE);

//For debugger
```

```
    __no_operation();
}

//*****
//
//This is the PORT1_VECTOR interrupt vector service routine
//
//*****
#pragma vector=PORT1_VECTOR
__interrupt void Port_1 (void)
{
    //P1.0 = toggle
    GPIO_toggleOutputOnPin(
        GPIO_PORT_P1,
        GPIO_PIN0
    );

    //P1.4 IFG cleared
    GPIO_clearInterruptFlag(
        GPIO_PORT_P1,
        GPIO_PIN4
    );
}
```

18 LCD_C Controller

Introduction	205
API Functions	205
Programming Example	222

18.1 Introduction

The LCD_C Controller APIs provides a set of functions for using the LCD_C module. Main functions include initialization, LCD enable/disable, charge pump config, voltage settings and memory/blink memory writing.

The difference between LCD_B and LCD_C is that LCD_C supports 5-mux ~ 8-mux and low power waveform.

This driver is contained in `lcd_c.c`, with `lcd_c.h` containing the API definitions for use by applications.

18.2 API Functions

Functions

- void [LCD_C_clearBlinkingMemory](#) (uint16_t baseAddress)
- void [LCD_C_clearInterrupt](#) (uint16_t baseAddress, uint16_t mask)
- void [LCD_C_clearMemory](#) (uint16_t baseAddress)
- void [LCD_C_configChargePump](#) (uint16_t baseAddress, uint16_t syncToClock, uint16_t functionControl)
- void [LCD_C_disableChargePump](#) (uint16_t baseAddress)
- void [LCD_C_disableInterrupt](#) (uint16_t baseAddress, uint16_t mask)
- void [LCD_C_enableChargePump](#) (uint16_t baseAddress)
- void [LCD_C_enableInterrupt](#) (uint16_t baseAddress, uint16_t mask)
- uint16_t [LCD_C_getInterruptStatus](#) (uint16_t baseAddress, uint16_t mask)
- void [LCD_C_init](#) (uint16_t baseAddress, LCD_C_initParam *initParams)
- void [LCD_C_off](#) (uint16_t baseAddress)
- void [LCD_C_on](#) (uint16_t baseAddress)
- void [LCD_C_selectBias](#) (uint16_t baseAddress, uint16_t bias)
- void [LCD_C_selectChargePumpReference](#) (uint16_t baseAddress, uint16_t reference)
- void [LCD_C_selectDisplayMemory](#) (uint16_t baseAddress, uint16_t displayMemory)
- void [LCD_C_setBlinkingControl](#) (uint16_t baseAddress, uint8_t clockDivider, uint8_t clockPrescaler, uint8_t mode)
- void [LCD_C_setBlinkingMemory](#) (uint16_t baseAddress, uint8_t pin, uint8_t value)
- void [LCD_C_setMemory](#) (uint16_t baseAddress, uint8_t pin, uint8_t value)
- void [LCD_C_setMemoryCharacter](#) (uint16_t baseAddress, uint8_t startPin, uint8_t character)
- void [LCD_C_setPinAsLCDFunction](#) (uint16_t baseAddress, uint8_t pin)
- void [LCD_C_setPinAsLCDFunctionEx](#) (uint16_t baseAddress, uint8_t startPin, uint8_t endPin)
- void [LCD_C_setPinAsPortFunction](#) (uint16_t baseAddress, uint8_t pin)
- void [LCD_C_setVLCDSource](#) (uint16_t baseAddress, uint16_t vlcdSource, uint16_t v2v3v4Source, uint16_t v5Source)
- void [LCD_C_setVLCDVoltage](#) (uint16_t baseAddress, uint16_t voltage)

18.2.1 Detailed Description

The LCD_C API is broken into four groups of functions: those that deal with the basic setup and pin config, those that handle charge pump, VLCD voltage and source, those that set memory and blink memory, and those auxiliary functions.

The LCD_C setup and pin config functions are

- [LCD_C_init\(\)](#)

- [LCD_C_on\(\)](#)
- [LCD_C_off\(\)](#)
- [LCD_C_setPinAsLCDFunction\(\)](#)
- [LCD_C_setPinAsPortFunction\(\)](#)
- [LCD_C_setPinAsLCDFunctionEx\(\)](#)

The LCD_C charge pump, VLCD voltage/source functions are

- [LCD_C_enableChargePump](#)
- [LCD_C_disableChargePump\(\)](#)
- [LCD_C_configChargePump\(\)](#)
- [LCD_C_selectBias\(\)](#)
- [LCD_C_selectChargePumpReference\(\)](#)
- [LCD_C_setVLCDSource\(\)](#)
- [LCD_C_setVLCDVoltage\(\)](#)

The LCD_C memory/blinking memory setting funtions are

- [LCD_C_clearMemory\(\)](#)
- [LCD_C_clearBlinkingMemory\(\)](#)
- [LCD_C_selectDisplayMemory\(\)](#)
- [LCD_C_setBlinkingControl\(\)](#)
- [LCD_C_setMemory\(\)](#)
- [LCD_C_setBlinkingMemory\(\)](#)
- [LCD_C_setMemoryCharacter\(\)](#)

The LCD_C auxiliary functions are

- [LCD_C_clearInterrupt\(\)](#)
- [LCD_C_getInterruptStatus\(\)](#)
- [LCD_C_enableInterrupt\(\)](#)
- [LCD_C_disableInterrupt\(\)](#)

18.2.2 Function Documentation

18.2.2.1 void LCD_C_clearBlinkingMemory (uint16_t *baseAddress*)

Clears all LCD blinking memory registers.

Parameters:

baseAddress is the base address of the LCD_C module.

Modified bits are **LCDCLRBM** of **LCDMEMCTL** register.

Returns:

None

18.2.2.2 void LCD_C_clearInterrupt (uint16_t *baseAddress*, uint16_t *mask*)

Clears the LCD interrupt flags.

Parameters:

baseAddress is the base address of the LCD_C module.

mask is the masked interrupt flag to be cleared. Valid values are:

- LCD_C_NO_CAPACITANCE_CONNECTED_INTERRUPT
- LCD_C_BLINKING_SEGMENTS_ON_INTERRUPT
- LCD_C_BLINKING_SEGMENTS_OFF_INTERRUPT
- LCD_C_FRAME_INTERRUPT

Modified bits are LCDCAPIFG, LCDBLKONIFG, LCDBLKOFFIFG and LCDFRMIFG of LCDCTL1 register.

Returns:

None

18.2.2.3 void LCD_C_clearMemory (uint16_t *baseAddress*)

Clears all LCD memory registers.

Parameters:

baseAddress is the base address of the LCD_C module.

Modified bits are LCDCLRM of LCDMEMCTL register.

Returns:

None

18.2.2.4 void LCD_C_configChargePump (uint16_t *baseAddress*, uint16_t *syncToClock*, uint16_t *functionControl*)

Configures the charge pump for synchronization and disabled capability.

This function is device-specific. The charge pump clock can be synchronized to a device-specific clock, and also can be disabled by connected function.

Parameters:

baseAddress is the base address of the LCD_C module.

syncToClock is the synchronization select. Valid values are:

- LCD_C_SYNCHRONIZATION_DISABLED [Default]
- LCD_C_SYNCHRONIZATION_ENABLED

functionControl is the connected function control select. Setting 0 to make connected function not disable charge pump.

Modified bits are MBITx of LCDBMx register.

Returns:

None

18.2.2.5 void LCD_C_disableChargePump (uint16_t *baseAddress*)

Disables the charge pump.

Parameters:

baseAddress is the base address of the LCD_C module.

Modified bits are **LCDCPEN** of **LCDVCTL** register; bits **LCDON** of **LCDCTL0** register.

Returns:
None

18.2.2.6 void LCD_C_disableInterrupt (uint16_t *baseAddress*, uint16_t *mask*)

Disables LCD interrupt sources.

Parameters:

baseAddress is the base address of the LCD_C module.

mask is the interrupts to be disabled. Valid values are:

- **LCD_C_NO_CAPACITANCE_CONNECTED_INTERRUPT**
- **LCD_C_BLINKING_SEGMENTS_ON_INTERRUPT**
- **LCD_C_BLINKING_SEGMENTS_OFF_INTERRUPT**
- **LCD_C_FRAME_INTERRUPT**

Modified bits are **LCDCAPIE**, **LCDBLKONIE**, **LCDBLKOFFIE** and **LCDFRMIE** of **LCDCTL1** register.

Returns:
None

18.2.2.7 void LCD_C_enableChargePump (uint16_t *baseAddress*)

Enables the charge pump.

Parameters:

baseAddress is the base address of the LCD_C module.

Modified bits are **LCDCPEN** of **LCDVCTL** register; bits **LCDON** of **LCDCTL0** register.

Returns:
None

18.2.2.8 void LCD_C_enableInterrupt (uint16_t *baseAddress*, uint16_t *mask*)

Enables LCD interrupt sources.

Parameters:

baseAddress is the base address of the LCD_C module.

mask is the interrupts to be enabled. Valid values are:

- **LCD_C_NO_CAPACITANCE_CONNECTED_INTERRUPT**
- **LCD_C_BLINKING_SEGMENTS_ON_INTERRUPT**
- **LCD_C_BLINKING_SEGMENTS_OFF_INTERRUPT**
- **LCD_C_FRAME_INTERRUPT**

Modified bits are **LCDCAPIE**, **LCDBLKONIE**, **LCDBLKOFFIE** and **LCDFRMIE** of **LCDCTL1** register.

Returns:
None

18.2.2.9 uint16_t LCD_C_getInterruptStatus (uint16_t *baseAddress*, uint16_t *mask*)

Gets the LCD interrupt status.

Parameters:

baseAddress is the base address of the LCD_C module.

mask is the masked interrupt flags. Valid values are:

- LCD_C_NO_CAPACITANCE_CONNECTED_INTERRUPT
- LCD_C_BLINKING_SEGMENTS_ON_INTERRUPT
- LCD_C_BLINKING_SEGMENTS_OFF_INTERRUPT
- LCD_C_FRAME_INTERRUPT

Returns:

None Return Logical OR of any of the following:

- LCD_C_NO_CAPACITANCE_CONNECTED_INTERRUPT
- LCD_C_BLINKING_SEGMENTS_ON_INTERRUPT
- LCD_C_BLINKING_SEGMENTS_OFF_INTERRUPT
- LCD_C_FRAME_INTERRUPT

indicating the status of the masked interrupts

18.2.2.10 void LCD_C_init (uint16_t *baseAddress*, LCD_C_initParam * *initParams*)

Initializes the LCD Module.

his function initializes the LCD but without turning on. It bascially setup the clock source, clock divider, clock prescalar, mux rate, low-power waveform and segments on/off. After calling this function, user can config charge pump, internal reference voltage and voltage sources.

Parameters:

baseAddress is the base address of the LCD_C module.

initParams is the pointer to LCD_InitParam structure. See the following parameters for each field.

Returns:

None

18.2.2.11 void LCD_C_off (uint16_t *baseAddress*)

Turns off the LCD module.

Parameters:

baseAddress is the base address of the LCD_C module.

Modified bits are **LCDON** of **LCDCTL0** register.

Returns:

None

18.2.2.12 void LCD_C_on (uint16_t *baseAddress*)

Turns on the LCD module.

Parameters:

baseAddress is the base address of the LCD_C module.

Modified bits are **LCDON** of **LCDCTL0** register.

Returns:

None

18.2.2.13 void LCD_C_selectBias (uint16_t *baseAddress*, uint16_t *bias*)

Selects the bias level.

Parameters:

baseAddress is the base address of the LCD_C module.

bias is the select for bias level. Valid values are:

- **LCD_C_BIAS_1_3** [Default] - 1/3 bias
- **LCD_C_BIAS_1_2** - 1/2 bias

Modified bits are **LCD2B** of **LCDVCTL** register; bits **LCDON** of **LCDCTL0** register.

Returns:

None

18.2.2.14 void LCD_C_selectChargePumpReference (uint16_t *baseAddress*, uint16_t *reference*)

Selects the charge pump reference.

The charge pump reference does not support LCD_C_EXTERNAL_REFERENCE_VOLTAGE, LCD_C_INTERNAL_REFERENCE_VOLTAGE_SWITCHED_TO_EXTERNAL_PIN when LCD_C_V2V3V4_SOURCED_EXTERNALLY or LCD_C_V2V3V4_GENERATED INTERNALLY_SWITCHED_TO_PINS is selected.

Parameters:

baseAddress is the base address of the LCD_C module.

reference is the select for charge pump reference. Valid values are:

- **LCD_C_INTERNAL_REFERENCE_VOLTAGE** [Default]
- **LCD_C_EXTERNAL_REFERENCE_VOLTAGE**
- **LCD_C_INTERNAL_REFERENCE_VOLTAGE_SWITCHED_TO_EXTERNAL_PIN**

Modified bits are **VLCDREFx** of **LCDVCTL** register; bits **LCDON** of **LCDCTL0** register.

Returns:

None

18.2.2.15 void LCD_C_selectDisplayMemory (uint16_t *baseAddress*, uint16_t *displayMemory*)

Selects display memory.

This function selects display memory either from memory or blinking memory. Please note if the blinking mode is selected as LCD_BLINKMODE_INDIVIDUALSEGMENTS or LCD_BLINKMODE_ALLSEGMENTS or mux rate ≥ 5 , display memory can not be changed. If LCD_BLINKMODE_SWITCHDISPLAYCONTENTS is selected, display memory bit reflects current displayed memory.

Parameters:

baseAddress is the base address of the LCD_C module.

displayMemory is the desired displayed memory. Valid values are:

- **LCD_C_DISPLAYSOURCE_MEMORY** [Default]
 - **LCD_C_DISPLAYSOURCE_BLINKINGMEMORY**
- Modified bits are **LCDDISP** of **LCDMEMCTL** register.

Returns:

None

18.2.2.16 void LCD_C_setBlinkingControl (uint16_t *baseAddress*, uint8_t *clockDivider*, uint8_t *clockPrescalar*, uint8_t *mode*)

Sets the blink settings.

Parameters:

baseAddress is the base address of the LCD_C module.

clockDivider is the clock divider for blinking frequency. Valid values are:

- LCD_C_BLINK_FREQ_CLOCK_DIVIDER_1 [Default]
- LCD_C_BLINK_FREQ_CLOCK_DIVIDER_2
- LCD_C_BLINK_FREQ_CLOCK_DIVIDER_3
- LCD_C_BLINK_FREQ_CLOCK_DIVIDER_4
- LCD_C_BLINK_FREQ_CLOCK_DIVIDER_5
- LCD_C_BLINK_FREQ_CLOCK_DIVIDER_6
- LCD_C_BLINK_FREQ_CLOCK_DIVIDER_7
- LCD_C_BLINK_FREQ_CLOCK_DIVIDER_8

Modified bits are LCDCLKDIVx of LCDCLKCTL register.

clockPrescalar is the clock pre-scalar for blinking frequency. Valid values are:

- LCD_C_BLINK_FREQ_CLOCK_PRESCALAR_512 [Default]
- LCD_C_BLINK_FREQ_CLOCK_PRESCALAR_1024
- LCD_C_BLINK_FREQ_CLOCK_PRESCALAR_2048
- LCD_C_BLINK_FREQ_CLOCK_PRESCALAR_4096
- LCD_C_BLINK_FREQ_CLOCK_PRESCALAR_8162
- LCD_C_BLINK_FREQ_CLOCK_PRESCALAR_16384
- LCD_C_BLINK_FREQ_CLOCK_PRESCALAR_32768
- LCD_C_BLINK_FREQ_CLOCK_PRESCALAR_65536

Modified bits are LCDCLKPREx of LCDCLKCTL register.

Returns:

None

18.2.2.17 void LCD_C_setBlinkingMemory (uint16_t *baseAddress*, uint8_t *pin*, uint8_t *value*)

Sets the LCD blink memory register.

Parameters:

baseAddress is the base address of the LCD_C module.

pin is the select pin for setting value. Valid values are:

- LCD_C_SEGMENT_LINE_0
- LCD_C_SEGMENT_LINE_1
- LCD_C_SEGMENT_LINE_2
- LCD_C_SEGMENT_LINE_3
- LCD_C_SEGMENT_LINE_4
- LCD_C_SEGMENT_LINE_5
- LCD_C_SEGMENT_LINE_6
- LCD_C_SEGMENT_LINE_7
- LCD_C_SEGMENT_LINE_8
- LCD_C_SEGMENT_LINE_9
- LCD_C_SEGMENT_LINE_10
- LCD_C_SEGMENT_LINE_11
- LCD_C_SEGMENT_LINE_12
- LCD_C_SEGMENT_LINE_13
- LCD_C_SEGMENT_LINE_14
- LCD_C_SEGMENT_LINE_15

- LCD_C_SEGMENT_LINE_16
- LCD_C_SEGMENT_LINE_17
- LCD_C_SEGMENT_LINE_18
- LCD_C_SEGMENT_LINE_19
- LCD_C_SEGMENT_LINE_20
- LCD_C_SEGMENT_LINE_21
- LCD_C_SEGMENT_LINE_22
- LCD_C_SEGMENT_LINE_23
- LCD_C_SEGMENT_LINE_24
- LCD_C_SEGMENT_LINE_25
- LCD_C_SEGMENT_LINE_26
- LCD_C_SEGMENT_LINE_27
- LCD_C_SEGMENT_LINE_28
- LCD_C_SEGMENT_LINE_29
- LCD_C_SEGMENT_LINE_30
- LCD_C_SEGMENT_LINE_31
- LCD_C_SEGMENT_LINE_32
- LCD_C_SEGMENT_LINE_33
- LCD_C_SEGMENT_LINE_34
- LCD_C_SEGMENT_LINE_35
- LCD_C_SEGMENT_LINE_36
- LCD_C_SEGMENT_LINE_37
- LCD_C_SEGMENT_LINE_38
- LCD_C_SEGMENT_LINE_39
- LCD_C_SEGMENT_LINE_40
- LCD_C_SEGMENT_LINE_41
- LCD_C_SEGMENT_LINE_42
- LCD_C_SEGMENT_LINE_43
- LCD_C_SEGMENT_LINE_44
- LCD_C_SEGMENT_LINE_45
- LCD_C_SEGMENT_LINE_46
- LCD_C_SEGMENT_LINE_47
- LCD_C_SEGMENT_LINE_48
- LCD_C_SEGMENT_LINE_49
- LCD_C_SEGMENT_LINE_50
- LCD_C_SEGMENT_LINE_51
- LCD_C_SEGMENT_LINE_52
- LCD_C_SEGMENT_LINE_53
- LCD_C_SEGMENT_LINE_54
- LCD_C_SEGMENT_LINE_55
- LCD_C_SEGMENT_LINE_56
- LCD_C_SEGMENT_LINE_57
- LCD_C_SEGMENT_LINE_58
- LCD_C_SEGMENT_LINE_59
- LCD_C_SEGMENT_LINE_60
- LCD_C_SEGMENT_LINE_61
- LCD_C_SEGMENT_LINE_62
- LCD_C_SEGMENT_LINE_63

value is the designated value for corresponding blink pin.

Modified bits are **MBITx** of **LCDBMx** register.

Returns:

None

18.2.2.18 void LCD_C_setMemory (uint16_t *baseAddress*, uint8_t *pin*, uint8_t *value*)

Sets the LCD memory register.

Parameters:

baseAddress is the base address of the LCD_C module.

pin is the select pin for setting value. Valid values are:

- LCD_C_SEGMENT_LINE_0
- LCD_C_SEGMENT_LINE_1
- LCD_C_SEGMENT_LINE_2
- LCD_C_SEGMENT_LINE_3
- LCD_C_SEGMENT_LINE_4
- LCD_C_SEGMENT_LINE_5
- LCD_C_SEGMENT_LINE_6
- LCD_C_SEGMENT_LINE_7
- LCD_C_SEGMENT_LINE_8
- LCD_C_SEGMENT_LINE_9
- LCD_C_SEGMENT_LINE_10
- LCD_C_SEGMENT_LINE_11
- LCD_C_SEGMENT_LINE_12
- LCD_C_SEGMENT_LINE_13
- LCD_C_SEGMENT_LINE_14
- LCD_C_SEGMENT_LINE_15
- LCD_C_SEGMENT_LINE_16
- LCD_C_SEGMENT_LINE_17
- LCD_C_SEGMENT_LINE_18
- LCD_C_SEGMENT_LINE_19
- LCD_C_SEGMENT_LINE_20
- LCD_C_SEGMENT_LINE_21
- LCD_C_SEGMENT_LINE_22
- LCD_C_SEGMENT_LINE_23
- LCD_C_SEGMENT_LINE_24
- LCD_C_SEGMENT_LINE_25
- LCD_C_SEGMENT_LINE_26
- LCD_C_SEGMENT_LINE_27
- LCD_C_SEGMENT_LINE_28
- LCD_C_SEGMENT_LINE_29
- LCD_C_SEGMENT_LINE_30
- LCD_C_SEGMENT_LINE_31
- LCD_C_SEGMENT_LINE_32
- LCD_C_SEGMENT_LINE_33
- LCD_C_SEGMENT_LINE_34
- LCD_C_SEGMENT_LINE_35
- LCD_C_SEGMENT_LINE_36
- LCD_C_SEGMENT_LINE_37
- LCD_C_SEGMENT_LINE_38
- LCD_C_SEGMENT_LINE_39
- LCD_C_SEGMENT_LINE_40
- LCD_C_SEGMENT_LINE_41
- LCD_C_SEGMENT_LINE_42
- LCD_C_SEGMENT_LINE_43
- LCD_C_SEGMENT_LINE_44
- LCD_C_SEGMENT_LINE_45
- LCD_C_SEGMENT_LINE_46
- LCD_C_SEGMENT_LINE_47
- LCD_C_SEGMENT_LINE_48

- LCD_C_SEGMENT_LINE_49
- LCD_C_SEGMENT_LINE_50
- LCD_C_SEGMENT_LINE_51
- LCD_C_SEGMENT_LINE_52
- LCD_C_SEGMENT_LINE_53
- LCD_C_SEGMENT_LINE_54
- LCD_C_SEGMENT_LINE_55
- LCD_C_SEGMENT_LINE_56
- LCD_C_SEGMENT_LINE_57
- LCD_C_SEGMENT_LINE_58
- LCD_C_SEGMENT_LINE_59
- LCD_C_SEGMENT_LINE_60
- LCD_C_SEGMENT_LINE_61
- LCD_C_SEGMENT_LINE_62
- LCD_C_SEGMENT_LINE_63

value is the designated value for corresponding pin.

Modified bits are **MBITx** of **LCDMx** register.

Returns:

None

18.2.2.19 void LCD_C_setMemoryCharacter (uint16_t *baseAddress*, uint8_t *startPin*, uint8_t *character*)

Sets the LCD memory with the specified character.

This function sets the LCD_C memory with user specified character. It only support 7-segment LCD layout and '0'~'9' characters.

Parameters:

baseAddress is the base address of the LCD_C module.

startPin is the beginning index of LCD pin for character display. Valid values are:

- LCD_C_SEGMENT_LINE_0
- LCD_C_SEGMENT_LINE_1
- LCD_C_SEGMENT_LINE_2
- LCD_C_SEGMENT_LINE_3
- LCD_C_SEGMENT_LINE_4
- LCD_C_SEGMENT_LINE_5
- LCD_C_SEGMENT_LINE_6
- LCD_C_SEGMENT_LINE_7
- LCD_C_SEGMENT_LINE_8
- LCD_C_SEGMENT_LINE_9
- LCD_C_SEGMENT_LINE_10
- LCD_C_SEGMENT_LINE_11
- LCD_C_SEGMENT_LINE_12
- LCD_C_SEGMENT_LINE_13
- LCD_C_SEGMENT_LINE_14
- LCD_C_SEGMENT_LINE_15
- LCD_C_SEGMENT_LINE_16
- LCD_C_SEGMENT_LINE_17
- LCD_C_SEGMENT_LINE_18
- LCD_C_SEGMENT_LINE_19
- LCD_C_SEGMENT_LINE_20
- LCD_C_SEGMENT_LINE_21
- LCD_C_SEGMENT_LINE_22

- LCD_C_SEGMENT_LINE_23
- LCD_C_SEGMENT_LINE_24
- LCD_C_SEGMENT_LINE_25
- LCD_C_SEGMENT_LINE_26
- LCD_C_SEGMENT_LINE_27
- LCD_C_SEGMENT_LINE_28
- LCD_C_SEGMENT_LINE_29
- LCD_C_SEGMENT_LINE_30
- LCD_C_SEGMENT_LINE_31
- LCD_C_SEGMENT_LINE_32
- LCD_C_SEGMENT_LINE_33
- LCD_C_SEGMENT_LINE_34
- LCD_C_SEGMENT_LINE_35
- LCD_C_SEGMENT_LINE_36
- LCD_C_SEGMENT_LINE_37
- LCD_C_SEGMENT_LINE_38
- LCD_C_SEGMENT_LINE_39
- LCD_C_SEGMENT_LINE_40
- LCD_C_SEGMENT_LINE_41
- LCD_C_SEGMENT_LINE_42
- LCD_C_SEGMENT_LINE_43
- LCD_C_SEGMENT_LINE_44
- LCD_C_SEGMENT_LINE_45
- LCD_C_SEGMENT_LINE_46
- LCD_C_SEGMENT_LINE_47
- LCD_C_SEGMENT_LINE_48
- LCD_C_SEGMENT_LINE_49
- LCD_C_SEGMENT_LINE_50
- LCD_C_SEGMENT_LINE_51
- LCD_C_SEGMENT_LINE_52
- LCD_C_SEGMENT_LINE_53
- LCD_C_SEGMENT_LINE_54
- LCD_C_SEGMENT_LINE_55
- LCD_C_SEGMENT_LINE_56
- LCD_C_SEGMENT_LINE_57
- LCD_C_SEGMENT_LINE_58
- LCD_C_SEGMENT_LINE_59
- LCD_C_SEGMENT_LINE_60
- LCD_C_SEGMENT_LINE_61
- LCD_C_SEGMENT_LINE_62
- LCD_C_SEGMENT_LINE_63

character is the character to display.

Modified bits are **LCDCLKSYNC** and **LCDPDISx** of **LDCCPCTL** register.

Returns:

None

18.2.2.20 void LCD_C_setPinAsLCDFunction (uint16_t *baseAddress*, uint8_t *pin*)

Sets the LCD Pin as LCD functions.

Parameters:

baseAddress is the base address of the LCD_C module.

pin is the select pin set as LCD function. Valid values are:

- LCD_C_SEGMENT_LINE_0
- LCD_C_SEGMENT_LINE_1
- LCD_C_SEGMENT_LINE_2
- LCD_C_SEGMENT_LINE_3
- LCD_C_SEGMENT_LINE_4
- LCD_C_SEGMENT_LINE_5
- LCD_C_SEGMENT_LINE_6
- LCD_C_SEGMENT_LINE_7
- LCD_C_SEGMENT_LINE_8
- LCD_C_SEGMENT_LINE_9
- LCD_C_SEGMENT_LINE_10
- LCD_C_SEGMENT_LINE_11
- LCD_C_SEGMENT_LINE_12
- LCD_C_SEGMENT_LINE_13
- LCD_C_SEGMENT_LINE_14
- LCD_C_SEGMENT_LINE_15
- LCD_C_SEGMENT_LINE_16
- LCD_C_SEGMENT_LINE_17
- LCD_C_SEGMENT_LINE_18
- LCD_C_SEGMENT_LINE_19
- LCD_C_SEGMENT_LINE_20
- LCD_C_SEGMENT_LINE_21
- LCD_C_SEGMENT_LINE_22
- LCD_C_SEGMENT_LINE_23
- LCD_C_SEGMENT_LINE_24
- LCD_C_SEGMENT_LINE_25
- LCD_C_SEGMENT_LINE_26
- LCD_C_SEGMENT_LINE_27
- LCD_C_SEGMENT_LINE_28
- LCD_C_SEGMENT_LINE_29
- LCD_C_SEGMENT_LINE_30
- LCD_C_SEGMENT_LINE_31
- LCD_C_SEGMENT_LINE_32
- LCD_C_SEGMENT_LINE_33
- LCD_C_SEGMENT_LINE_34
- LCD_C_SEGMENT_LINE_35
- LCD_C_SEGMENT_LINE_36
- LCD_C_SEGMENT_LINE_37
- LCD_C_SEGMENT_LINE_38
- LCD_C_SEGMENT_LINE_39
- LCD_C_SEGMENT_LINE_40
- LCD_C_SEGMENT_LINE_41
- LCD_C_SEGMENT_LINE_42
- LCD_C_SEGMENT_LINE_43
- LCD_C_SEGMENT_LINE_44
- LCD_C_SEGMENT_LINE_45
- LCD_C_SEGMENT_LINE_46
- LCD_C_SEGMENT_LINE_47
- LCD_C_SEGMENT_LINE_48
- LCD_C_SEGMENT_LINE_49
- LCD_C_SEGMENT_LINE_50
- LCD_C_SEGMENT_LINE_51
- LCD_C_SEGMENT_LINE_52
- LCD_C_SEGMENT_LINE_53
- LCD_C_SEGMENT_LINE_54

- LCD_C_SEGMENT_LINE_55
- LCD_C_SEGMENT_LINE_56
- LCD_C_SEGMENT_LINE_57
- LCD_C_SEGMENT_LINE_58
- LCD_C_SEGMENT_LINE_59
- LCD_C_SEGMENT_LINE_60
- LCD_C_SEGMENT_LINE_61
- LCD_C_SEGMENT_LINE_62
- LCD_C_SEGMENT_LINE_63

Modified bits are **LCDSx** of **LCDPCTLx** register; bits **LCDON** of **LCDCTL0** register.

Returns:

None

18.2.2.21 void LCD_C_setPinAsLCDFunctionEx (uint16_t *baseAddress*, uint8_t *startPin*, uint8_t *endPin*)

Sets the LCD pins as LCD function pin.

This function sets the LCD pins as LCD function pin. Instead of passing the all the possible pins, it just requires the start pin and the end pin.

Parameters:

baseAddress is the base address of the LCD_C module.

startPin is the starting pin to be configed as LCD function pin. Valid values are:

- LCD_C_SEGMENT_LINE_0
- LCD_C_SEGMENT_LINE_1
- LCD_C_SEGMENT_LINE_2
- LCD_C_SEGMENT_LINE_3
- LCD_C_SEGMENT_LINE_4
- LCD_C_SEGMENT_LINE_5
- LCD_C_SEGMENT_LINE_6
- LCD_C_SEGMENT_LINE_7
- LCD_C_SEGMENT_LINE_8
- LCD_C_SEGMENT_LINE_9
- LCD_C_SEGMENT_LINE_10
- LCD_C_SEGMENT_LINE_11
- LCD_C_SEGMENT_LINE_12
- LCD_C_SEGMENT_LINE_13
- LCD_C_SEGMENT_LINE_14
- LCD_C_SEGMENT_LINE_15
- LCD_C_SEGMENT_LINE_16
- LCD_C_SEGMENT_LINE_17
- LCD_C_SEGMENT_LINE_18
- LCD_C_SEGMENT_LINE_19
- LCD_C_SEGMENT_LINE_20
- LCD_C_SEGMENT_LINE_21
- LCD_C_SEGMENT_LINE_22
- LCD_C_SEGMENT_LINE_23
- LCD_C_SEGMENT_LINE_24
- LCD_C_SEGMENT_LINE_25
- LCD_C_SEGMENT_LINE_26
- LCD_C_SEGMENT_LINE_27
- LCD_C_SEGMENT_LINE_28
- LCD_C_SEGMENT_LINE_29

- LCD_C_SEGMENT_LINE_30
- LCD_C_SEGMENT_LINE_31
- LCD_C_SEGMENT_LINE_32
- LCD_C_SEGMENT_LINE_33
- LCD_C_SEGMENT_LINE_34
- LCD_C_SEGMENT_LINE_35
- LCD_C_SEGMENT_LINE_36
- LCD_C_SEGMENT_LINE_37
- LCD_C_SEGMENT_LINE_38
- LCD_C_SEGMENT_LINE_39
- LCD_C_SEGMENT_LINE_40
- LCD_C_SEGMENT_LINE_41
- LCD_C_SEGMENT_LINE_42
- LCD_C_SEGMENT_LINE_43
- LCD_C_SEGMENT_LINE_44
- LCD_C_SEGMENT_LINE_45
- LCD_C_SEGMENT_LINE_46
- LCD_C_SEGMENT_LINE_47
- LCD_C_SEGMENT_LINE_48
- LCD_C_SEGMENT_LINE_49
- LCD_C_SEGMENT_LINE_50
- LCD_C_SEGMENT_LINE_51
- LCD_C_SEGMENT_LINE_52
- LCD_C_SEGMENT_LINE_53
- LCD_C_SEGMENT_LINE_54
- LCD_C_SEGMENT_LINE_55
- LCD_C_SEGMENT_LINE_56
- LCD_C_SEGMENT_LINE_57
- LCD_C_SEGMENT_LINE_58
- LCD_C_SEGMENT_LINE_59
- LCD_C_SEGMENT_LINE_60
- LCD_C_SEGMENT_LINE_61
- LCD_C_SEGMENT_LINE_62
- LCD_C_SEGMENT_LINE_63

endPin is the ending pin to be configed as LCD function pin. Valid values are:

- LCD_C_SEGMENT_LINE_0
- LCD_C_SEGMENT_LINE_1
- LCD_C_SEGMENT_LINE_2
- LCD_C_SEGMENT_LINE_3
- LCD_C_SEGMENT_LINE_4
- LCD_C_SEGMENT_LINE_5
- LCD_C_SEGMENT_LINE_6
- LCD_C_SEGMENT_LINE_7
- LCD_C_SEGMENT_LINE_8
- LCD_C_SEGMENT_LINE_9
- LCD_C_SEGMENT_LINE_10
- LCD_C_SEGMENT_LINE_11
- LCD_C_SEGMENT_LINE_12
- LCD_C_SEGMENT_LINE_13
- LCD_C_SEGMENT_LINE_14
- LCD_C_SEGMENT_LINE_15
- LCD_C_SEGMENT_LINE_16
- LCD_C_SEGMENT_LINE_17
- LCD_C_SEGMENT_LINE_18
- LCD_C_SEGMENT_LINE_19

- LCD_C_SEGMENT_LINE_20
- LCD_C_SEGMENT_LINE_21
- LCD_C_SEGMENT_LINE_22
- LCD_C_SEGMENT_LINE_23
- LCD_C_SEGMENT_LINE_24
- LCD_C_SEGMENT_LINE_25
- LCD_C_SEGMENT_LINE_26
- LCD_C_SEGMENT_LINE_27
- LCD_C_SEGMENT_LINE_28
- LCD_C_SEGMENT_LINE_29
- LCD_C_SEGMENT_LINE_30
- LCD_C_SEGMENT_LINE_31
- LCD_C_SEGMENT_LINE_32
- LCD_C_SEGMENT_LINE_33
- LCD_C_SEGMENT_LINE_34
- LCD_C_SEGMENT_LINE_35
- LCD_C_SEGMENT_LINE_36
- LCD_C_SEGMENT_LINE_37
- LCD_C_SEGMENT_LINE_38
- LCD_C_SEGMENT_LINE_39
- LCD_C_SEGMENT_LINE_40
- LCD_C_SEGMENT_LINE_41
- LCD_C_SEGMENT_LINE_42
- LCD_C_SEGMENT_LINE_43
- LCD_C_SEGMENT_LINE_44
- LCD_C_SEGMENT_LINE_45
- LCD_C_SEGMENT_LINE_46
- LCD_C_SEGMENT_LINE_47
- LCD_C_SEGMENT_LINE_48
- LCD_C_SEGMENT_LINE_49
- LCD_C_SEGMENT_LINE_50
- LCD_C_SEGMENT_LINE_51
- LCD_C_SEGMENT_LINE_52
- LCD_C_SEGMENT_LINE_53
- LCD_C_SEGMENT_LINE_54
- LCD_C_SEGMENT_LINE_55
- LCD_C_SEGMENT_LINE_56
- LCD_C_SEGMENT_LINE_57
- LCD_C_SEGMENT_LINE_58
- LCD_C_SEGMENT_LINE_59
- LCD_C_SEGMENT_LINE_60
- LCD_C_SEGMENT_LINE_61
- LCD_C_SEGMENT_LINE_62
- LCD_C_SEGMENT_LINE_63

Modified bits are **LCDSx** of **LCDPCTLx** register; bits **LCDON** of **LCDCTL0** register.

Returns:

None

18.2.2.22 void LCD_C_setPinAsPortFunction (uint16_t *baseAddress*, uint8_t *pin*)

Sets the LCD Pin as Port functions.

Parameters:

baseAddress is the base address of the LCD_C module.

pin is the select pin set as Port function. Valid values are:

- LCD_C_SEGMENT_LINE_0
- LCD_C_SEGMENT_LINE_1
- LCD_C_SEGMENT_LINE_2
- LCD_C_SEGMENT_LINE_3
- LCD_C_SEGMENT_LINE_4
- LCD_C_SEGMENT_LINE_5
- LCD_C_SEGMENT_LINE_6
- LCD_C_SEGMENT_LINE_7
- LCD_C_SEGMENT_LINE_8
- LCD_C_SEGMENT_LINE_9
- LCD_C_SEGMENT_LINE_10
- LCD_C_SEGMENT_LINE_11
- LCD_C_SEGMENT_LINE_12
- LCD_C_SEGMENT_LINE_13
- LCD_C_SEGMENT_LINE_14
- LCD_C_SEGMENT_LINE_15
- LCD_C_SEGMENT_LINE_16
- LCD_C_SEGMENT_LINE_17
- LCD_C_SEGMENT_LINE_18
- LCD_C_SEGMENT_LINE_19
- LCD_C_SEGMENT_LINE_20
- LCD_C_SEGMENT_LINE_21
- LCD_C_SEGMENT_LINE_22
- LCD_C_SEGMENT_LINE_23
- LCD_C_SEGMENT_LINE_24
- LCD_C_SEGMENT_LINE_25
- LCD_C_SEGMENT_LINE_26
- LCD_C_SEGMENT_LINE_27
- LCD_C_SEGMENT_LINE_28
- LCD_C_SEGMENT_LINE_29
- LCD_C_SEGMENT_LINE_30
- LCD_C_SEGMENT_LINE_31
- LCD_C_SEGMENT_LINE_32
- LCD_C_SEGMENT_LINE_33
- LCD_C_SEGMENT_LINE_34
- LCD_C_SEGMENT_LINE_35
- LCD_C_SEGMENT_LINE_36
- LCD_C_SEGMENT_LINE_37
- LCD_C_SEGMENT_LINE_38
- LCD_C_SEGMENT_LINE_39
- LCD_C_SEGMENT_LINE_40
- LCD_C_SEGMENT_LINE_41
- LCD_C_SEGMENT_LINE_42
- LCD_C_SEGMENT_LINE_43
- LCD_C_SEGMENT_LINE_44
- LCD_C_SEGMENT_LINE_45
- LCD_C_SEGMENT_LINE_46
- LCD_C_SEGMENT_LINE_47
- LCD_C_SEGMENT_LINE_48

- LCD_C_SEGMENT_LINE_49
- LCD_C_SEGMENT_LINE_50
- LCD_C_SEGMENT_LINE_51
- LCD_C_SEGMENT_LINE_52
- LCD_C_SEGMENT_LINE_53
- LCD_C_SEGMENT_LINE_54
- LCD_C_SEGMENT_LINE_55
- LCD_C_SEGMENT_LINE_56
- LCD_C_SEGMENT_LINE_57
- LCD_C_SEGMENT_LINE_58
- LCD_C_SEGMENT_LINE_59
- LCD_C_SEGMENT_LINE_60
- LCD_C_SEGMENT_LINE_61
- LCD_C_SEGMENT_LINE_62
- LCD_C_SEGMENT_LINE_63

Modified bits are **LCDSx** of **LCDPCTLx** register; bits **LCDON** of **LCDCTL0** register.

Returns:

None

18.2.2.23 void LCD_C_setVLCDSource (uint16_t *baseAddress*, uint16_t *vlcdSource*, uint16_t *v2v3v4Source*, uint16_t *v5Source*)

Sets the voltage source for V2/V3/V4 and V5.

The charge pump reference does not support LCD_C_EXTERNAL_REFERENCE_VOLTAGE, LCD_C_INTERNAL_REFERENCE_VOLTAGE_SWITCHED_TO_EXTERNAL_PIN when LCD_C_V2V3V4_SOURCED_EXTERNALLY or LCD_C_V2V3V4_GENERATED INTERNALLY_SWITCHED_TO_PINS is selected.

Parameters:

baseAddress is the base address of the LCD_C module.

vlcdSource is the V(LCD) source select. Valid values are:

- LCD_C_VLCD_GENERATED INTERNALLY [Default]
- LCD_C_VLCD_SOURCED_EXTERNALLY

v2v3v4Source is the V2/V3/V4 source select. Valid values are:

- LCD_C_V2V3V4_GENERATED INTERNALLY_NOT_SWITCHED_TO_PINS [Default]
- LCD_C_V2V3V4_GENERATED INTERNALLY_SWITCHED_TO_PINS
- LCD_C_V2V3V4_SOURCED_EXTERNALLY

v5Source is the V5 source select. Valid values are:

- LCD_C_V5_VSS [Default]
- LCD_C_V5_SOURCED_FROM_R03

Modified bits are **VLCDEXT**, **LCDREXT**, **LCDEXTBIAS** and **R03EXT** of **LCDVCTL** register; bits **LCDON** of **LCDCTL0** register.

Returns:

None

18.2.2.24 void LCD_C_setVLCDVoltage (uint16_t *baseAddress*, uint16_t *voltage*)

Selects the charge pump reference.

Sets LCD charge pump voltage.

Parameters:

baseAddress is the base address of the LCD_C module.

voltage is the charge pump select. Valid values are:

- LCD_C_CHARGE_PUMP_DISABLED [Default]
- LCD_C_CHARGE_PUMP_VOLTAGE_2_60V_OR_2_17VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_2_66V_OR_2_22VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_2_72V_OR_2_27VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_2_78V_OR_2_32VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_2_84V_OR_2_37VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_2_90V_OR_2_42VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_2_96V_OR_2_47VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_3_02V_OR_2_52VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_3_08V_OR_2_57VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_3_14V_OR_2_62VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_3_20V_OR_2_67VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_3_26V_OR_2_72VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_3_32V_OR_2_77VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_3_38V_OR_2_82VREF
- LCD_C_CHARGE_PUMP_VOLTAGE_3_44V_OR_2_87VREF

Modified bits are VLCDx of LCDVCTL register; bits LCDON of LCDCTL0 register.

Returns:

None

18.3 Programming Example

The following example shows how to initialize a 4-mux LCD and display "09" on the LCD screen.

```
// Set pin to LCD function
LCD_C_setPinAsLCDFunctionEx(LCD_C_BASE, LCD_C_SEGMENT_LINE_0, LCD_C_SEGMENT_LINE_21);
LCD_C_setPinAsLCDFunctionEx(LCD_C_BASE, LCD_C_SEGMENT_LINE_26, LCD_C_SEGMENT_LINE_43);

LCD_C_InitParam initParams = {0};
initParams.clockSource = LCD_C_CLOCKSOURCE_ACLK;
initParams.clockDivider = LCD_C_CLOLKDIVIDER_1;
initParams.clockPrescaler = LCD_C_CLOCKPRESCALAR_16;
initParams.muxRate = LCD_C_4_MUX;
initParams.waveforms = LCD_C_LOW_POWER_WAVEFORMS;
initParams.segments = LCD_C_SEGMENTS_ENABLED;

LCD_C_init(LCD_C_BASE, &initParams);

// LCD Operation - VLCD generated internally, V2-V4 generated internally, v5 to ground
LCD_C_setVLCDSource(LCD_C_BASE, LCD_C_VLCD_GENERATED INTERNALLY, LCD_C_V2V3V4_GENERATED INTERNALLY,
LCD_C_V5_VSS);

// Set VLCD voltage to 2.60v
LCD_C_setVLCDVoltage(LCD_C_BASE, LCD_C_CHARGE_PUMP_VOLTAGE_2_60V_OR_2_17VREF);

// Enable charge pump and select internal reference for it
LCD_C_enableChargePump(LCD_C_BASE);
LCD_C_selectChargePumpReference(LCD_C_BASE, LCD_C_INTERNAL_REFERENCE_VOLTAGE);

LCD_C_configChargePump(LCD_C_BASE, LCD_C_SYNCHRONIZATION_ENABLED, 0);

// Clear LCD memory
LCD_C_clearMemory(LCD_C_BASE);
```

```
// Display "09"
LCD_C_setMemory(LCD_C_BASE, LCD_C_SEGMENT_LINE_8, 0xC);
LCD_C_setMemory(LCD_C_BASE, LCD_C_SEGMENT_LINE_9, 0xF);

LCD_C_setMemory(LCD_C_BASE, LCD_C_SEGMENT_LINE_12, 0x7);
LCD_C_setMemory(LCD_C_BASE, LCD_C_SEGMENT_LINE_13, 0xF);

//Turn LCD on
LCD_C_on(LCD_C_BASE);
```

19 Memory Protection Unit (MPU)

Introduction	224
API Functions	224
Programming Example	233

19.1 Introduction

The MPU protects against accidental writes to designated read-only memory segments or execution of code from a constant memory segment memory. Clearing the MPUENA bit disables the MPU, making the complete memory accessible for read, write, and execute operations. After a BOR, the complete memory is accessible without restrictions for read, write, and execute operations.

MPU features include:

- Main memory can be configured up to three segments of variable size
- Access rights for each segment can be set independently
- Information memory can have its access rights set independently
- All MPU registers are protected from access by password

This driver is contained in `mpu.c`, with `mpu.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	1218
TI Compiler 4.2.1	Size	824
TI Compiler 4.2.1	Speed	824
IAR 5.51.6	None	1058
IAR 5.51.6	Size	714
IAR 5.51.6	Speed	818
MSPGCC 4.8.0	None	2096
MSPGCC 4.8.0	Size	938
MSPGCC 4.8.0	Speed	1328

19.2 API Functions

Functions

- `uint16_t MPU_clearAllInterruptFlags` (`uint16_t baseAddress`)
- `uint16_t MPU_clearInterruptFlag` (`uint16_t baseAddress`, `uint16_t memAccFlag`)
- `void MPU_createThreeSegments` (`uint16_t baseAddress`, `uint16_t seg1boundary`, `uint16_t seg2boundary`, `uint8_t seg1accmask`, `uint8_t seg2accmask`, `uint8_t seg3accmask`)
- `void MPU_disablePUConViolation` (`uint16_t baseAddress`, `uint16_t segment`)
- `void MPU_enableNMlevent` (`uint16_t baseAddress`)
- `void MPU_enablePUConViolation` (`uint16_t baseAddress`, `uint16_t segment`)
- `uint16_t MPU_getInterruptStatus` (`uint16_t baseAddress`, `uint16_t memAccFlag`)
- `void MPU_initInfoSegment` (`uint16_t baseAddress`, `uint8_t accmask`)
- `void MPU_initThreeSegments` (`uint16_t baseAddress`, `MPU_initThreeSegmentsParam *param`)

- void [MPU_initTwoSegments](#) (uint16_t baseAddress, uint16_t seg1boundary, uint8_t seg1accmask, uint8_t seg2accmask)
- void [MPU_lockMPU](#) (uint16_t baseAddress)
- void [MPU_start](#) (uint16_t baseAddress)

19.2.1 Detailed Description

The MPU API is broken into three groups of functions: those that handle initialization, those that deal with memory segmentation and access rights definition, and those that handle interrupts. Please note that write access to all MPU registers is disabled after calling any MPU API.

The MPU initialization function is

- [MPU_start\(\)](#)

The MPU memory segmentation and access right definition functions are

- [MPU_initTwoSegments\(\)](#)
- [MPU_initThreeSegments\(\)](#)
- [MPU_initInfoSegment\(\)](#)

The MPU interrupt handler functions

- [MPU_enablePUCOnViolation\(\)](#)
- [MPU_disablePUCOnViolation\(\)](#)
- [MPU_getInterruptStatus\(\)](#)
- [MPU_clearInterruptFlag\(\)](#)
- [MPU_clearAllInterruptFlags\(\)](#)
- [MPU_enableNMlevent\(\)](#)

The MPU lock function is

- [MPU_lockMPU\(\)](#)

19.2.2 Function Documentation

19.2.2.1 uint16_t MPU_clearAllInterruptFlags (uint16_t *baseAddress*)

Clears all Memory Segment Access Violation Interrupt Flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	48
TI Compiler 4.2.1	Size	32
TI Compiler 4.2.1	Speed	32
IAR 5.51.6	None	40
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	60
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

Parameters:

baseAddress is the base address of the MPU module.

Modified bits of **MPUCTL1** register.

Returns:

Logical OR of any of the following:

- **MPU_SEG_1_ACCESS_VIOLATION** is set if an access violation in Main Memory Segment 1 is detected
 - **MPU_SEG_2_ACCESS_VIOLATION** is set if an access violation in Main Memory Segment 2 is detected
 - **MPU_SEG_3_ACCESS_VIOLATION** is set if an access violation in Main Memory Segment 3 is detected
 - **MPU_SEG_INFO_ACCESS_VIOLATION** is set if an access violation in User Information Memory Segment is detected
- indicating the status of the interrupt flags.

19.2.2.2 uint16_t MPU_clearInterruptFlag (uint16_t *baseAddress*, uint16_t *memAccFlag*)

Clears the masked interrupt flags.

Returns the memory segment violation flag status requested by the user or if user is providing a bit mask value, the function will return a value indicating if all flags were cleared.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	52
TI Compiler 4.2.1	Size	28
TI Compiler 4.2.1	Speed	28
IAR 5.51.6	None	36
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

Parameters:

baseAddress is the base address of the MPU module.

memAccFlag is the is the memory access violation flag. Mask value is the logical OR of any of the following:

- **MPU_SEG_1_ACCESS_VIOLATION** - is set if an access violation in Main Memory Segment 1 is detected
- **MPU_SEG_2_ACCESS_VIOLATION** - is set if an access violation in Main Memory Segment 2 is detected
- **MPU_SEG_3_ACCESS_VIOLATION** - is set if an access violation in Main Memory Segment 3 is detected
- **MPU_SEG_INFO_ACCESS_VIOLATION** - is set if an access violation in User Information Memory Segment is detected

Returns:

Logical OR of any of the following:

- **MPU_SEG_1_ACCESS_VIOLATION** is set if an access violation in Main Memory Segment 1 is detected
 - **MPU_SEG_2_ACCESS_VIOLATION** is set if an access violation in Main Memory Segment 2 is detected
 - **MPU_SEG_3_ACCESS_VIOLATION** is set if an access violation in Main Memory Segment 3 is detected
 - **MPU_SEG_INFO_ACCESS_VIOLATION** is set if an access violation in User Information Memory Segment is detected
- indicating the status of the masked flags.

19.2.2.3 void MPU_createThreeSegments (uint16_t *baseAddress*, uint16_t *seg1boundary*, uint16_t *seg2boundary*, uint8_t *seg1accmask*, uint8_t *seg2accmask*, uint8_t *seg3accmask*)

DEPRECATED - Initializes MPU with three memory segments.

This function creates three memory segments in FRAM allowing the user to set access right to each segment. To set the correct value for `seg1boundary`, the user must consult the Device Family User's Guide and provide the MPUSBx value corresponding to the memory address where the user wants to create the partition. Consult the "Segment Border Setting" section in the User's Guide to find the options available for MPUSBx.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	88
TI Compiler 4.2.1	Size	68
TI Compiler 4.2.1	Speed	68
IAR 5.51.6	None	72
IAR 5.51.6	Size	62
IAR 5.51.6	Speed	62
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	40
MSPGCC 4.8.0	Speed	384

Parameters:

baseAddress is the base address of the MPU module.

seg1boundary Valid values can be found in the Family User's Guide

seg2boundary Valid values can be found in the Family User's Guide

seg1accmask is the bit mask of access right for memory segment 1. Mask value is the logical OR of any of the following:

- **MPU_READ** - Read rights
- **MPU_WRITE** - Write rights
- **MPU_EXEC** - Execute rights
- **MPU_NO_READ_WRITE_EXEC** - no read/write/execute rights

seg2accmask is the bit mask of access right for memory segment 2. Mask value is the logical OR of any of the following:

- **MPU_READ** - Read rights
- **MPU_WRITE** - Write rights
- **MPU_EXEC** - Execute rights
- **MPU_NO_READ_WRITE_EXEC** - no read/write/execute rights

seg3accmask is the bit mask of access right for memory segment 3. Mask value is the logical OR of any of the following:

- **MPU_READ** - Read rights
- **MPU_WRITE** - Write rights
- **MPU_EXEC** - Execute rights
- **MPU_NO_READ_WRITE_EXEC** - no read/write/execute rights

Modified bits of **MPUSAM** register, bits of **MPUSEG** register and bits of **MPUCTL0** register.

Returns:

None

19.2.2.4 void MPU_disablePUCOnViolation (uint16_t baseAddress, uint16_t segment)

The following function disables PUC generation when an access violation has Occurred on the memory segment selected by the user.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	42
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	30
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Parameters:

baseAddress is the base address of the MPU module.

segment is the bit mask of memory segment that will NOT generate a PUC when an access violation occurs. Mask value is the logical OR of any of the following:

- **MPU_FIRST_SEG** - PUC generation on first memory segment
- **MPU_SECOND_SEG** - PUC generation on second memory segment
- **MPU_THIRD_SEG** - PUC generation on third memory segment
- **MPU_INFO_SEG** - PUC generation on user information memory segment

Modified bits of **MPUSAM** register and bits of **MPUCTL0** register.

Returns:

None

19.2.2.5 void MPU_enableNMlevent (uint16_t baseAddress)

The following function enables the NMI Event if a Segment violation has occurred.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	22
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

baseAddress is the base address of the MPU module.

Modified bits of **MPUCTL0** register.

Returns:

None

19.2.2.6 void MPU_enablePUCOnViolation (uint16_t baseAddress, uint16_t segment)

The following function enables PUC generation when an access violation has Occurred on the memory segment selected by the user.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	42
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	30
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	62
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Parameters:

baseAddress is the base address of the MPU module.

segment is the bit mask of memory segment that will generate a PUC when an access violation occurs. Mask value is the logical OR of any of the following:

- **MPU_FIRST_SEG** - PUC generation on first memory segment
- **MPU_SECOND_SEG** - PUC generation on second memory segment
- **MPU_THIRD_SEG** - PUC generation on third memory segment
- **MPU_INFO_SEG** - PUC generation on user information memory segment

Modified bits of **MPUSAM** register and bits of **MPUCTL0** register.

Returns:

None

19.2.2.7 uint16_t MPU_getInterruptStatus (uint16_t baseAddress, uint16_t memAccFlag)

Returns the memory segment violation flag status requested by the user.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the MPU module.

memAccFlag is the memory access violation flag. Mask value is the logical OR of any of the following:

- **MPU_SEG_1_ACCESS_VIOLATION** - is set if an access violation in Main Memory Segment 1 is detected
- **MPU_SEG_2_ACCESS_VIOLATION** - is set if an access violation in Main Memory Segment 2 is detected
- **MPU_SEG_3_ACCESS_VIOLATION** - is set if an access violation in Main Memory Segment 3 is detected
- **MPU_SEG_INFO_ACCESS_VIOLATION** - is set if an access violation in User Information Memory Segment is detected

Returns:

Logical OR of any of the following:

- **MPU_SEG_1_ACCESS_VIOLATION** is set if an access violation in Main Memory Segment 1 is detected
- **MPU_SEG_2_ACCESS_VIOLATION** is set if an access violation in Main Memory Segment 2 is detected
- **MPU_SEG_3_ACCESS_VIOLATION** is set if an access violation in Main Memory Segment 3 is detected

- **MPU_SEG_INFO_ACCESS_VIOLATION** is set if an access violation in User Information Memory Segment is detected indicating the status of the masked flags.

19.2.2.8 void MPU_initInfoSegment (uint16_t *baseAddress*, uint8_t *accmask*)

Initializes user information memory segment.

This function initializes user information memory segment with specified access rights.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	148
TI Compiler 4.2.1	Size	106
TI Compiler 4.2.1	Speed	106
IAR 5.51.6	None	134
IAR 5.51.6	Size	94
IAR 5.51.6	Speed	110
MSPGCC 4.8.0	None	282
MSPGCC 4.8.0	Size	130
MSPGCC 4.8.0	Speed	138

Parameters:

baseAddress is the base address of the MPU module.

accmask is the bit mask of access right for user information memory segment. Mask value is the logical OR of any of the following:

- **MPU_READ** - Read rights
- **MPU_WRITE** - Write rights
- **MPU_EXEC** - Execute rights
- **MPU_NO_READ_WRITE_EXEC** - no read/write/execute rights

Modified bits of **MPUSAM** register and bits of **MPUCTL0** register.

Returns:

None

19.2.2.9 void MPU_initThreeSegments (uint16_t *baseAddress*, MPU_initThreeSegmentsParam * *param*)

Initializes MPU with three memory segments.

This function creates three memory segments in FRAM allowing the user to set access right to each segment. To set the correct value for seg1boundary, the user must consult the Device Family User's Guide and provide the MPUSBx value corresponding to the memory address where the user wants to create the partition. Consult the "Segment Border Setting" section in the User's Guide to find the options available for MPUSBx.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	408
TI Compiler 4.2.1	Size	296
TI Compiler 4.2.1	Speed	296
IAR 5.51.6	None	378
IAR 5.51.6	Size	268
IAR 5.51.6	Speed	276
MSPGCC 4.8.0	None	794
MSPGCC 4.8.0	Size	362
MSPGCC 4.8.0	Speed	382

Parameters:

baseAddress is the base address of the MPU module.

param is the pointer to struct for initializing three segments.

Modified bits of **MPUSAM** register, bits of **MPUSEG** register and bits of **MPUCTL0** register.

Returns:

None

19.2.2.10 void MPU_initTwoSegments (uint16_t baseAddress, uint16_t seg1boundary, uint8_t seg1accmask, uint8_t seg2accmask)

Initializes MPU with two memory segments.

This function creates two memory segments in FRAM allowing the user to set access right to each segment. To set the correct value for seg1boundary, the user must consult the Device Family User's Guide and provide the MPUSBx value corresponding to the memory address where the user wants to create the partition. Consult the "Segment Border Setting" section in the User's Guide to find the options available for MPUSBx.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	286
TI Compiler 4.2.1	Size	198
TI Compiler 4.2.1	Speed	198
IAR 5.51.6	None	264
IAR 5.51.6	Size	194
IAR 5.51.6	Speed	234
MSPGCC 4.8.0	None	552
MSPGCC 4.8.0	Size	250
MSPGCC 4.8.0	Speed	268

Parameters:

baseAddress is the base address of the MPU module.

seg1boundary Valid values can be found in the Family User's Guide

seg1accmask is the bit mask of access right for memory segment 1. Mask value is the logical OR of any of the following:

- **MPU_READ** - Read rights
- **MPU_WRITE** - Write rights
- **MPU_EXEC** - Execute rights
- **MPU_NO_READ_WRITE_EXEC** - no read/write/execute rights

seg2accmask is the bit mask of access right for memory segment 2 Mask value is the logical OR of any of the following:

- **MPU_READ** - Read rights
- **MPU_WRITE** - Write rights
- **MPU_EXEC** - Execute rights
- **MPU_NO_READ_WRITE_EXEC** - no read/write/execute rights

Modified bits of **MPUSAM** register, bits of **MPUSEG** register and bits of **MPUCTL0** register.

Returns:

None

19.2.2.11 void MPU_lockMPU (uint16_t *baseAddress*)

Lock MPU to protect from write access.

Sets MPULOCK to protect MPU from write access on all MPU registers except MPUCTL1, MPUIPC0 and MPUIPSEGBx until a BOR occurs. MPULOCK bit cannot be cleared manually. [MPU_clearInterruptFlag\(\)](#) and [MPU_clearAllInterruptFlags\(\)](#) still can be used after this API is called.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	22
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

baseAddress is the base address of the MPU module.

Modified bits are **MPULOCK** of **MPUCTL1** register.

Returns:

None

19.2.2.12 void MPU_start (uint16_t *baseAddress*)

The following function enables the MPU module in the device.

This function needs to be called once all memory segmentation has been done. If this function is not called the MPU module will not be activated.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	22
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

baseAddress is the base address of the MPU module.

Modified bits of **MPUCTL0** register.

Returns:

None

19.3 Programming Example

The following example shows some MPU operations using the APIs

```
//Initialize struct for three segments configuration
MPU_initThreeSegmentsParam threeSegParam;
threeSegParam.seg1boundary = 0x0600;
threeSegParam.seg1boundary = 0x0800;
threeSegParam.seg1accmask = MPU_READ|MPU_WRITE|MPU_EXEC;
threeSegParam.seg2accmask = MPU_READ;
threeSegParam.seg3accmask = MPU_READ|MPU_WRITE|MPU_EXEC;

//Define memory segment boundaries and set access right for each memory segment
MPU_initThreeSegments(MPU_BASE,&threeSegParam);

// Configures MPU to generate a PUC on access violation on the second segment
MPU_enablePUConViolation(MPU_BASE,MPU_SECOND_SEG);

//Enables the MPU module
MPU_start(MPU_BASE);
```

20 32-Bit Hardware Multiplier (MPY32)

Introduction	234
API Functions	234
Programming Example	247

20.1 Introduction

The 32-Bit Hardware Multiplier (MPY32) API provides a set of functions for using the MSP430Ware MPY32 modules. Functions are provided to setup the MPY32 modules, set the operand registers, and obtain the results.

The MPY32 Modules does not generate any interrupts.

This driver is contained in `mpy32.c`, with `mpy32.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	578
TI Compiler 4.2.1	Size	294
TI Compiler 4.2.1	Speed	294
IAR 5.51.6	None	404
IAR 5.51.6	Size	250
IAR 5.51.6	Speed	274
MSPGCC 4.8.0	None	1702
MSPGCC 4.8.0	Size	644
MSPGCC 4.8.0	Speed	642

20.2 API Functions

Functions

- void [MPY32_clearCarryBitValue](#) (void)
- void [MPY32_disableFractionalMode](#) (void)
- void [MPY32_disableSaturationMode](#) (void)
- void [MPY32_enableFractionalMode](#) (void)
- void [MPY32_enableSaturationMode](#) (void)
- uint16_t [MPY32_getCarryBitValue](#) (void)
- uint8_t [MPY32_getFractionalMode](#) (void)
- uint64_t [MPY32_getResult](#) (void)
- uint16_t [MPY32_getResult16Bit](#) (void)
- uint32_t [MPY32_getResult24Bit](#) (void)
- uint32_t [MPY32_getResult32Bit](#) (void)
- uint64_t [MPY32_getResult64Bit](#) (void)
- uint8_t [MPY32_getResult8Bit](#) (void)
- uint8_t [MPY32_getSaturationMode](#) (void)
- uint16_t [MPY32_getSumExtension](#) (void)
- void [MPY32_preloadResult](#) (uint64_t result)
- void [MPY32_setOperandOne16Bit](#) (uint8_t multiplicationType, uint16_t operand)

- void [MPY32_setOperandOne24Bit](#) (uint8_t multiplicationType, uint32_t operand)
- void [MPY32_setOperandOne32Bit](#) (uint8_t multiplicationType, uint32_t operand)
- void [MPY32_setOperandOne8Bit](#) (uint8_t multiplicationType, uint8_t operand)
- void [MPY32_setOperandTwo16Bit](#) (uint16_t operand)
- void [MPY32_setOperandTwo24Bit](#) (uint32_t operand)
- void [MPY32_setOperandTwo32Bit](#) (uint32_t operand)
- void [MPY32_setOperandTwo8Bit](#) (uint8_t operand)
- void [MPY32_setWriteDelay](#) (uint16_t writeDelaySelect)

20.2.1 Detailed Description

The MPY32 API is broken into three groups of functions: those that control the settings, those that set the operand registers, and those that return the results, sum extension, and carry bit value.

The settings are handled by

- [MPY32_setWriteDelay](#)()
- [MPY32_enableSaturationMode](#)()
- [MPY32_disableSaturationMode](#)()
- [MPY32_enableFractionalMode](#)()
- [MPY32_disableFractionalMode](#)()
- [MPY32_preloadResult](#)()

The operand registers are set by

- [MPY32_setOperandOne8Bit](#)()
- [MPY32_setOperandOne16Bit](#)()
- [MPY32_setOperandOne24Bit](#)()
- [MPY32_setOperandOne32Bit](#)()
- [MPY32_setOperandTwo8Bit](#)()
- [MPY32_setOperandTwo16Bit](#)()
- [MPY32_setOperandTwo24Bit](#)()
- [MPY32_setOperandTwo32Bit](#)()

The results can be returned by

- [MPY32_getResult](#)()
- [MPY32_getSumExtension](#)()
- [MPY32_getCarryBitValue](#)()
- [MPY32_getSaturationMode](#)()
- [MPY32_getFractionalMode](#)()

20.2.2 Function Documentation

20.2.2.1 void MPY32_clearCarryBitValue (void)

Clears the Carry Bit of the last multiplication operation.

This function clears the Carry Bit of the MPY module

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

The value of the MPY32 module Carry Bit 0x0 or 0x1.

20.2.2.2 void MPY32_disableFractionalMode (void)

Disables Fraction Mode.

This function disables fraction mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

20.2.2.3 void MPY32_disableSaturationMode (void)

Disables Saturation Mode.

This function disables saturation mode, which allows the raw result of the MPY result registers to be returned.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

20.2.2.4 void MPY32_enableFractionalMode (void)

Enables Fraction Mode.

This function enables fraction mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

20.2.2.5 void MPY32_enableSaturationMode (void)

Enables Saturation Mode.

This function enables saturation mode. When this is enabled, the result read out from the MPY result registers is converted to the most-positive number in the case of an overflow, or the most-negative number in the case of an underflow. Please note, that the raw value in the registers does not reflect the result returned, and if the saturation mode is disabled, then the raw value of the registers will be returned instead.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

20.2.2.6 uint16_t MPY32_getCarryBitValue (void)

Returns the Carry Bit of the last multiplication operation.

This function returns the Carry Bit of the MPY module, which either gives the sign after a signed operation or shows a carry after a multiply- and- accumulate operation.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	8
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	10
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Returns:

The value of the MPY32 module Carry Bit 0x0 or 0x1.

20.2.2.7 uint8_t MPY32_getFractionalMode (void)

Gets the Fractional Mode.

This function gets the current fractional mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	8
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	16
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Returns:

Gets the fractional mode Return one of the following:

- **MPY32_FRACTIONAL_MODE_DISABLED**
 - **MPY32_FRACTIONAL_MODE_ENABLED**
- Gets the Fractional Mode

20.2.2.8 uint64_t MPY32_getResult (void)

Returns an 64-bit result of the last multiplication operation.

This function returns all 64 bits of the result registers

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	114
TI Compiler 4.2.1	Size	42
TI Compiler 4.2.1	Speed	42
IAR 5.51.6	None	102
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	694
MSPGCC 4.8.0	Size	242
MSPGCC 4.8.0	Speed	240

Returns:

The 64-bit result is returned as a uint64_t type

20.2.2.9 uint16_t MPY32_getResult16Bit (void)

Deprecated - Returns an 16-bit result of the last multiplication operation.

This function returns the 16 least significant bits of the result registers. This can improve efficiency if the operation has no more than a 16-bit result.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	8
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

The 16-bit result of the last multiplication operation.

20.2.2.10 uint32_t MPY32_getResult24Bit (void)

Deprecated - Returns an 24-bit result of the last multiplication operation.

This function returns the 24 least significant bits of the result registers. This can improve efficiency if the operation has no more than an 24-bit result.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	36
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	22
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

Returns:

The 24-bit result of the last multiplication operation.

20.2.2.11 uint32_t MPY32_getResult32Bit (void)

Deprecated - Returns an 32-bit result of the last multiplication operation.

This function returns a 32-bit result of the last multiplication operation, which is the maximum amount of bits of a 16 x 16 operation.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	36
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	22
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

Returns:

The 32-bit result of the last multiplication operation.

20.2.2.12 uint64 MPY32_getResult64Bit (void)

Deprecated - Returns an 64-bit result of the last multiplication operation.

This function returns all 64 bits of the result registers. The way this is passed is with 4 integers contained within a uint16 struct.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	42
TI Compiler 4.2.1	Size	42
TI Compiler 4.2.1	Speed	42
IAR 5.51.6	None	52
IAR 5.51.6	Size	52
IAR 5.51.6	Speed	52
MSPGCC 4.8.0	None	82
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

Returns:

The 64-bit result separated into 4 uint16_ts in a uint16 struct

20.2.2.13 uint8_t MPY32_getResult8Bit (void)

Deprecated - Returns an 8-bit result of the last multiplication operation.

This function returns the 8 least significant bits of the result registers. This can improve efficiency if the operation has no more than an 8-bit result.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	10
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

The 8-bit result of the last multiplication operation.

20.2.2.14 uint8_t MPY32_getSaturationMode (void)

Gets the Saturation Mode.

This function gets the current saturation mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	8
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	16
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Returns:

Gets the Saturation Mode Return one of the following:

- **MPY32_SATURATION_MODE_DISABLED**
- **MPY32_SATURATION_MODE_ENABLED**

Gets the Saturation Mode

20.2.2.15 uint16_t MPY32_getSumExtension (void)

Returns the Sum Extension of the last multiplication operation.

This function returns the Sum Extension of the MPY module, which either gives the sign after a signed operation or shows a carry after a multiply- and-accumulate operation. The Sum Extension acts as a check for overflows or underflows.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	8
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

The value of the MPY32 module Sum Extension.

20.2.2.16 void MPY32_preloadResult (uint64_t *result*)

Preloads the result register.

This function Preloads the result register

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	52
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	198
MSPGCC 4.8.0	Size	78
MSPGCC 4.8.0	Speed	78

Returns:

None

20.2.2.17 void MPY32_setOperandOne16Bit (uint8_t *multiplicationType*, uint16_t *operand*)

Sets an 16-bit value into operand 1.

This function sets the first operand for multiplication and determines what type of operation should be performed. Once the second operand is set, then the operation will begin.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

multiplicationType is the type of multiplication to perform once the second operand is set. Valid values are:

- **MPY32_MULTIPLY_UNSIGNED**
- **MPY32_MULTIPLY_SIGNED**
- **MPY32_MULTIPLYACCUMULATE_UNSIGNED**
- **MPY32_MULTIPLYACCUMULATE_SIGNED**

operand is the 16-bit value to load into the 1st operand.

Returns:

None

20.2.2.18 void MPY32_setOperandOne24Bit (uint8_t *multiplicationType*, uint32_t *operand*)

Sets an 24-bit value into operand 1.

This function sets the first operand for multiplication and determines what type of operation should be performed. Once the second operand is set, then the operation will begin.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	46
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	24
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	38
MSPGCC 4.8.0	Speed	38

Parameters:

multiplicationType is the type of multiplication to perform once the second operand is set. Valid values are:

- **MPY32_MULTIPLY_UNSIGNED**
- **MPY32_MULTIPLY_SIGNED**
- **MPY32_MULTIPLYACCUMULATE_UNSIGNED**
- **MPY32_MULTIPLYACCUMULATE_SIGNED**

operand is the 24-bit value to load into the 1st operand.

Returns:

None

20.2.2.19 void MPY32_setOperandOne32Bit (uint8_t *multiplicationType*, uint32_t *operand*)

Sets an 32-bit value into operand 1.

This function sets the first operand for multiplication and determines what type of operation should be performed. Once the second operand is set, then the operation will begin.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	44
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	24
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	82
MSPGCC 4.8.0	Size	38
MSPGCC 4.8.0	Speed	38

Parameters:

multiplicationType is the type of multiplication to perform once the second operand is set. Valid values are:

- **MPY32_MULTIPLY_UNSIGNED**
- **MPY32_MULTIPLY_SIGNED**
- **MPY32_MULTIPLYACCUMULATE_UNSIGNED**
- **MPY32_MULTIPLYACCUMULATE_SIGNED**

operand is the 32-bit value to load into the 1st operand.

Returns:
None

20.2.2.20 void MPY32_setOperandOne8Bit (uint8_t *multiplicationType*, uint8_t *operand*)

Sets an 8-bit value into operand 1.

This function sets the first operand for multiplication and determines what type of operation should be performed. Once the second operand is set, then the operation will begin.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

multiplicationType is the type of multiplication to perform once the second operand is set. Valid values are:

- **MPY32_MULTIPLY_UNSIGNED**
- **MPY32_MULTIPLY_SIGNED**
- **MPY32_MULTIPLYACCUMULATE_UNSIGNED**
- **MPY32_MULTIPLYACCUMULATE_SIGNED**

operand is the 8-bit value to load into the 1st operand.

Returns:
None

20.2.2.21 void MPY32_setOperandTwo16Bit (uint16_t *operand*)

Sets an 16-bit value into operand 2, which starts the multiplication.

This function sets the second operand of the multiplication operation and starts the operation.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

operand is the 16-bit value to load into the 2nd operand.

Returns:
None

20.2.2.22 void MPY32_setOperandTwo24Bit (uint32_t *operand*)

Sets an 24-bit value into operand 2, which starts the multiplication.

This function sets the second operand of the multiplication operation and starts the operation.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:
operand is the 24-bit value to load into the 2nd operand.

Returns:
None

20.2.2.23 void MPY32_setOperandTwo32Bit (uint32_t *operand*)

Sets an 32-bit value into operand 2, which starts the multiplication.

This function sets the second operand of the multiplication operation and starts the operation.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	54
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:
operand is the 32-bit value to load into the 2nd operand.

Returns:
None

20.2.2.24 void MPY32_setOperandTwo8Bit (uint8_t *operand*)

Sets an 8-bit value into operand 2, which starts the multiplication.

This function sets the second operand of the multiplication operation and starts the operation.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	20
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

operand is the 8-bit value to load into the 2nd operand.

Returns:

None

20.2.2.25 void MPY32_setWriteDelay (uint16_t *writeDelaySelect*)

Sets the write delay setting for the MPY32 module.

This function sets up a write delay to the MPY module's registers, which holds any writes to the registers until all calculations are complete. There are two different settings, one which waits for 32-bit results to be ready, and one which waits for 64-bit results to be ready. This prevents unpredictable results if registers are changed before the results are ready.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

writeDelaySelect delays the write to any MPY32 register until the selected bit size of result has been written. Valid values are:

- **MPY32_WRITEDELAY_OFF** [Default] - writes are not delayed
 - **MPY32_WRITEDELAY_32BIT** - writes are delayed until a 32-bit result is available in the result registers
 - **MPY32_WRITEDELAY_64BIT** - writes are delayed until a 64-bit result is available in the result registers
- Modified bits are **MPYDLY32** and **MPYDLYWRNEN** of **MPY32CTL0** register.

Returns:

None

20.3 Programming Example

The following example shows how to initialize and use the MPY32 API to calculate a 16-bit by 16-bit unsigned multiplication operation.

```
WDT_hold(WDT_A_BASE);    // Stop WDT

// Set a 16-bit Operand into the specific Operand 1 register to specify
// unsigned multiplication
MPY32_setOperandOne16Bit(MPY32_MULTIPLY_UNSIGNED,
                        0x1234);
// Set Operand 2 to begin the multiplication operation
MPY32_setOperandTwo16Bit(0x5678);

__bis_SR_register(LPM4_bits);    // Enter LPM4
__no_operation();                // BREAKPOINT HERE to verify the
                                // correct result in the registers
```

21 Power Management Module (PMM)

Introduction	248
API Functions	248
Programming Example	254

21.1 Introduction

The PMM manages all functions related to the power supply and its supervision for the device. Its primary functions are first to generate a supply voltage for the core logic, and second, provide several mechanisms for the supervision of the voltage applied to the device (DVCC).

The PMM uses an integrated low-dropout voltage regulator (LDO) to produce a secondary core voltage (VCORE) from the primary one applied to the device (DVCC). In general, VCORE supplies the CPU, memories, and the digital modules, while DVCC supplies the I/Os and analog modules. The VCORE output is maintained using a dedicated voltage reference. The input or primary side of the regulator is referred to as its high side. The output or secondary side is referred to as its low side.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	184
TI Compiler 4.2.1	Size	168
TI Compiler 4.2.1	Speed	172
IAR 5.51.6	None	168
IAR 5.51.6	Size	112
IAR 5.51.6	Speed	130
MSPGCC 4.8.0	None	420
MSPGCC 4.8.0	Size	204
MSPGCC 4.8.0	Speed	204

21.2 API Functions

Functions

- void [PMM_clearInterrupt](#) (uint16_t mask)
- void [PMM_disableLowPowerReset](#) (void)
- void [PMM_disableSVSH](#) (void)
- void [PMM_enableLowPowerReset](#) (void)
- void [PMM_enableSVSH](#) (void)
- uint16_t [PMM_getInterruptStatus](#) (uint16_t mask)
- void [PMM_regOff](#) (void)
- void [PMM_regOn](#) (void)
- void [PMM_trigBOR](#) (void)
- void [PMM_trigPOR](#) (void)
- void [PMM_unlockLPM5](#) (void)

21.2.1 Detailed Description

PMM_enableLowPowerReset() / **PMM_disableLowPowerReset()** If enabled, SVSH does not reset device but triggers a system NMI. If disabled, SVSH resets device.

PMM_enableSVSH() / **PMM_disableSVSH()** If disabled on FR58xx/FR59xx, High-side SVS (SVSH) is disabled in LPM2, LPM3, LPM4, LPM3.5 and LPM4.5. SVSH is always enabled in active mode, LPM0, and LPM1. If enabled, SVSH is always enabled. Note: this API has different functionality depending on the part.

PMM_regOff() / **PMM_regOn()** If off, Regulator is turned off when going to LPM3/4. System enters LPM3.5 or LPM4.5, respectively. If on, Regulator remains on when going into LPM3/4

PMM_clearInterrupt() Clear selected or all interrupt flags for the PMM

PMM_getInterruptStatus() Returns interrupt status of the selected flag in the PMM module

PMM_lockLPM5() / **PMM_unlockLPM5()** If unlocked, LPMx.5 configuration is not locked and defaults to its reset condition. if locked, LPMx.5 configuration remains locked. Pin state is held during LPMx.5 entry and exit.

This driver is contained in `pmm.c`, with `pmm.h` containing the API definitions for use by applications.

21.2.2 Function Documentation

21.2.2.1 void PMM_clearInterrupt (uint16_t mask)

Clears interrupt flags for the PMM.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	16
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Parameters:

mask is the mask for specifying the required flag Mask value is the logical OR of any of the following:

- **PMM_BOR_INTERRUPT** - Software BOR interrupt
- **PMM_RST_INTERRUPT** - RESET pin interrupt
- **PMM_POR_INTERRUPT** - Software POR interrupt
- **PMM_SVSH_INTERRUPT** - SVS high side interrupt
- **PMM_LPM5_INTERRUPT** - LPM5 indication
- **PMM_ALL** - All interrupts

Modified bits of **PMMCTL0** register and bits of **PMMIFG** register.

Returns:

None

21.2.2.2 void PMM_disableLowPowerReset (void)

Disables the low power reset. SVSH resets device.

Modified bits of **PMMCTL0** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Returns:

None

21.2.2.3 void PMM_disableSVSH (void)

Disables the high-side SVS circuitry.

Modified bits of **PMMCTL0** register.**Code Metrics:**

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Returns:

None

21.2.2.4 void PMM_enableLowPowerReset (void)

Enables the low power reset. SVSH does not reset device, but triggers a system NMI.

Modified bits of **PMMCTL0** register.**Code Metrics:**

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Returns:
None

21.2.2.5 void PMM_enableSVSH (void)

Enables the high-side SVS circuitry.

Modified bits of **PMMCTL0** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Returns:
None

21.2.2.6 uint16_t PMM_getInterruptStatus (uint16_t *mask*)

Returns interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

mask is the mask for specifying the required flag Mask value is the logical OR of any of the following:

- **PMM_BOR_INTERRUPT** - Software BOR interrupt
- **PMM_RST_INTERRUPT** - RESET pin interrupt
- **PMM_POR_INTERRUPT** - Software POR interrupt
- **PMM_SVSH_INTERRUPT** - SVS high side interrupt
- **PMM_LPM5_INTERRUPT** - LPM5 indication
- **PMM_ALL** - All interrupts

Returns:

Logical OR of any of the following:

- **PMM_BOR_INTERRUPT** Software BOR interrupt
- **PMM_RST_INTERRUPT** RESET pin interrupt

- **PMM_POR_INTERRUPT** Software POR interrupt
- **PMM_SVSH_INTERRUPT** SVS high side interrupt
- **PMM_LPM5_INTERRUPT** LPM5 indication
- **PMM_ALL** All interrupts
indicating the status of the selected interrupt flags

21.2.2.7 void PMM_regOff (void)

Turns OFF the low-dropout voltage regulator (LDO) when going into LPM3/4, thus the system will enter LPM3.5 or LPM4.5 respectively.

Modified bits of **PMMCTL0** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Returns:

None

21.2.2.8 void PMM_regOn (void)

Makes the low-dropout voltage regulator (LDO) remain ON when going into LPM 3/4.

Modified bits of **PMMCTL0** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	18
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Returns:

None

21.2.2.9 void PMM_trigBOR (void)

Calling this function will trigger a software Brown Out Rest (BOR).

Modified bits of **PMMCTL0** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	16
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	40
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Returns:

None

21.2.2.10 void PMM_trigPOR (void)

Calling this function will trigger a software Power On Reset (POR).

Modified bits of **PMMCTL0** register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	16
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	40
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Returns:

None

21.2.2.11 void PMM_unlockLPM5 (void)

Unlock LPM5.

LPMx.5 configuration is not locked and defaults to its reset condition. Disable the GPIO power-on default high-impedance mode to activate previously configured port settings.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

21.3 Programming Example

```

//Unlock the GPIO pins.
/*
Base Address of Comparator D,
By default, the pins are unlocked unless waking
up from an LPMx.5 state in which case all GPIO
are previously locked.

*/
PMM_unlockLPM5(PMM_BASE);

//Get Interrupt Status from the PMMIFG register.
/* Base Address of Comparator D,
mask:
PMM_PMMBORIFG
PMM_PMMRSTIFG,
PMM_PMPORIFG,
PMM_SVSLIFG,
PMM_SVSHIFG
PMM_PMMLPM5IFG,
return STATUS_SUCCESS (0x01) or STATUS_FAIL (0x00)
*/
if (PMM_getInterruptStatus(PMM_BASE, PMM_PMMLPM5IFG)) // Was this device in LPMx.5 mode before t
{
//Clear Interrupt Flag from the PMMIFG register.
/* Base Address of Comparator D,
mask:
PMM_PMMBORIFG
PMM_PMMRSTIFG,
PMM_PMPORIFG,
PMM_SVSLIFG,
PMM_SVSHIFG
PMM_PMMLPM5IFG,
PMM_ALL
*/
PMM_clearInterrupt(PMM_BASE, PMM_PMMLPM5IFG); // Clear the LPMx.5 flag
}

if (PMM_getInterruptStatus(PMM_BASE, PMM_PMMRSTIFG)) // Was this reset triggered by the Reset
{
PMM_clearInterrupt(PMM_BASE, PMM_PMMRSTIFG); // Clear reset flag

__delay_cycles(1000000);
//Lock GPIO output states (before triggering a BOR)
/*

```

```

Base Address of Comparator D,
Forces all GPIO to retain their output
states during a reset.
    */
    PMM_lockLPM5(PMM_BASE);
    //Trigger a software Brown Out Reset (BOR)
    /*
Base Address of Comparator D,
Forces the devices to perform a BOR.
    */
    PMM_trigBOR(PMM_BASE);                                     // Software trigger BOR

}

if (PMM_getInterruptStatus(PMM_BASE, PMM_PMMBORIFG))           // Was this reset triggered by the BOR flag
{
    PMM_clearInterrupt(PMM_BASE, PMM_PMMBORIFG);               // Clear BOR flag

    __delay_cycles(1000000);

    PMM_lockLPM5(PMM_BASE);

    //Disable SVSH
    /*
Base Address of Comparator D,
High-side SVS (SVSH) is disabled in LPM4.5. SVSH is
always enabled in active mode and LPM0/1/2/3/4 and LPM3.5.
    */
    PMM_disableSVSH(PMM_BASE);
    //Disable SVSL
    /*
Base Address of Comparator D,
Low-side SVS (SVSL) is disabled in low power modes.
SVSL is always enabled in active mode and LPM0.
    */
    PMM_disableSVSL(PMM_BASE);
    //Disable Regulator
    /*
Base Address of Comparator D,
Regulator is turned off when going to LPM3/4.
System enters LPM3.5 or LPM4.5, respectively.
    */
    PMM_regOff(PMM_BASE);
    __bis_SR_register(LPM4_bits); // Enter LPM4.5, This automatically locks
                                   // (if not locked already) all GPIO pins
                                   // and will set the LPM5 flag and set the PM5CTL0
                                   // in the PM5CTL0 register upon wake up
}
//-----
while (1)
{
    __no_operation();           // Don't sleep
}

```

22 RAM Controller

Introduction	256
API Functions	256
Programming Example	257

22.1 Introduction

The RAMCTL provides access to the different power modes of the RAM. The RAMCTL allows the ability to reduce the leakage current while the CPU is off. The RAM can also be switched off. In retention mode, the RAM content is saved while the RAM content is lost in off mode. The RAM is partitioned in sectors, typically of 4KB (sector) size. See the device-specific data sheet for actual block allocation and size. Each sector is controlled by the RAM controller RAM Sector Off control bit (RCRSyOFF) of the RAMCTL Control 0 register (RCCTL0). The RCCTL0 register is protected with a key. Only if the correct key is written during a word write, the RCCTL0 register content can be modified. Byte write accesses or write accesses with a wrong key are ignored.

This driver is contained in `ramcontroller.c`, with `ramcontroller.h` containing the API definitions for use by applications.

22.2 API Functions

Functions

- `uint8_t RAM_getSectorState (uint8_t sector)`
- `void RAM_setSectorOff (uint8_t sector, uint8_t mode)`

22.2.1 Detailed Description

The MSP430ware API that configure the RAM controller are:

`RAM_setSectorOff()` - Set specified RAM sector off `RAM_getSectorState()` - Get RAM sector ON/OFF status

22.2.2 Function Documentation

22.2.2.1 `uint8_t RAM_getSectorState (uint8_t sector)`

Get RAM sector off status.

Parameters:

sector is specified sector to get status. Valid values are:

- `RAM_SECTOR0`
- `RAM_SECTOR1`
- `RAM_SECTOR2`
- `RAM_SECTOR3`

Returns:

Return one of the following:

- `RAM_RETENTION_MODE`
- `RAM_OFF_WAKEUP_MODE`
- `RAM_OFF_NON_WAKEUP_MODE`
indicating the status of the masked sectors

22.2.2.2 void RAM_setSectorOff (uint8_t sector, uint8_t mode)

Set specified RAM sector off.

Parameters:

sector is specified sector to be set off. Valid values are:

- RAM_SECTOR0
- RAM_SECTOR1
- RAM_SECTOR2
- RAM_SECTOR3

mode is sector off mode. Valid values are:

- RAM_RETENTION_MODE
- RAM_OFF_WAKEUP_MODE
- RAM_OFF_NON_WAKEUP_MODE

Modified bits of **RCCTL0** register.

Returns:

None

22.3 Programming Example

The following example shows some RAM Controller operations using the APIs

```
//Start timer
Timer_startUpMode(  TIMER_B0_BASE,
                    TIMER_CLOCKSOURCE_ACLK,
                    TIMER_CLOCKSOURCE_DIVIDER_1,
                    25000,
                    TIMER_TAIE_INTERRUPT_DISABLE,
                    TIMER_CAPTURECOMPARE_INTERRUPT_ENABLE,
                    TIMER_DO_CLEAR
                    );

//set RAM controller sector0 retention mode
RAM_setSectorOff(RAM_SECTOR0, RAM_RETENTION_MODE);

//Enter LPM0, enable interrupts
__bis_SR_register(LPM3_bits + GIE);

//For debugger
__no_operation();
}

//*****
//
//This is the Timer B0 interrupt vector service routine.
//
//*****
#pragma vector=TIMERB0_VECTOR
__interrupt void TIMERB0_ISR (void)
{
    returnValue = RAM_getSectorState(RAM_SECTOR0);
}
```

23 Internal Reference (REF_A)

Introduction	258
API Functions	258
Programming Example	267

23.1 Introduction

The Internal Reference (REF_A) API provides a set of functions for using the MSP430Ware REF_A modules. Functions are provided to setup and enable use of the Reference voltage, enable or disable the internal temperature sensor, and view the status of the inner workings of the REF_A module.

The reference module (REF) is responsible for generation of all critical reference voltages that can be used by various analog peripherals in a given device. The heart of the reference system is the bandgap from which all other references are derived by unity or non-inverting gain stages. The REFGEN sub-system consists of the bandgap, the bandgap bias, and the non-inverting buffer stage which generates the three primary voltage reference available in the system, namely 1.2 V, 2.0 V, and 2.5 V. In addition, when enabled, a buffered bandgap voltage is available.

This driver is contained in `ref_a.c`, with `ref_a.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	302
TI Compiler 4.2.1	Size	148
TI Compiler 4.2.1	Speed	148
IAR 5.51.6	None	168
IAR 5.51.6	Size	110
IAR 5.51.6	Speed	110
MSPGCC 4.8.0	None	434
MSPGCC 4.8.0	Size	126
MSPGCC 4.8.0	Speed	126

23.2 API Functions

Functions

- void [REF_A_disableReferenceVoltage](#) (uint16_t baseAddress)
- void [REF_A_disableReferenceVoltageOutput](#) (uint16_t baseAddress)
- void [REF_A_disableTempSensor](#) (uint16_t baseAddress)
- void [REF_A_enableReferenceVoltage](#) (uint16_t baseAddress)
- void [REF_A_enableReferenceVoltageOutput](#) (uint16_t baseAddress)
- void [REF_A_enableTempSensor](#) (uint16_t baseAddress)
- uint16_t [REF_A_getBandgapMode](#) (uint16_t baseAddress)
- bool [REF_A_isBandgapActive](#) (uint16_t baseAddress)
- bool [REF_A_isBufferedBandgapVoltageReady](#) (uint16_t baseAddress)
- bool [REF_A_isRefGenActive](#) (uint16_t baseAddress)
- uint16_t [REF_A_isRefGenBusy](#) (uint16_t baseAddress)
- bool [REF_A_isVariableReferenceVoltageOutputReady](#) (uint16_t baseAddress)

- void [REF_A_setBufferedBandgapVoltageOneTimeTrigger](#) (uint16_t baseAddress)
- void [REF_A_setReferenceVoltage](#) (uint16_t baseAddress, uint8_t referenceVoltageSelect)
- void [REF_A_setReferenceVoltageOneTimeTrigger](#) (uint16_t baseAddress)

23.2.1 Detailed Description

The REF_A API is broken into three groups of functions: those that deal with the reference voltage, those that handle the internal temperature sensor, and those that return the status of the REF_A module.

The reference voltage of the REF_A module is handled by

- [REF_A_setReferenceVoltage\(\)](#)
- [REF_A_enableReferenceVoltageOutput\(\)](#)
- [REF_A_disableReferenceVoltageOutput\(\)](#)
- [REF_A_enableReferenceVoltage\(\)](#)
- [REF_A_disableReferenceVoltage\(\)](#)

The internal temperature sensor is handled by

- [REF_A_disableTempSensor\(\)](#)
- [REF_A_enableTempSensor\(\)](#)

The status of the REF_A module is handled by

- [REF_A_getBandgapMode\(\)](#)
- [REF_A_isBandgapActive\(\)](#)
- [REF_A_isRefGenBusy\(\)](#)
- [REF_A_isRefGenActive\(\)](#)
- [REF_A_getBufferedBandgapVoltageStatus\(\)](#)
- [REF_A_getVariableReferenceVoltageStatus\(\)](#)
- [REF_A_setReferenceVoltageOneTimeTrigger\(\)](#)
- [REF_A_setBufBandgapVoltageOneTimeTrigger\(\)](#)

23.2.2 Function Documentation

23.2.2.1 void REF_A_disableReferenceVoltage (uint16_t *baseAddress*)

Disables the reference voltage.

This function is used to disable the generated reference voltage. Please note, if the [REF_A_isRefGenBusy\(\)](#) returns REF_A_BUSY, this function will have no effect.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:**baseAddress** is the base address of the REF module.Modified bits are **REFON** of **REFCTL0** register.**Returns:**

None

23.2.2.2 void REF_A_disableReferenceVoltageOutput (uint16_t baseAddress)

Disables the reference voltage as an output to a pin.

This function is used to disables the reference voltage being generated to be given to an output pin. Please note, if the [REF_A_isRefGenBusy\(\)](#) returns REF_A_BUSY, this function will have no effect.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:**baseAddress** is the base address of the REF module.Modified bits are **REFOUT** of **REFCTL0** register.**Returns:**

None

23.2.2.3 void REF_A_disableTempSensor (uint16_t baseAddress)

Disables the internal temperature sensor to save power consumption.

This function is used to turn off the internal temperature sensor to save on power consumption. The temperature sensor is enabled by default. Please note, that giving ADC12 module control over the REF module, the state of the temperature sensor is dependent on the controls of the ADC12 module. Please note, if the [REF_A_isRefGenBusy\(\)](#) returns REF_A_BUSY, this function will have no effect.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the REF module.

Modified bits are **REFTCOFF** of **REFCTL0** register.

Returns:

None

23.2.2.4 void REF_A_enableReferenceVoltage (uint16_t baseAddress)

Enables the reference voltage to be used by peripherals.

This function is used to enable the generated reference voltage to be used other peripherals or by an output pin, if enabled. Please note, that giving ADC12 module control over the REF module, the state of the reference voltage is dependent on the controls of the ADC12 module. Please note, if the [REF_A_isRefGenBusy\(\)](#) returns REF_A_BUSY, this function will have no effect.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the REF module.

Modified bits are **REFON** of **REFCTL0** register.

Returns:

None

23.2.2.5 void REF_A_enableReferenceVoltageOutput (uint16_t baseAddress)

Outputs the reference voltage to an output pin.

This function is used to output the reference voltage being generated to an output pin. Please note, the output pin is device specific. Please note, that giving ADC12 module control over the REF module, the state of the reference voltage as an output to a pin is dependent on the controls of the ADC12 module. Please note, if the [REF_A_isRefGenBusy\(\)](#) returns REF_A_BUSY, this function will have no effect.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the REF module.

Modified bits are **REFOUT** of **REFCTL0** register.

Returns:

None

23.2.2.6 void REF_A_enableTempSensor (uint16_t baseAddress)

Enables the internal temperature sensor.

This function is used to turn on the internal temperature sensor to use by other peripherals. The temperature sensor is enabled by default. Please note, if the [REF_A_isRefGenBusy\(\)](#) returns REF_A_BUSY, this function will have no effect.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the REF module.

Modified bits are **REFTCOFF** of **REFCTL0** register.

Returns:

None

23.2.2.7 uint16_t REF_A_getBandgapMode (uint16_t baseAddress)

Returns the bandgap mode of the REF module.

This function is used to return the bandgap mode of the REF module, requested by the peripherals using the bandgap. If a peripheral requests static mode, then the bandgap mode will be static for all modules, whereas if all of the peripherals using the bandgap request sample mode, then that will be the mode returned. Sample mode allows the bandgap to be active only when necessary to save on power consumption, static mode requires the bandgap to be active until no peripherals are using it anymore.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the REF module.

Returns:

One of the following:

- **REF_A_STATICMODE** if the bandgap is operating in static mode
- **REF_A_SAMPLEMODE** if the bandgap is operating in sample mode indicating the bandgap mode of the module

23.2.2.8 bool REF_A_isBandgapActive (uint16_t baseAddress)

Returns the active status of the bandgap in the REF module.

This function is used to return the active status of the bandgap in the REF module. If the bandgap is in use by a peripheral, then the status will be seen as active.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	18
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the REF module.

Returns:

One of the following:

- **REF_A_ACTIVE** if active
- **REF_A_INACTIVE** if not active indicating the bandgap active status of the module

23.2.2.9 bool REF_A_isBufferedBandgapVoltageReady (uint16_t baseAddress)

Returns the busy status of the reference generator in the REF module.

This function is used to return the busy status of the buffered bandgap voltage in the REF module. If the ref. generator is on and ready to use, then the status will be seen as active.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	18
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the REF module.

Returns:

One of the following:

- **REF_A_NOTREADY** if NOT ready to be used
- **REF_A_READY** if ready to be used
indicating the the busy status of the reference generator in the module

23.2.2.10 bool REF_A_isRefGenActive (uint16_t baseAddress)

Returns the active status of the reference generator in the REF module.

This function is used to return the active status of the reference generator in the REF module. If the ref. generator is on and ready to use, then the status will be seen as active.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	18
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the REF module.

Returns:

One of the following:

- **REF_A_ACTIVE** if active
- **REF_A_INACTIVE** if not active
indicating the reference generator active status of the module

23.2.2.11 uint16_t REF_A_isRefGenBusy (uint16_t baseAddress)

Returns the busy status of the reference generator in the REF module.

This function is used to return the busy status of the reference generator in the REF module. If the ref. generator is in use by a peripheral, then the status will be seen as busy.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	18
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the REF module.

Returns:

One of the following:

- **REF_A_NOTBUSY** if the reference generator is not being used
- **REF_A_BUSY** if the reference generator is being used, disallowing changes to be made to the REF module controls indicating the reference generator busy status of the module

23.2.2.12 bool REF_A_isVariableReferenceVoltageOutputReady (uint16_t baseAddress)

Returns the busy status of the variable reference voltage in the REF module.

This function is used to return the busy status of the variable reference voltage in the REF module. If the ref. generator is on and ready to use, then the status will be seen as active.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	18
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the REF module.

Returns:

One of the following:

- **REF_A_NOTREADY** if NOT ready to be used
- **REF_A_READY** if ready to be used indicating the the busy status of the variable reference voltage in the module

23.2.2.13 void REF_A_setBufferedBandgapVoltageOneTimeTrigger (uint16_t baseAddress)

Enables the one-time trigger of the buffered bandgap voltage.

Triggers the one-time generation of the buffered bandgap voltage. Once the buffered bandgap voltage request is set, this bit is cleared by hardware

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the REF module.

Modified bits are **REFBGOT** of **REFCTL0** register.

Returns:

None

23.2.2.14 void REF_A_setReferenceVoltage (uint16_t baseAddress, uint8_t referenceVoltageSelect)

Sets the reference voltage for the voltage generator.

This function sets the reference voltage generated by the voltage generator to be used by other peripherals. This reference voltage will only be valid while the REF module is in control. Please note, if the [REF_A_isRefGenBusy\(\)](#) returns REF_A_BUSY, this function will have no effect.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the REF module.

referenceVoltageSelect is the desired voltage to generate for a reference voltage. Valid values are:

- **REF_A_VREF1_2V** [Default]
 - **REF_A_VREF2_0V**
 - **REF_A_VREF2_5V**
- Modified bits are **REFVSEL** of **REFCTL0** register.

Returns:

None

23.2.2.15 void REF_A_setReferenceVoltageOneTimeTrigger (uint16_t baseAddress)

Enables the one-time trigger of the reference voltage.

Triggers the one-time generation of the variable reference voltage. Once the reference voltage request is set, this bit is cleared by hardware

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the REF module.

Modified bits are **REFGENOT** of **REFCTL0** register.

Returns:

None

23.3 Programming Example

The following example shows how to initialize and use the REF_A API with the ADC12 module to use the internal 2.5V reference and perform a single conversion on channel A0. The conversion results are stored in ADC12BMEM0. Test by applying a voltage to channel A0, then setting and running to a break point at the "__no_operation()" instruction. To view the conversion results, open an ADC12B register window in debugger and view the contents of ADC12BMEM0.

```
//If ref generator busy, WAIT
while (REF_A_isRefGenBusy(REF_A_BASE)) ;
//Select internal ref = 2.5V
REF_A_setReferenceVoltage(REF_A_BASE,
    REF_A_VREF2_5V);
//Internal Reference ON
REF_A_enableReferenceVoltage(REF_A_BASE);

//Delay (~75us) for Ref to settle
__delay_cycles(75);

//Initialize the ADC12 Module
/*
Base address of ADC12 Module
Use internal ADC12 bit as sample/hold signal to start conversion
USE MODOSC 5MHZ Digital Oscillator as clock source
Use default clock divider/pre-divider of 1
Map to internal channel 0
*/
ADC12_B_init(ADC12_B_BASE,
    ADC12_B_SAMPLEHOLDSOURCE_SC,
    ADC12_B_CLOCKSOURCE_ADC12OSC,
    ADC12_B_CLOCKDIVIDER_1,
    ADC12_B_CLOCKPREDIVIDER__1,
    ADC12_B_MAPINTCH0);
```

24 Real-Time Clock (RTC_B)

Introduction	268
API Functions	268
Programming Example	280

24.1 Introduction

The Real Time Clock (RTC_B) API provides a set of functions for using the MSP430Ware RTC_B modules. Functions are provided to calibrate the clock, initialize the RTC modules in calendar mode, and setup conditions for, and enable, interrupts for the RTC modules. If an RTC_B module is used, then prescale counters are also initialized.

The RTC_B module provides the ability to keep track of the current time and date in calendar mode.

The RTC_B module generates multiple interrupts. There are 2 interrupts that can be defined in calendar mode, and 1 interrupt for user-configured event, as well as an interrupt for each prescaler.

This driver is contained in `rtc_b.c`, with `rtc_b.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	1096
TI Compiler 4.2.1	Size	570
TI Compiler 4.2.1	Speed	568
IAR 5.51.6	None	840
IAR 5.51.6	Size	674
IAR 5.51.6	Speed	708
MSPGCC 4.8.0	None	1822
MSPGCC 4.8.0	Size	648
MSPGCC 4.8.0	Speed	652

24.2 API Functions

Functions

- void [RTC_B_calendarInit](#) (uint16_t baseAddress, Calendar CalendarTime, uint16_t formatSelect)
- void [RTC_B_clearInterrupt](#) (uint16_t baseAddress, uint8_t interruptFlagMask)
- void [RTC_B_configureCalendarAlarm](#) (uint16_t baseAddress, RTC_B_configureCalendarAlarmParam *param)
- uint16_t [RTC_B_convertBCDToBinary](#) (uint16_t baseAddress, uint16_t valueToConvert)
- uint16_t [RTC_B_convertBinaryToBCD](#) (uint16_t baseAddress, uint16_t valueToConvert)
- void [RTC_B_definePrescaleEvent](#) (uint16_t baseAddress, uint8_t prescaleSelect, uint8_t prescaleEventDivider)
- void [RTC_B_disableInterrupt](#) (uint16_t baseAddress, uint8_t interruptMask)
- void [RTC_B_enableInterrupt](#) (uint16_t baseAddress, uint8_t interruptMask)
- Calendar [RTC_B_getCalendarTime](#) (uint16_t baseAddress)
- uint8_t [RTC_B_getInterruptStatus](#) (uint16_t baseAddress, uint8_t interruptFlagMask)
- uint8_t [RTC_B_getPrescaleValue](#) (uint16_t baseAddress, uint8_t prescaleSelect)
- void [RTC_B_holdClock](#) (uint16_t baseAddress)
- void [RTC_B_initCalendar](#) (uint16_t baseAddress, Calendar *CalendarTime, uint16_t formatSelect)

- void [RTC_B_setCalendarAlarm](#) (uint16_t baseAddress, uint8_t minutesAlarm, uint8_t hoursAlarm, uint8_t dayOfWeekAlarm, uint8_t dayOfMonthAlarm)
- void [RTC_B_setCalendarEvent](#) (uint16_t baseAddress, uint16_t eventSelect)
- void [RTC_B_setCalibrationData](#) (uint16_t baseAddress, uint8_t offsetDirection, uint8_t offsetValue)
- void [RTC_B_setCalibrationFrequency](#) (uint16_t baseAddress, uint16_t frequencySelect)
- void [RTC_B_setPrescaleValue](#) (uint16_t baseAddress, uint8_t prescaleSelect, uint8_t prescaleCounterValue)
- void [RTC_B_startClock](#) (uint16_t baseAddress)

24.2.1 Detailed Description

The RTC_B API is broken into 5 groups of functions: clock settings, calender mode, prescale counter, interrupt condition setup/enable functions and data conversion.

The RTC_B clock settings are handled by

- [RTC_B_startClock\(\)](#)
- [RTC_B_holdClock\(\)](#)
- [RTC_B_setCalibrationFrequency\(\)](#)
- [RTC_B_setCalibrationData\(\)](#)

The RTC_B calender mode is initialized and handled by

- [RTC_B_initCalendar\(\)](#)
- [RTC_B_configureCalendarAlarm\(\)](#)
- [RTC_B_getCalendarTime\(\)](#)

The RTC_B prescale counter is handled by

- [RTC_B_getPrescaleValue\(\)](#)
- [RTC_B_setPrescaleValue\(\)](#)

The RTC_B interrupts are handled by

- [RTC_B_definePrescaleEvent\(\)](#)
- [RTC_B_setCalendarEvent\(\)](#)
- [RTC_B_enableInterrupt\(\)](#)
- [RTC_B_disableInterrupt\(\)](#)
- [RTC_B_getInterruptStatus\(\)](#)
- [RTC_B_clearInterrupt\(\)](#)

The RTC_B conversions are handled by

- [RTC_B_convertBCDToBinary\(\)](#)
- [RTC_B_convertBinaryToBCD\(\)](#)

24.2.2 Function Documentation

24.2.2.1 void [RTC_B_calendarInit](#) (uint16_t *baseAddress*, Calendar *CalendarTime*, uint16_t *formatSelect*)

Deprecated - Initializes the settings to operate the RTC in calendar mode.

This function initializes the Calendar mode of the RTC module.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	126
TI Compiler 4.2.1	Size	84
TI Compiler 4.2.1	Speed	84
IAR 5.51.6	None	108
IAR 5.51.6	Size	90
IAR 5.51.6	Speed	104
MSPGCC 4.8.0	None	218
MSPGCC 4.8.0	Size	126
MSPGCC 4.8.0	Speed	126

Parameters:

baseAddress is the base address of the RTC_B module.

CalendarTime is the structure containing the values for the Calendar to be initialized to. Valid values should be of type Calendar and should contain the following members and corresponding values: **Seconds** between 0-59 **Minutes** between 0-59 **Hours** between 0-24 **DayOfWeek** between 0-6 **DayOfMonth** between 0-31 **Year** between 0-4095 NOTE: Values beyond the ones specified may result in erratic behavior.

formatSelect is the format for the Calendar registers to use. Valid values are:

- **RTC_B_FORMAT_BINARY** [Default]
- **RTC_B_FORMAT_BCD**
Modified bits are **RTCB**CD of **RTCCTL1** register.

Returns:

None

24.2.2.2 void RTC_B_clearInterrupt (uint16_t baseAddress, uint8_t interruptFlagMask)

Clears selected RTC interrupt flags.

This function clears the RTC interrupt flag is cleared, so that it no longer asserts.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	70
TI Compiler 4.2.1	Size	34
TI Compiler 4.2.1	Speed	34
IAR 5.51.6	None	44
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	144
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	36

Parameters:

baseAddress is the base address of the RTC_B module.

interruptFlagMask is a bit mask of the interrupt flags to be cleared. Mask value is the logical OR of any of the following:

- **RTC_B_TIME_EVENT_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC_B_CLOCK_ALARM_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC_B_CLOCK_READ_READY_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC_B_PRESCALE_TIMER0_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC_B_PRESCALE_TIMER1_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC_B_OSCILLATOR_FAULT_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

Returns:

None

24.2.2.3 void RTC_B_configureCalendarAlarm (uint16_t *baseAddress*, RTC_B_configureCalendarAlarmParam * *param*)

Sets and Enables the desired Calendar Alarm settings.

This function sets a Calendar interrupt condition to assert the RTCAIFG interrupt flag. The condition is a logical and of all of the parameters. For example if the minutes and hours alarm is set, then the interrupt will only assert when the minutes AND the hours change to the specified setting. Use the RTC_B_ALARM_OFF for any alarm settings that should not be apart of the alarm condition.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	78
TI Compiler 4.2.1	Size	48
TI Compiler 4.2.1	Speed	48
IAR 5.51.6	None	72
IAR 5.51.6	Size	68
IAR 5.51.6	Speed	68
MSPGCC 4.8.0	None	116
MSPGCC 4.8.0	Size	48
MSPGCC 4.8.0	Speed	48

Parameters:

baseAddress is the base address of the RTC_B module.

param is the pointer to struct for calendar alarm configuration.

Returns:

None

24.2.2.4 uint16_t RTC_B_convertBCDToBinary (uint16_t *baseAddress*, uint16_t *valueToConvert*)

Convert the given BCD value to binary format.

This function converts BCD values to binary format. This API uses the hardware registers to perform the conversion rather than a software method.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	18
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the RTC_B module.

valueToConvert is the raw value in BCD format to convert to Binary.

Modified bits are **BCD2BIN** of **BCD2BIN** register.

Returns:

The binary version of the input parameter

24.2.2.5 uint16_t RTC_B_convertBinaryToBCD (uint16_t *baseAddress*, uint16_t *valueToConvert*)

Convert the given binary value to BCD format.

This function converts binary values to BCD format. This API uses the hardware registers to perform the conversion rather than a software method.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	26
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	18
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the RTC_B module.

valueToConvert is the raw value in Binary format to convert to BCD.
Modified bits are **BIN2BCD** of **BIN2BCD** register.

Returns:

The BCD version of the valueToConvert parameter

24.2.2.6 void RTC_B_definePrescaleEvent (uint16_t *baseAddress*, uint8_t *prescaleSelect*, uint8_t *prescaleEventDivider*)

Sets up an interrupt condition for the selected Prescaler.

This function sets the condition for an interrupt to assert based on the individual prescalers.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	46
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	28
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Parameters:

baseAddress is the base address of the RTC_B module.

prescaleSelect is the prescaler to define an interrupt for. Valid values are:

- **RTC_B_PRESCALE_0**
- **RTC_B_PRESCALE_1**

prescaleEventDivider is a divider to specify when an interrupt can occur based on the clock source of the selected prescaler. (Does not affect timer of the selected prescaler). Valid values are:

- **RTC_B_PSEVENTDIVIDER_2** [Default]

- **RTC_B_PSEVENTDIVIDER_4**
 - **RTC_B_PSEVENTDIVIDER_8**
 - **RTC_B_PSEVENTDIVIDER_16**
 - **RTC_B_PSEVENTDIVIDER_32**
 - **RTC_B_PSEVENTDIVIDER_64**
 - **RTC_B_PSEVENTDIVIDER_128**
 - **RTC_B_PSEVENTDIVIDER_256**
- Modified bits are **RTxIP** of **RTCPSxCTL** register.

Returns:
None

24.2.2.7 void RTC_B_disableInterrupt (uint16_t *baseAddress*, uint8_t *interruptMask*)

Disables selected RTC interrupt sources.

This function disables the selected RTC interrupt source. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	66
TI Compiler 4.2.1	Size	36
TI Compiler 4.2.1	Speed	36
IAR 5.51.6	None	44
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	148
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	36

Parameters:

baseAddress is the base address of the RTC_B module.

interruptMask is a bit mask of the interrupts to disable. Mask value is the logical OR of any of the following:

- **RTC_B_TIME_EVENT_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC_B_CLOCK_ALARM_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC_B_CLOCK_READ_READY_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC_B_PRESCALE_TIMER0_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC_B_PRESCALE_TIMER1_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC_B_OSCILLATOR_FAULT_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

Returns:
None

24.2.2.8 void RTC_B_enableInterrupt (uint16_t *baseAddress*, uint8_t *interruptMask*)

Enables selected RTC interrupt sources.

This function enables the selected RTC interrupt source. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. Does not clear interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	66
TI Compiler 4.2.1	Size	32
TI Compiler 4.2.1	Speed	32
IAR 5.51.6	None	44
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	136
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	36

Parameters:

baseAddress is the base address of the RTC_B module.

interruptMask is a bit mask of the interrupts to enable. Mask value is the logical OR of any of the following:

- **RTC_B_TIME_EVENT_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC_B_CLOCK_ALARM_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC_B_CLOCK_READ_READY_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC_B_PRESCALE_TIMER0_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC_B_PRESCALE_TIMER1_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC_B_OSCILLATOR_FAULT_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

Returns:

None

24.2.2.9 Calendar RTC_B_getCalendarTime (uint16_t baseAddress)

Returns the Calendar Time stored in the Calendar registers of the RTC.

This function returns the current Calendar time in the form of a Calendar structure. The `RTCRDY` polling is used in this function to prevent reading invalid time.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	80
TI Compiler 4.2.1	Size	68
TI Compiler 4.2.1	Speed	68
IAR 5.51.6	None	104
IAR 5.51.6	Size	104
IAR 5.51.6	Speed	104
MSPGCC 4.8.0	None	148
MSPGCC 4.8.0	Size	68
MSPGCC 4.8.0	Speed	68

Parameters:

baseAddress is the base address of the RTC_B module.

Returns:

A Calendar structure containing the current time.

24.2.2.10 uint8_t RTC_B_getInterruptStatus (uint16_t baseAddress, uint8_t interruptFlagMask)

Returns the status of the selected interrupts flags.

This function returns the status of the interrupt flag for the selected channel.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	86
TI Compiler 4.2.1	Size	40
TI Compiler 4.2.1	Speed	40
IAR 5.51.6	None	54
IAR 5.51.6	Size	44
IAR 5.51.6	Speed	44
MSPGCC 4.8.0	None	142
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	54

Parameters:

baseAddress is the base address of the RTC_B module.

interruptFlagMask is a bit mask of the interrupt flags to return the status of. Mask value is the logical OR of any of the following:

- **RTC_B_TIME_EVENT_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC_B_CLOCK_ALARM_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC_B_CLOCK_READ_READY_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC_B_PRESCALE_TIMER0_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC_B_PRESCALE_TIMER1_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC_B_OSCILLATOR_FAULT_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

Returns:

Logical OR of any of the following:

- **RTC_B_TIME_EVENT_INTERRUPT** asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
 - **RTC_B_CLOCK_ALARM_INTERRUPT** asserts when alarm condition in Calendar mode is met.
 - **RTC_B_CLOCK_READ_READY_INTERRUPT** asserts when Calendar registers are settled.
 - **RTC_B_PRESCALE_TIMER0_INTERRUPT** asserts when Prescaler 0 event condition is met.
 - **RTC_B_PRESCALE_TIMER1_INTERRUPT** asserts when Prescaler 1 event condition is met.
 - **RTC_B_OSCILLATOR_FAULT_INTERRUPT** asserts if there is a problem with the 32kHz oscillator, while the RTC is running.
- indicating the status of the masked interrupts

24.2.2.11 `uint8_t RTC_B_getPrescaleValue (uint16_t baseAddress, uint8_t prescaleSelect)`

Returns the selected prescaler value.

This function returns the value of the selected prescale counter register. Note that the counter value should be held by calling `RTC_B_holdClock()` before calling this API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	40
TI Compiler 4.2.1	Size	24
TI Compiler 4.2.1	Speed	24
IAR 5.51.6	None	30
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	28
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	28

Parameters:

baseAddress is the base address of the RTC_B module.

prescaleSelect is the prescaler to obtain the value of. Valid values are:

■ RTC_B_PRESCALE_0

■ RTC_B_PRESCALE_1

Returns:

The value of the specified prescaler count register

24.2.2.12 void RTC_B_holdClock (uint16_t baseAddress)

Holds the RTC.

This function sets the RTC main hold bit to disable RTC functionality.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the RTC_B module.

Returns:

None

24.2.2.13 void RTC_B_initCalendar (uint16_t baseAddress, Calendar * CalendarTime, uint16_t formatSelect)

Initializes the settings to operate the RTC in calendar mode.

This function initializes the Calendar mode of the RTC module.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	130
TI Compiler 4.2.1	Size	58
TI Compiler 4.2.1	Speed	58
IAR 5.51.6	None	106
IAR 5.51.6	Size	88
IAR 5.51.6	Speed	102
MSPGCC 4.8.0	None	220
MSPGCC 4.8.0	Size	58
MSPGCC 4.8.0	Speed	58

Parameters:

baseAddress is the base address of the RTC_B module.

CalendarTime is the pointer to the structure containing the values for the Calendar to be initialized to. Valid values should be of type pointer to Calendar and should contain the following members and corresponding values: **Seconds** between 0-59 **Minutes** between 0-59 **Hours** between 0-24 **DayOfWeek** between 0-6 **DayOfMonth** between 0-31 **Year** between 0-4095 NOTE: Values beyond the ones specified may result in erratic behavior.

formatSelect is the format for the Calendar registers to use. Valid values are:

- **RTC_B_FORMAT_BINARY** [Default]
- **RTC_B_FORMAT_BCD**
Modified bits are **RTCBCD** of **RTCCTL1** register.

Returns:
None

24.2.2.14 void RTC_B_setCalendarAlarm (uint16_t *baseAddress*, uint8_t *minutesAlarm*, uint8_t *hoursAlarm*, uint8_t *dayOfWeekAlarm*, uint8_t *dayOfMonthAlarm*)

DEPRECATED - Sets and Enables the desired Calendar Alarm settings.

This function sets a Calendar interrupt condition to assert the RTCAIFG interrupt flag. The condition is a logical and of all of the parameters. For example if the minutes and hours alarm is set, then the interrupt will only assert when the minutes AND the hours change to the specified setting. Use the **RTC_B_ALARM_OFF** for any alarm settings that should not be apart of the alarm condition.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	78
TI Compiler 4.2.1	Size	32
TI Compiler 4.2.1	Speed	32
IAR 5.51.6	None	64
IAR 5.51.6	Size	54
IAR 5.51.6	Speed	60
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	38

Parameters:

baseAddress is the base address of the RTC_B module.

minutesAlarm is the alarm condition for the minutes. Valid values are:

- **RTC_B_ALARMCONDITION_OFF** [Default]
- An integer between 0-59

hoursAlarm is the alarm condition for the hours. Valid values are:

- **RTC_B_ALARMCONDITION_OFF** [Default]
- An integer between 0-24

dayOfWeekAlarm is the alarm condition for the day of week. Valid values are:

- **RTC_B_ALARMCONDITION_OFF** [Default]
- An integer between 0-6

dayOfMonthAlarm is the alarm condition for the day of the month. Valid values are:

- **RTC_B_ALARMCONDITION_OFF** [Default]
- An integer between 0-31

Returns:
None

24.2.2.15 void RTC_B_setCalendarEvent (uint16_t baseAddress, uint16_t eventSelect)

Sets a single specified Calendar interrupt condition.

This function sets a specified event to assert the RTCTEVIFG interrupt. This interrupt is independent from the Calendar alarm interrupt.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the RTC_B module.

eventSelect is the condition selected. Valid values are:

- **RTC_B_CALENDAREVENT_MINUTECHANGE** - assert interrupt on every minute
 - **RTC_B_CALENDAREVENT_HOURLCHANGE** - assert interrupt on every hour
 - **RTC_B_CALENDAREVENT_NOON** - assert interrupt when hour is 12
 - **RTC_B_CALENDAREVENT_MIDNIGHT** - assert interrupt when hour is 0
- Modified bits are **RTCCTEV** of **RTCCTL** register.

Returns:

None

24.2.2.16 void RTC_B_setCalibrationData (uint16_t baseAddress, uint8_t offsetDirection, uint8_t offsetValue)

Sets the specified calibration for the RTC.

This function sets the calibration offset to make the RTC as accurate as possible. The offsetDirection can be either +4-ppm or -2-ppm, and the offsetValue should be from 1-63 and is multiplied by the direction setting (i.e. +4-ppm * 8 (offsetValue) = +32-ppm). Please note, when measuring the frequency after setting the calibration, you will only see a change on the 1Hz frequency.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	18
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the RTC_B module.

offsetDirection is the direction that the calibration offset will go. Valid values are:

- **RTC_B_CALIBRATION_DOWN2PPM** - calibrate at steps of -2
- **RTC_B_CALIBRATION_UP4PPM** - calibrate at steps of +4

Modified bits are **RTCCALS** of **RTCCTL2** register.

offsetValue is the value that the offset will be a factor of; a valid value is any integer from 1-63.
Modified bits are **RTCCAL** of **RTCCTL2** register.

Returns:
None

24.2.2.17 void RTC_B_setCalibrationFrequency (uint16_t *baseAddress*, uint16_t *frequencySelect*)

Allows and Sets the frequency output to RTCCLK pin for calibration measurement.

This function sets a frequency to measure at the RTCCLK output pin. After testing the set frequency, the calibration could be set accordingly.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the RTC_B module.

frequencySelect is the frequency output to RTCCLK. Valid values are:

- **RTC_B_CALIBRATIONFREQ_OFF** [Default] - turn off calibration output
 - **RTC_B_CALIBRATIONFREQ_512HZ** - output signal at 512Hz for calibration
 - **RTC_B_CALIBRATIONFREQ_256HZ** - output signal at 256Hz for calibration
 - **RTC_B_CALIBRATIONFREQ_1HZ** - output signal at 1Hz for calibration
- Modified bits are **RTCCALF** of **RTCCTL3** register.

Returns:
None

24.2.2.18 void RTC_B_setPrescaleValue (uint16_t *baseAddress*, uint8_t *prescaleSelect*, uint8_t *prescaleCounterValue*)

Sets the selected prescaler value.

This function sets the prescale counter value. Before setting the prescale counter, it should be held by calling [RTC_B_holdClock\(\)](#).

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	44
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	32
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

Parameters:

baseAddress is the base address of the RTC_B module.

prescaleSelect is the prescaler to set the value for. Valid values are:

- RTC_B_PRESCALE_0
- RTC_B_PRESCALE_1

prescaleCounterValue is the specified value to set the prescaler to. Valid values are any integer between 0-255
Modified bits are **RTxPS** of **RTxPS** register.

Returns:

None

24.2.2.19 void RTC_B_startClock (uint16_t baseAddress)

Starts the RTC.

This function clears the RTC main hold bit to allow the RTC to function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the RTC_B module.

Returns:

None

24.3 Programming Example

The following example shows how to initialize and use the RTC API to setup Calendar Mode with the current time and various interrupts.

```
//Initialize calendar struct
Calendar currentTime;
currentTime.Seconds    = 0x00;
```

```

    currentTime.Minutes    = 0x26;
    currentTime.Hours      = 0x13;
    currentTime.DayOfWeek  = 0x03;
    currentTime.DayOfMonth = 0x20;
    currentTime.Month      = 0x07;
    currentTime.Year       = 0x2011;

    //Initialize alarm struct
    RTC_B_configureCalendarAlarmParam alarmParam;
    alarmParam.minutesAlarm = 0x00;
    alarmParam.hoursAlarm = 0x17;
    alarmParam.dayOfWeekAlarm = RTC_B_ALARMCONDITION_OFF;
    alarmParam.dayOfMonthAlarm = 0x05;

    //Initialize Calendar Mode of RTC_B
    /*
    Base Address of the RTC_B
    Pass in current time, initialized above
    Use BCD as Calendar Register Format
    */
    RTC_B_initCalendar(RTC_B_BASE,
        &currentTime,
        RTC_B_FORMAT_BCD);

    //Setup Calendar Alarm for 5:00pm on the 5th day of the month.
    //Note: Does not specify day of the week.
    RTC_B_setCalendarAlarm(RTC_B_BASE, &alarmParam);

    //Specify an interrupt to assert every minute
    RTC_B_setCalendarEvent(RTC_B_BASE,
        RTC_B_CALENDAREVENT_MINUTECHANGE);

    //Enable interrupt for RTC_B Ready Status, which asserts when the RTC_B
    //Calendar registers are ready to read.
    //Also, enable interrupts for the Calendar alarm and Calendar event.
    RTC_B_enableInterrupt(RTC_B_BASE,
        RTC_B_CLOCK_READ_READY_INTERRUPT +
        RTC_B_TIME_EVENT_INTERRUPT +
        RTC_B_CLOCK_ALARM_INTERRUPT);

    //Start RTC_B Clock
    RTC_B_startClock(RTC_B_BASE);

    //Enter LPM3 mode with interrupts enabled
    __bis_SR_register(LPM3_bits + GIE);
    __no_operation();

```

25 Real-Time Clock (RTC_C)

Introduction	282
API Functions	282
Programming Example	292

25.1 Introduction

The Real Time Clock (RTC_C) API provides a set of functions for using the MSP430Ware RTC_C modules. Functions are provided to calibrate the clock, initialize the RTC_C modules in Calendar mode, and setup conditions for, and enable, interrupts for the RTC_C modules.

The RTC_C module provides the ability to keep track of the current time and date in calendar mode. The counter mode (device-dependent) provides a 32-bit counter.

The RTC_C module generates multiple interrupts. There are 2 interrupts that can be defined in calendar mode, and 1 interrupt in counter mode for counter overflow, as well as an interrupt for each prescaler.

If the device header file defines the baseaddress as RTC_C_BASE, pass in RTC_C_BASE as the baseaddress parameter. If the device header file defines the baseaddress as RTC_CE_BASE, pass in RTC_CE_BASE as the baseaddress parameter.

This driver is contained in `rtc_c.c`, with `rtc_c.h` containing the API definitions for use by applications.

25.2 API Functions

Functions

- void [RTC_C_calendarInit](#) (uint16_t baseAddress, Calendar CalendarTime, uint16_t formatSelect)
- void [RTC_C_clearInterrupt](#) (uint16_t baseAddress, uint8_t interruptFlagMask)
- void [RTC_C_configureCalendarAlarm](#) (uint16_t baseAddress, RTC_C_configureCalendarAlarmParam *param)
- uint16_t [RTC_C_convertBCDToBinary](#) (uint16_t baseAddress, uint16_t valueToConvert)
- uint16_t [RTC_C_convertBinaryToBCD](#) (uint16_t baseAddress, uint16_t valueToConvert)
- void [RTC_C_definePrescaleEvent](#) (uint16_t baseAddress, uint8_t prescaleSelect, uint8_t prescaleEventDivider)
- void [RTC_C_disableInterrupt](#) (uint16_t baseAddress, uint8_t interruptMask)
- void [RTC_C_enableInterrupt](#) (uint16_t baseAddress, uint8_t interruptMask)
- Calendar [RTC_C_getCalendarTime](#) (uint16_t baseAddress)
- uint32_t [RTC_C_getCounterValue](#) (uint16_t baseAddress)
- uint8_t [RTC_C_getInterruptStatus](#) (uint16_t baseAddress, uint8_t interruptFlagMask)
- uint8_t [RTC_C_getPrescaleValue](#) (uint16_t baseAddress, uint8_t prescaleSelect)
- void [RTC_C_holdClock](#) (uint16_t baseAddress)
- void [RTC_C_holdCounterPrescale](#) (uint16_t baseAddress, uint8_t prescaleSelect)
- void [RTC_C_initCalendar](#) (uint16_t baseAddress, Calendar *CalendarTime, uint16_t formatSelect)
- void [RTC_C_initCounter](#) (uint16_t baseAddress, uint16_t clockSelect, uint16_t counterSizeSelect)
- void [RTC_C_initCounterPrescale](#) (uint16_t baseAddress, uint8_t prescaleSelect, uint16_t prescaleClockSelect, uint16_t prescaleDivider)
- void [RTC_C_setCalendarAlarm](#) (uint16_t baseAddress, uint8_t minutesAlarm, uint8_t hoursAlarm, uint8_t dayOfWeekAlarm, uint8_t dayOfMonthAlarm)
- void [RTC_C_setCalendarEvent](#) (uint16_t baseAddress, uint16_t eventSelect)
- void [RTC_C_setCalibrationData](#) (uint16_t baseAddress, uint8_t offsetDirection, uint8_t offsetValue)
- void [RTC_C_setCalibrationFrequency](#) (uint16_t baseAddress, uint16_t frequencySelect)
- void [RTC_C_setCounterValue](#) (uint16_t baseAddress, uint32_t counterValue)
- void [RTC_C_setPrescaleValue](#) (uint16_t baseAddress, uint8_t prescaleSelect, uint8_t prescaleCounterValue)
- bool [RTC_C_setTemperatureCompensation](#) (uint16_t baseAddress, uint16_t offsetDirection, uint8_t offsetValue)
- void [RTC_C_startClock](#) (uint16_t baseAddress)
- void [RTC_C_startCounterPrescale](#) (uint16_t baseAddress, uint8_t prescaleSelect)

25.2.1 Detailed Description

The RTC_C API is broken into 6 groups of functions: clock settings, calender mode, counter mode, prescale counter, interrupt condition setup/enable functions and data conversion.

The RTC_C clock settings are handled by

- [RTC_C_startClock\(\)](#)
- [RTC_C_holdClock\(\)](#)
- [RTC_C_setCalibrationFrequency\(\)](#)
- [RTC_C_setCalibrationData\(\)](#)
- [RTC_C_setTemperatureCompensation\(\)](#)

The RTC_C calender mode is initialized and setup by

- [RTC_C_initCalendar\(\)](#)
- [RTC_C_getCalendarTime\(\)](#)

The RTC_C counter mode is initialized and handled by

- [RTC_C_initCounter\(\)](#)
- [RTC_C_setCounterValue\(\)](#)
- [RTC_C_getCounterValue\(\)](#)
- [RTC_C_initCounterPrescale\(\)](#)
- [RTC_C_holdCounterPrescale\(\)](#)
- [RTC_C_startCounterPrescale\(\)](#)

The RTC_C prescale counter is handled by

- [RTC_C_getPrescaleValue\(\)](#)
- [RTC_C_setPrescaleValue\(\)](#)

The RTC_C interrupts are handled by

- [RTC_C_configureCalendarAlarm\(\)](#)
- [RTC_C_setCalendarEvent\(\)](#)
- [RTC_C_definePrescaleEvent\(\)](#)
- [RTC_C_enableInterrupt\(\)](#)
- [RTC_C_disableInterrupt\(\)](#)
- [RTC_C_getInterruptStatus\(\)](#)
- [RTC_C_clearInterrupt\(\)](#)

The RTC_C data conversion is handled by

- [RTC_C_convertBCDToBinary\(\)](#)
- [RTC_C_convertBinaryToBCD\(\)](#)

25.2.2 Function Documentation

25.2.2.1 void RTC_C_calendarInit (uint16_t *baseAddress*, Calendar *CalendarTime*, uint16_t *formatSelect*)

Deprecated - Initializes the settings to operate the RTC in calendar mode.

This function initializes the Calendar mode of the RTC module.

Parameters:

baseAddress is the base address of the RTC_C module.

CalendarTime is the structure containing the values for the Calendar to be initialized to. Valid values should be of type Calendar and should contain the following members and corresponding values: **Seconds** between 0-59 **Minutes** between 0-59 **Hours** between 0-24 **DayOfWeek** between 0-6 **DayOfMonth** between 0-31 **Year** between 0-4095 NOTE: Values beyond the ones specified may result in erratic behavior.

formatSelect is the format for the Calendar registers to use. Valid values are:

- **RTC_C_FORMAT_BINARY** [Default]
- **RTC_C_FORMAT_BCD**
Modified bits are **RTCB**CD of **RTCCTL1** register.

Returns:

None

25.2.2.2 void RTC_C_clearInterrupt (uint16_t baseAddress, uint8_t interruptFlagMask)

Clears selected RTC interrupt flags.

This function clears the RTC interrupt flag is cleared, so that it no longer asserts.

Parameters:

baseAddress is the base address of the RTC_C module.

interruptFlagMask is a bit mask of the interrupt flags to be cleared. Mask value is the logical OR of any of the following:

- **RTC_C_TIME_EVENT_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by defineCalendarEvent() is met.
- **RTC_C_CLOCK_ALARM_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC_C_CLOCK_READ_READY_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC_C_PRESCALE_TIMER0_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC_C_PRESCALE_TIMER1_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC_C_OSCILLATOR_FAULT_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

Returns:

None

25.2.2.3 void RTC_C_configureCalendarAlarm (uint16_t baseAddress, RTC_C_configureCalendarAlarmParam * param)

Sets and Enables the desired Calendar Alarm settings.

This function sets a Calendar interrupt condition to assert the RTCAIFG interrupt flag. The condition is a logical and of all of the parameters. For example if the minutes and hours alarm is set, then the interrupt will only assert when the minutes AND the hours change to the specified setting. Use the RTC_C_ALARM_OFF for any alarm settings that should not be apart of the alarm condition.

Parameters:

baseAddress is the base address of the RTC_C module.

param is the pointer to struct for calendar alarm configuration.

Returns:

None

25.2.2.4 uint16_t RTC_C_convertBCDToBinary (uint16_t *baseAddress*, uint16_t *valueToConvert*)

Convert the given BCD value to binary format.

This function converts BCD values to binary format. This API uses the hardware registers to perform the conversion rather than a software method.

Parameters:

baseAddress is the base address of the RTC_C module.

valueToConvert is the raw value in BCD format to convert to Binary.
Modified bits are **BCD2BIN** of **BCD2BIN** register.

Returns:

The binary version of the input parameter

25.2.2.5 uint16_t RTC_C_convertBinaryToBCD (uint16_t *baseAddress*, uint16_t *valueToConvert*)

Convert the given binary value to BCD format.

This function converts binary values to BCD format. This API uses the hardware registers to perform the conversion rather than a software method.

Parameters:

baseAddress is the base address of the RTC_C module.

valueToConvert is the raw value in Binary format to convert to BCD.
Modified bits are **BIN2BCD** of **BIN2BCD** register.

Returns:

The BCD version of the valueToConvert parameter

25.2.2.6 void RTC_C_definePrescaleEvent (uint16_t *baseAddress*, uint8_t *prescaleSelect*, uint8_t *prescaleEventDivider*)

Sets up an interrupt condition for the selected Prescaler.

This function sets the condition for an interrupt to assert based on the individual prescalers.

Parameters:

baseAddress is the base address of the RTC_C module.

prescaleSelect is the prescaler to define an interrupt for. Valid values are:

- **RTC_C_PRESCALE_0**
- **RTC_C_PRESCALE_1**

prescaleEventDivider is a divider to specify when an interrupt can occur based on the clock source of the selected prescaler. (Does not affect timer of the selected prescaler). Valid values are:

- **RTC_C_PSEVENTDIVIDER_2** [Default]
- **RTC_C_PSEVENTDIVIDER_4**
- **RTC_C_PSEVENTDIVIDER_8**
- **RTC_C_PSEVENTDIVIDER_16**
- **RTC_C_PSEVENTDIVIDER_32**
- **RTC_C_PSEVENTDIVIDER_64**
- **RTC_C_PSEVENTDIVIDER_128**
- **RTC_C_PSEVENTDIVIDER_256**

Modified bits are **RTxIP** of **RTCPSxCTL** register.

Returns:

None

25.2.2.7 void RTC_C_disableInterrupt (uint16_t *baseAddress*, uint8_t *interruptMask*)

Disables selected RTC interrupt sources.

This function disables the selected RTC interrupt source. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Parameters:

baseAddress is the base address of the RTC_C module.

interruptMask is a bit mask of the interrupts to disable. Mask value is the logical OR of any of the following:

- **RTC_C_TIME_EVENT_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC_C_CLOCK_ALARM_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC_C_CLOCK_READ_READY_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC_C_PRESCALE_TIMER0_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC_C_PRESCALE_TIMER1_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC_C_OSCILLATOR_FAULT_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

Returns:

None

25.2.2.8 void RTC_C_enableInterrupt (uint16_t *baseAddress*, uint8_t *interruptMask*)

Enables selected RTC interrupt sources.

This function enables the selected RTC interrupt source. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. Does not clear interrupt flags.

Parameters:

baseAddress is the base address of the RTC_C module.

interruptMask is a bit mask of the interrupts to enable. Mask value is the logical OR of any of the following:

- **RTC_C_TIME_EVENT_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC_C_CLOCK_ALARM_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC_C_CLOCK_READ_READY_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC_C_PRESCALE_TIMER0_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC_C_PRESCALE_TIMER1_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC_C_OSCILLATOR_FAULT_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

Returns:

None

25.2.2.9 Calendar RTC_C_getCalendarTime (uint16_t *baseAddress*)

Returns the Calendar Time stored in the Calendar registers of the RTC.

This function returns the current Calendar time in the form of a Calendar structure. The `RTCRDY` polling is used in this function to prevent reading invalid time.

Parameters:

baseAddress is the base address of the RTC_C module.

Returns:

A Calendar structure containing the current time.

25.2.2.10 uint32_t RTC_C_getCounterValue (uint16_t baseAddress)

Returns the value of the Counter register.

This function returns the value of the counter register for the RTC_C module. It will return the 32-bit value no matter the size set during initialization. The RTC should be held before trying to use this function.

Parameters:

baseAddress is the base address of the RTC_C module.

Returns:

The raw value of the full 32-bit Counter Register.

25.2.2.11 uint8_t RTC_C_getInterruptStatus (uint16_t baseAddress, uint8_t interruptFlagMask)

Returns the status of the selected interrupts flags.

This function returns the status of the interrupt flag for the selected channel.

Parameters:

baseAddress is the base address of the RTC_C module.

interruptFlagMask is a bit mask of the interrupt flags to return the status of. Mask value is the logical OR of any of the following:

- **RTC_C_TIME_EVENT_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC_C_CLOCK_ALARM_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC_C_CLOCK_READ_READY_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC_C_PRESCALE_TIMER0_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC_C_PRESCALE_TIMER1_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC_C_OSCILLATOR_FAULT_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

Returns:

Logical OR of any of the following:

- **RTC_C_TIME_EVENT_INTERRUPT** asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
 - **RTC_C_CLOCK_ALARM_INTERRUPT** asserts when alarm condition in Calendar mode is met.
 - **RTC_C_CLOCK_READ_READY_INTERRUPT** asserts when Calendar registers are settled.
 - **RTC_C_PRESCALE_TIMER0_INTERRUPT** asserts when Prescaler 0 event condition is met.
 - **RTC_C_PRESCALE_TIMER1_INTERRUPT** asserts when Prescaler 1 event condition is met.
 - **RTC_C_OSCILLATOR_FAULT_INTERRUPT** asserts if there is a problem with the 32kHz oscillator, while the RTC is running.
- indicating the status of the masked interrupts

25.2.2.12 uint8_t RTC_C_getPrescaleValue (uint16_t baseAddress, uint8_t prescaleSelect)

Returns the selected prescaler value.

This function returns the value of the selected prescale counter register. Note that the counter value should be held by calling `RTC_C_holdClock()` before calling this API.

Parameters:

baseAddress is the base address of the RTC_C module.

prescaleSelect is the prescaler to obtain the value of. Valid values are:

- **RTC_C_PRESCALE_0**
- **RTC_C_PRESCALE_1**

Returns:

The value of the specified prescaler count register

25.2.2.13 void RTC_C_holdClock (uint16_t *baseAddress*)

Holds the RTC.

This function sets the RTC main hold bit to disable RTC functionality.

Parameters:

baseAddress is the base address of the RTC_C module.

Returns:

None

25.2.2.14 void RTC_C_holdCounterPrescale (uint16_t *baseAddress*, uint8_t *prescaleSelect*)

Holds the selected Prescaler.

This function holds the prescale counter from continuing. This will only work in counter mode, in Calendar mode, the [RTC_C_holdClock\(\)](#) must be used. In counter mode, if using both prescalers in conjunction with the main RTC counter, then stopping RT0PS will stop RT1PS, but stopping RT1PS will not stop RT0PS.

Parameters:

baseAddress is the base address of the RTC_C module.

prescaleSelect is the prescaler to hold. Valid values are:

- **RTC_C_PRESCALE_0**
- **RTC_C_PRESCALE_1**

Returns:

None

25.2.2.15 void RTC_C_initCalendar (uint16_t *baseAddress*, Calendar * *CalendarTime*, uint16_t *formatSelect*)

Initializes the settings to operate the RTC in calendar mode.

This function initializes the Calendar mode of the RTC module.

Parameters:

baseAddress is the base address of the RTC_C module.

CalendarTime is the pointer to the structure containing the values for the Calendar to be initialized to. Valid values should be of type pointer to Calendar and should contain the following members and corresponding values:

Seconds between 0-59 **Minutes** between 0-59 **Hours** between 0-24 **DayOfWeek** between 0-6 **DayOfMonth** between 0-31 **Year** between 0-4095 NOTE: Values beyond the ones specified may result in erratic behavior.

formatSelect is the format for the Calendar registers to use. Valid values are:

- **RTC_C_FORMAT_BINARY** [Default]
- **RTC_C_FORMAT_BCD**
Modified bits are **RTCBCD** of **RTCCTL1** register.

Returns:

None

25.2.2.16 void RTC_C_initCounter (uint16_t *baseAddress*, uint16_t *clockSelect*, uint16_t *counterSizeSelect*)

Initializes the settings to operate the RTC in Counter mode.

This function initializes the Counter mode of the RTC_C. Setting the clock source and counter size will allow an interrupt from the RTCTEVIFG once an overflow to the counter register occurs.

Parameters:

baseAddress is the base address of the RTC_C module.

clockSelect is the selected clock for the counter mode to use. Valid values are:

- **RTC_C_CLOCKSELECT_32KHZ_OSC**
 - **RTC_C_CLOCKSELECT_RT1PS**
- Modified bits are **RTCSEL** of **RTCCTL1** register.

counterSizeSelect is the size of the counter. Valid values are:

- **RTC_C_COUNTERSIZE_8BIT** [Default]
 - **RTC_C_COUNTERSIZE_16BIT**
 - **RTC_C_COUNTERSIZE_24BIT**
 - **RTC_C_COUNTERSIZE_32BIT**
- Modified bits are **RTCCTEV** of **RTCCTL1** register.

Returns:

None

25.2.2.17 void RTC_C_initCounterPrescale (uint16_t baseAddress, uint8_t prescaleSelect, uint16_t prescaleClockSelect, uint16_t prescaleDivider)

Initializes the Prescaler for Counter mode.

This function initializes the selected prescaler for the counter mode in the RTC_C module. If the RTC is initialized in Calendar mode, then these are automatically initialized. The Prescalers can be used to divide a clock source additionally before it gets to the main RTC clock.

Parameters:

baseAddress is the base address of the RTC_C module.

prescaleSelect is the prescaler to initialize. Valid values are:

- **RTC_C_PRESCALE_0**
- **RTC_C_PRESCALE_1**

prescaleClockSelect is the clock to drive the selected prescaler. Valid values are:

- **RTC_C_PSCLOCKSELECT_ACLK**
 - **RTC_C_PSCLOCKSELECT_SMCLK**
 - **RTC_C_PSCLOCKSELECT_RT0PS** - use Prescaler 0 as source to Prescaler 1 (May only be used if prescaleSelect is **RTC_C_PRESCALE_1**)
- Modified bits are **RTxSSEL** of **RTCPSxCTL** register.

prescaleDivider is the divider for the selected clock source. Valid values are:

- **RTC_C_PSDIVIDER_2** [Default]
 - **RTC_C_PSDIVIDER_4**
 - **RTC_C_PSDIVIDER_8**
 - **RTC_C_PSDIVIDER_16**
 - **RTC_C_PSDIVIDER_32**
 - **RTC_C_PSDIVIDER_64**
 - **RTC_C_PSDIVIDER_128**
 - **RTC_C_PSDIVIDER_256**
- Modified bits are **RTxPSDIV** of **RTCPSxCTL** register.

Returns:

None

25.2.2.18 void RTC_C_setCalendarAlarm (uint16_t baseAddress, uint8_t minutesAlarm, uint8_t hoursAlarm, uint8_t dayOfWeekAlarm, uint8_t dayOfMonthAlarm)

DEPRECATED - Sets and Enables the desired Calendar Alarm settings.

This function sets a Calendar interrupt condition to assert the RTCAIFG interrupt flag. The condition is a logical and of all of the parameters. For example if the minutes and hours alarm is set, then the interrupt will only assert when the minutes AND the hours change to the specified setting. Use the **RTC_C_ALARM_OFF** for any alarm settings that should not be apart of the alarm condition.

Parameters:

baseAddress is the base address of the RTC_C module.

minutesAlarm is the alarm condition for the minutes. Valid values are:

- **RTC_C_ALARMCONDITION_OFF** [Default]
- An integer between 0-59

hoursAlarm is the alarm condition for the hours. Valid values are:

- **RTC_C_ALARMCONDITION_OFF** [Default]
- An integer between 0-24

dayOfWeekAlarm is the alarm condition for the day of week. Valid values are:

- **RTC_C_ALARMCONDITION_OFF** [Default]
- An integer between 0-6

dayOfMonthAlarm is the alarm condition for the day of the month. Valid values are:

- **RTC_C_ALARMCONDITION_OFF** [Default]
- An integer between 0-31

Returns:

None

25.2.2.19 void RTC_C_setCalendarEvent (uint16_t baseAddress, uint16_t eventSelect)

Sets a single specified Calendar interrupt condition.

This function sets a specified event to assert the RTCTEVIFG interrupt. This interrupt is independent from the Calendar alarm interrupt.

Parameters:

baseAddress is the base address of the RTC_C module.

eventSelect is the condition selected. Valid values are:

- **RTC_C_CALENDAREVENT_MINUTECHANGE** - assert interrupt on every minute
- **RTC_C_CALENDAREVENT_HOURLCHANGE** - assert interrupt on every hour
- **RTC_C_CALENDAREVENT_NOON** - assert interrupt when hour is 12
- **RTC_C_CALENDAREVENT_MIDNIGHT** - assert interrupt when hour is 0

Modified bits are **RTCTEV** of **RTCCTL** register.

Returns:

None

25.2.2.20 void RTC_C_setCalibrationData (uint16_t baseAddress, uint8_t offsetDirection, uint8_t offsetValue)

Sets the specified calibration for the RTC.

This function sets the calibration offset to make the RTC as accurate as possible. The offsetDirection can be either +4-ppm or -2-ppm, and the offsetValue should be from 1-63 and is multiplied by the direction setting (i.e. +4-ppm * 8 (offsetValue) = +32-ppm).

Parameters:

baseAddress is the base address of the RTC_C module.

offsetDirection is the direction that the calibration offset will go. Valid values are:

- **RTC_C_CALIBRATION_DOWN1PPM** - calibrate at steps of -1
- **RTC_C_CALIBRATION_UP1PPM** - calibrate at steps of +1

Modified bits are **RTC0CALs** of **RTC0CAL** register.

offsetValue is the value that the offset will be a factor of; a valid value is any integer from 1-240.

Modified bits are **RTC0CALx** of **RTC0CAL** register.

Returns:

None

25.2.2.21 void RTC_C_setCalibrationFrequency (uint16_t *baseAddress*, uint16_t *frequencySelect*)

Allows and Sets the frequency output to RTCCLK pin for calibration measurement.

This function sets a frequency to measure at the RTCCLK output pin. After testing the set frequency, the calibration could be set accordingly.

Parameters:

baseAddress is the base address of the RTC_C module.

frequencySelect is the frequency output to RTCCLK. Valid values are:

- **RTC_C_CALIBRATIONFREQ_OFF** [Default] - turn off calibration output
- **RTC_C_CALIBRATIONFREQ_512HZ** - output signal at 512Hz for calibration
- **RTC_C_CALIBRATIONFREQ_256HZ** - output signal at 256Hz for calibration
- **RTC_C_CALIBRATIONFREQ_1HZ** - output signal at 1Hz for calibration

Modified bits are **RTCCALF** of **RTCCTL3** register.

Returns:

None

25.2.2.22 void RTC_C_setCounterValue (uint16_t *baseAddress*, uint32_t *counterValue*)

Sets the value of the Counter register.

This function sets the counter register of the RTC_C module.

Parameters:

baseAddress is the base address of the RTC_C module.

counterValue is the value to set the Counter register to; a valid value may be any 32-bit integer.

Returns:

None

25.2.2.23 void RTC_C_setPrescaleValue (uint16_t *baseAddress*, uint8_t *prescaleSelect*, uint8_t *prescaleCounterValue*)

Sets the selected Prescaler value.

This function sets the prescale counter value. Before setting the prescale counter, it should be held by calling [RTC_C_holdClock\(\)](#).

Parameters:

baseAddress is the base address of the RTC_C module.

prescaleSelect is the prescaler to set the value for. Valid values are:

- **RTC_C_PRESCALE_0**
- **RTC_C_PRESCALE_1**

prescaleCounterValue is the specified value to set the prescaler to. Valid values are any integer between 0-255

Modified bits are **RTxPS** of **RTxPS** register.

Returns:

None

25.2.2.24 bool RTC_C_setTemperatureCompensation (uint16_t *baseAddress*, uint16_t *offsetDirection*, uint8_t *offsetValue*)

Sets the specified temperature compensation for the RTC.

This function sets the calibration offset to make the RTC as accurate as possible. The *offsetDirection* can be either +1-ppm or -1-ppm, and the *offsetValue* should be from 1-240 and is multiplied by the direction setting (i.e. +1-ppm * 8 (*offsetValue*) = +8-ppm).

Parameters:

baseAddress is the base address of the RTC_C module.

offsetDirection is the direction that the calibration offset will go. Valid values are:

- **RTC_C_COMPENSATION_DOWN1PPM**
 - **RTC_C_COMPENSATION_UP1PPM**
- Modified bits are **RTCTCMPS** of **RTCTCMP** register.

offsetValue is the value that the offset will be a factor of; a valid value is any integer from 1-240. Modified bits are **RTCTCMPx** of **RTCTCMP** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of setting the temperature compensation

25.2.2.25 void RTC_C_startClock (uint16_t *baseAddress*)

Starts the RTC.

This function clears the RTC main hold bit to allow the RTC to function.

Parameters:

baseAddress is the base address of the RTC_C module.

Returns:

None

25.2.2.26 void RTC_C_startCounterPrescale (uint16_t *baseAddress*, uint8_t *prescaleSelect*)

Starts the selected Prescaler.

This function starts the selected prescale counter. This function will only work if the RTC is in counter mode.

Parameters:

baseAddress is the base address of the RTC_C module.

prescaleSelect is the prescaler to start. Valid values are:

- **RTC_C_PRESCALE_0**
- **RTC_C_PRESCALE_1**

Returns:

None

25.3 Programming Example

The following example shows how to initialize and use the RTC_C API to setup Calendar Mode with the current time and various interrupts.

```

//Initialize calendar struct
Calendar currentTime;
currentTime.Seconds    = 0x00;
currentTime.Minutes    = 0x26;
currentTime.Hours      = 0x13;
currentTime.DayOfWeek  = 0x03;
currentTime.DayOfMonth = 0x20;
currentTime.Month       = 0x07;
currentTime.Year        = 0x2011;

//Initialize alarm struct
RTC_C_configureCalendarAlarmParam alarmParam;
alarmParam.minutesAlarm = 0x00;
alarmParam.hoursAlarm   = 0x17;
alarmParam.dayOfWeekAlarm = RTC_C_ALARMCONDITION_OFF;
alarmParam.dayOfMonthAlarm = 0x05;

//Initialize Calendar Mode of RTC_C
/*
Base Address of the RTC_C_A
Pass in current time, initialized above
Use BCD as Calendar Register Format
*/
RTC_C_initCalendar(RTC_C_BASE,
                  &currentTime,
                  RTC_C_FORMAT_BCD);

//Setup Calendar Alarm for 5:00pm on the 5th day of the month.
//Note: Does not specify day of the week.
RTC_C_setCalendarAlarm(RTC_C_BASE, &alarmParam);

//Specify an interrupt to assert every minute
RTC_C_setCalendarEvent(RTC_C_BASE,
                      RTC_C_CALENDAREVENT_MINUTECHANGE);

//Enable interrupt for RTC_C Ready Status, which asserts when the RTC_C
//Calendar registers are ready to read.
//Also, enable interrupts for the Calendar alarm and Calendar event.
RTC_C_enableInterrupt(RTC_C_BASE,
                     RTC_C_CLOCK_READ_READY_INTERRUPT +
                     RTC_C_TIME_EVENT_INTERRUPT +
                     RTC_C_CLOCK_ALARM_INTERRUPT);

//Start RTC_C Clock
RTC_C_startClock(RTC_C_BASE);

//Enter LPM3 mode with interrupts enabled
__bis_SR_register(LPM3_bits + GIE);
__no_operation();

```


26 SFR Module

Introduction	294
API Functions	294
Programming Example	299

26.1 Introduction

The Special Function Registers API provides a set of functions for using the MSP430Ware SFR module. Functions are provided to enable and disable interrupts and control the \sim RST/NMI pin

The SFR module can enable interrupts to be generated from other peripherals of the device.

This driver is contained in `sfr.c`, with `sfr.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	114
TI Compiler 4.2.1	Size	58
TI Compiler 4.2.1	Speed	60
IAR 5.51.6	None	56
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	316
MSPGCC 4.8.0	Size	68
MSPGCC 4.8.0	Speed	68

26.2 API Functions

Functions

- void [SFR_clearInterrupt](#) (uint8_t interruptFlagMask)
- void [SFR_disableInterrupt](#) (uint8_t interruptMask)
- void [SFR_enableInterrupt](#) (uint8_t interruptMask)
- uint8_t [SFR_getInterruptStatus](#) (uint8_t interruptFlagMask)
- void [SFR_setNMIEdge](#) (uint16_t edgeDirection)
- void [SFR_setResetNMIPinFunction](#) (uint8_t resetPinFunction)
- void [SFR_setResetPinPullResistor](#) (uint16_t pullResistorSetup)

26.2.1 Detailed Description

The SFR API is broken into 2 groups: the SFR interrupts and the SFR \sim RST/NMI pin control

The SFR interrupts are handled by

- [SFR_enableInterrupt\(\)](#)
- [SFR_disableInterrupt\(\)](#)

- [SFR_getInterruptStatus\(\)](#)
- [SFR_clearInterrupt\(\)](#)

The SFR \sim RST/NMI pin is controlled by

- [SFR_setResetPinPullResistor\(\)](#)
- [SFR_setNMIEdge\(\)](#)
- [SFR_setResetNMIPinFunction\(\)](#)

26.2.2 Function Documentation

26.2.2.1 void SFR_clearInterrupt (uint8_t *interruptFlagMask*)

Clears the selected SFR interrupt flags.

This function clears the status of the selected SFR interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

interruptFlagMask is the bit mask of interrupt flags that will be cleared. Mask value is the logical OR of any of the following:

- **SFR_JTAG_OUTBOX_INTERRUPT** - JTAG outbox interrupt
- **SFR_JTAG_INBOX_INTERRUPT** - JTAG inbox interrupt
- **SFR_NMI_PIN_INTERRUPT** - NMI pin interrupt, if NMI function is chosen
- **SFR_VACANT_MEMORY_ACCESS_INTERRUPT** - Vacant memory access interrupt
- **SFR_OSCILLATOR_FAULT_INTERRUPT** - Oscillator fault interrupt
- **SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT** - Watchdog interval timer interrupt

Returns:

None

26.2.2.2 void SFR_disableInterrupt (uint8_t *interruptMask*)

Disables selected SFR interrupt sources.

This function disables the selected SFR interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

interruptMask is the bit mask of interrupts that will be disabled. Mask value is the logical OR of any of the following:

- **SFR_JTAG_OUTBOX_INTERRUPT** - JTAG outbox interrupt
- **SFR_JTAG_INBOX_INTERRUPT** - JTAG inbox interrupt
- **SFR_NMI_PIN_INTERRUPT** - NMI pin interrupt, if NMI function is chosen
- **SFR_VACANT_MEMORY_ACCESS_INTERRUPT** - Vacant memory access interrupt
- **SFR_OSCILLATOR_FAULT_INTERRUPT** - Oscillator fault interrupt
- **SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT** - Watchdog interval timer interrupt

Returns:

None

26.2.2.3 void SFR_enableInterrupt (uint8_t *interruptMask*)

Enables selected SFR interrupt sources.

This function enables the selected SFR interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. Does not clear interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	32
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

interruptMask is the bit mask of interrupts that will be enabled. Mask value is the logical OR of any of the following:

- **SFR_JTAG_OUTBOX_INTERRUPT** - JTAG outbox interrupt
- **SFR_JTAG_INBOX_INTERRUPT** - JTAG inbox interrupt
- **SFR_NMI_PIN_INTERRUPT** - NMI pin interrupt, if NMI function is chosen
- **SFR_VACANT_MEMORY_ACCESS_INTERRUPT** - Vacant memory access interrupt
- **SFR_OSCILLATOR_FAULT_INTERRUPT** - Oscillator fault interrupt
- **SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT** - Watchdog interval timer interrupt

Returns:

None

26.2.2.4 uint8_t SFR_getInterruptStatus (uint8_t *interruptFlagMask*)

Returns the status of the selected SFR interrupt flags.

This function returns the status of the selected SFR interrupt flags in a bit mask format matching that passed into the *interruptFlagMask* parameter.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

interruptFlagMask is the bit mask of interrupt flags that the status of should be returned. Mask value is the logical OR of any of the following:

- **SFR_JTAG_OUTBOX_INTERRUPT** - JTAG outbox interrupt
- **SFR_JTAG_INBOX_INTERRUPT** - JTAG inbox interrupt
- **SFR_NMI_PIN_INTERRUPT** - NMI pin interrupt, if NMI function is chosen
- **SFR_VACANT_MEMORY_ACCESS_INTERRUPT** - Vacant memory access interrupt
- **SFR_OSCILLATOR_FAULT_INTERRUPT** - Oscillator fault interrupt
- **SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT** - Watchdog interval timer interrupt

Returns:

A bit mask of the status of the selected interrupt flags. Return Logical OR of any of the following:

- **SFR_JTAG_OUTBOX_INTERRUPT** JTAG outbox interrupt
- **SFR_JTAG_INBOX_INTERRUPT** JTAG inbox interrupt
- **SFR_NMI_PIN_INTERRUPT** NMI pin interrupt, if NMI function is chosen
- **SFR_VACANT_MEMORY_ACCESS_INTERRUPT** Vacant memory access interrupt
- **SFR_OSCILLATOR_FAULT_INTERRUPT** Oscillator fault interrupt
- **SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT** Watchdog interval timer interrupt indicating the status of the masked interrupts

26.2.2.5 void SFR_setNMIEdge (uint16_t *edgeDirection*)

Sets the edge direction that will assert an NMI from a signal on the \sim RST/NMI pin if NMI function is active.

This function sets the edge direction that will assert an NMI from a signal on the \sim RST/NMI pin if the NMI function is active. To activate the NMI function of the \sim RST/NMI use the [SFR_setResetNMIPinFunction\(\)](#) passing SFR_RESETPINFUNC_NMI into the resetPinFunction parameter.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

edgeDirection is the direction that the signal on the \sim RST/NMI pin should go to signal an interrupt, if enabled. Valid values are:

- **SFR_NMI_RISINGEDGE** [Default]
 - **SFR_NMI_FALLINGEDGE**
- Modified bits are **SYSNMIIES** of **SFRRPCR** register.

Returns:

None

26.2.2.6 void SFR_setResetNMIPinFunction (uint8_t *resetPinFunction*)

Sets the function of the \sim RST/NMI pin.

This function sets the functionality of the \sim RST/NMI pin, whether in reset mode which will assert a reset if a low signal is observed on that pin, or an NMI which will assert an interrupt from an edge of the signal dependent on the setting of the edgeDirection parameter in [SFR_setNMIEdge\(\)](#).

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	52
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

resetPinFunction is the function that the \sim RST/NMI pin should take on. Valid values are:

- **SFR_RESETPINFUNC_RESET** [Default]
 - **SFR_RESETPINFUNC_NMI**
- Modified bits are **SYSNMI** of **SFRRPCR** register.

Returns:

None

26.2.2.7 void SFR_setResetPinPullResistor (uint16_t *pullResistorSetup*)

Sets the pull-up/down resistor on the \sim RST/NMI pin.

This function sets the pull-up/down resistors on the \sim RST/NMI pin to the settings from the pullResistorSetup parameter.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

pullResistorSetup is the selection of how the pull-up/down resistor on the \sim RST/NMI pin should be setup or disabled. Valid values are:

- **SFR_RESISTORDISABLE**
 - **SFR_RESISTORENABLE_PULLUP** [Default]
 - **SFR_RESISTORENABLE_PULLDOWN**
- Modified bits are **SYSRSTUP** and **SYSRSTRE** of **SFRRPCR** register.

Returns:

None

26.3 Programming Example

The following example shows how to initialize and use the SFR API

```
do
{
    // Clear SFR Fault Flag
    SFR_clearInterrupt(SFR_BASE,
                      OFIFG);

    // Test oscillator fault flag
}while (SFR_getInterruptStatus(SFR_BASE,OFIFG));
```

27 SYS Module

Introduction	300
API Functions	300
Programming Example	310

27.1 Introduction

The System Control (SYS) API provides a set of functions for using the MSP430Ware SYS module. Functions are provided to control various SYS controls, setup the BSL, and control the JTAG Mailbox.

This driver is contained in `sys.c`, with `sys.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	268
TI Compiler 4.2.1	Size	160
TI Compiler 4.2.1	Speed	162
IAR 5.51.6	None	178
IAR 5.51.6	Size	150
IAR 5.51.6	Speed	150
MSPGCC 4.8.0	None	618
MSPGCC 4.8.0	Size	196
MSPGCC 4.8.0	Speed	196

27.2 API Functions

Functions

- void [SYS_clearJTAGMailboxFlagStatus](#) (uint8_t mailboxFlagMask)
- void [SYS_disableBSLMemory](#) (void)
- void [SYS_disableBSLProtect](#) (void)
- void [SYS_disableRAMBasedInterruptVectors](#) (void)
- void [SYS_enableBSLMemory](#) (void)
- void [SYS_enableBSLProtect](#) (void)
- void [SYS_enableDedicatedJTAGPins](#) (void)
- void [SYS_enablePMMAccessProtect](#) (void)
- void [SYS_enableRAMBasedInterruptVectors](#) (void)
- uint8_t [SYS_getBSLEntryIndication](#) (void)
- uint16_t [SYS_getJTAGInboxMessage16Bit](#) (uint8_t inboxSelect)
- uint32_t [SYS_getJTAGInboxMessage32Bit](#) (void)
- uint8_t [SYS_getJTAGMailboxFlagStatus](#) (uint8_t mailboxFlagMask)
- void [SYS_JTAGMailboxInit](#) (uint8_t mailboxSizeSelect, uint8_t autoClearInboxFlagSelect)
- void [SYS_setBSLSize](#) (uint8_t BSLSizeSelect)
- void [SYS_setJTAGOutgoingMessage16Bit](#) (uint8_t outboxSelect, uint16_t outgoingMessage)
- void [SYS_setJTAGOutgoingMessage32Bit](#) (uint32_t outgoingMessage)
- void [SYS_setRAMAssignedToBSL](#) (uint8_t BSLRAMAssignment)

27.2.1 Detailed Description

The SYS API is broken into 3 groups: the various SYS controls, the BSL controls, and the JTAG mailbox controls.

The various SYS controls are handled by

- [SYS_enableDedicatedJTAGPins\(\)](#)
- [SYS_getBSLEntryIndication\(\)](#)
- [SYS_enablePMMAccessProtect\(\)](#)
- [SYS_enableRAMBasedInterruptVectors\(\)](#)
- [SYS_disableRAMBasedInterruptVectors\(\)](#)

The BSL controls are handled by

- [SYS_enableBSLProtect\(\)](#)
- [SYS_disableBSLProtect\(\)](#)
- [SYS_disableBSLMemory\(\)](#)
- [SYS_enableBSLMemory\(\)](#)
- [SYS_setRAMAssignedToBSL\(\)](#)
- [SYS_setBSLSize\(\)](#)

The JTAG Mailbox controls are handled by

- [SYS_JTAGMailboxInit\(\)](#)
- [SYS_getJTAGMailboxFlagStatus\(\)](#)
- [SYS_getJTAGInboxMessage16Bit\(\)](#)
- [SYS_getJTAGInboxMessage32Bit\(\)](#)
- [SYS_setJTAGOutgoingMessage16Bit\(\)](#)
- [SYS_setJTAGOutgoingMessage32Bit\(\)](#)
- [SYS_clearJTAGMailboxFlagStatus\(\)](#)

27.2.2 Function Documentation

27.2.2.1 void SYS_clearJTAGMailboxFlagStatus (uint8_t *mailboxFlagMask*)

Clears the status of the selected JTAG Mailbox flags.

This function clears the selected JTAG Mailbox flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	14
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

mailboxFlagMask is the bit mask of JTAG mailbox flags that the status of should be cleared. Mask value is the logical OR of any of the following:

- **SYS_JTAGOUTBOX_FLAG0** - flag for JTAG outbox 0
- **SYS_JTAGOUTBOX_FLAG1** - flag for JTAG outbox 1
- **SYS_JTAGINBOX_FLAG0** - flag for JTAG inbox 0
- **SYS_JTAGINBOX_FLAG1** - flag for JTAG inbox 1

Returns:

None

27.2.2.2 void SYS_disableBSLMemory (void)

Disables BSL memory.

This function disables BSL memory, which makes BSL memory act like vacant memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	8
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	20
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Returns:

None

27.2.2.3 void SYS_disableBSLProtect (void)

Disables BSL memory protection.

This function disables protection on the BSL memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	8
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	20
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Returns:

None

27.2.2.4 void SYS_disableRAMBasedInterruptVectors (void)

Disables RAM-based Interrupt Vectors.

This function disables the interrupt vectors from being generated at the top of the RAM.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

None

27.2.2.5 void SYS_enableBSLMemory (void)

Enables BSL memory.

This function enables BSL memory, which allows BSL memory to be addressed

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	8
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	20
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Returns:

None

27.2.2.6 void SYS_enableBSLProtect (void)

Enables BSL memory protection.

This function enables protection on the BSL memory, which prevents any reading, programming, or erasing of the BSL memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	8
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	20
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Returns:

None

27.2.2.7 void SYS_enableDedicatedJTAGPins (void)

Sets the JTAG pins to be exclusively for JTAG until a BOR occurs.

This function sets the JTAG pins to be exclusively used for the JTAG, and not to be shared with the GPIO pins. This setting can only be cleared when a BOR occurs.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	8
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	24
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Returns:

None

27.2.2.8 void SYS_enablePMMAccessProtect (void)

Enables PMM Access Protection.

This function enables the PMM Access Protection, which will lock any changes on the PMM control registers until a BOR occurs.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:
None

27.2.2.9 void SYS_enableRAMBasedInterruptVectors (void)

Enables RAM-based Interrupt Vectors.

This function enables RAM-base Interrupt Vectors, which means that interrupt vectors are generated with the end address at the top of RAM, instead of the top of the lower 64kB of flash.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	6
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:
None

27.2.2.10 uint8_t SYS_getBSLEntryIndication (void)

Returns the indication of a BSL entry sequence from the Spy-Bi-Wire.

This function returns the indication of a BSL entry sequence from the Spy- Bi-Wire.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	16
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Returns:
One of the following:

- **SYS_BSLENTY_INDICATED**
- **SYS_BSLENTY_NOTINDICATED**
indicating if a BSL entry sequence was detected

27.2.2.11 uint16_t SYS_getJTAGInboxMessage16Bit (uint8_t *inboxSelect*)

Returns the contents of the selected JTAG Inbox in a 16 bit format.

This function returns the message contents of the selected JTAG inbox. If the auto clear settings for the Inbox flags were set, then using this function will automatically clear the corresponding JTAG inbox flag.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	20
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

inboxSelect is the chosen JTAG inbox that the contents of should be returned Valid values are:

- **SYS_JTAGINBOX_0** - return contents of JTAG inbox 0
- **SYS_JTAGINBOX_1** - return contents of JTAG inbox 1

Returns:

The contents of the selected JTAG inbox in a 16 bit format.

27.2.2.12 uint32_t SYS_getJTAGInboxMessage32Bit (void)

Returns the contents of JTAG Inboxes in a 32 bit format.

This function returns the message contents of both JTAG inboxes in a 32 bit format. This function should be used if 32-bit messaging has been set in the [SYS_JTAGMailboxInit\(\)](#) function. If the auto clear settings for the Inbox flags were set, then using this function will automatically clear both JTAG inbox flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	36
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	22
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	70
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

Returns:

The contents of both JTAG messages in a 32 bit format.

27.2.2.13 uint8_t SYS_getJTAGMailboxFlagStatus (uint8_t *mailboxFlagMask*)

Returns the status of the selected JTAG Mailbox flags.

This function will return the status of the selected JTAG Mailbox flags in bit mask format matching that passed into the mailboxFlagMask parameter.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

mailboxFlagMask is the bit mask of JTAG mailbox flags that the status of should be returned. Mask value is the logical OR of any of the following:

- **SYS_JTAGOUTBOX_FLAG0** - flag for JTAG outbox 0
- **SYS_JTAGOUTBOX_FLAG1** - flag for JTAG outbox 1
- **SYS_JTAGINBOX_FLAG0** - flag for JTAG inbox 0
- **SYS_JTAGINBOX_FLAG1** - flag for JTAG inbox 1

Returns:

A bit mask of the status of the selected mailbox flags.

27.2.2.14 void SYS_JTAGMailboxInit (uint8_t mailboxSizeSelect, uint8_t autoClearInboxFlagSelect)

Initializes JTAG Mailbox with selected properties.

This function sets the specified settings for the JTAG Mailbox system. The settings that can be set are the size of the JTAG messages, and the auto- clearing of the inbox flags. If the inbox flags are set to auto-clear, then the inbox flags will be cleared upon reading of the inbox message buffer, otherwise they will have to be reset by software using the [SYS_clearJTAGMailboxFlagStatus\(\)](#) function.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	16
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	74
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:

mailboxSizeSelect is the size of the JTAG Mailboxes, whether 16- or 32-bits. Valid values are:

- **SYS_JTAGMBSIZE_16BIT** [Default] - the JTAG messages will take up only one JTAG mailbox (i. e. an outgoing message will take up only 1 outbox of the JTAG mailboxes)
 - **SYS_JTAGMBSIZE_32BIT** - the JTAG messages will be contained within both JTAG mailboxes (i. e. an outgoing message will take up both Outboxes of the JTAG mailboxes)
- Modified bits are **JMBMODE** of **SYSJMBC** register.

autoClearInboxFlagSelect decides how the JTAG inbox flags should be cleared, whether automatically after the corresponding outbox has been written to, or manually by software. Valid values are:

- **SYS_JTAGINBOX0AUTO_JTAGINBOX1AUTO** [Default] - both JTAG inbox flags will be reset automatically when the corresponding inbox is read from.

- **SYS_JTAGINBOX0AUTO_JTAGINBOX1SW** - only JTAG inbox 0 flag is reset automatically, while JTAG inbox 1 is reset with the
- **SYS_JTAGINBOX0SW_JTAGINBOX1AUTO** - only JTAG inbox 1 flag is reset automatically, while JTAG inbox 0 is reset with the
- **SYS_JTAGINBOX0SW_JTAGINBOX1SW** - both JTAG inbox flags will need to be reset manually by the Modified bits are **JMBCLR0OFF** and **JMBCLR1OFF** of **SYSJMBC** register.

Returns:
None

27.2.2.15 void SYS_setBSLSize (uint8_t *BSLSizeSelect*)

Sets the size of the BSL in Flash.

This function sets the size of the BSL in Flash memory.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	54
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

BSLSizeSelect is the amount of segments the BSL should take. Valid values are:

- **SYS_BSLSIZE_SEG3**
- **SYS_BSLSIZE_SEGS23**
- **SYS_BSLSIZE_SEGS123**
- **SYS_BSLSIZE_SEGS1234** [Default]
Modified bits are **SYSBSLSIZE** of **SYSBSLC** register.

Returns:
None

27.2.2.16 void SYS_setJTAGOutgoingMessage16Bit (uint8_t *outboxSelect*, uint16_t *outgoingMessage*)

Sets a 16 bit outgoing message in to the selected JTAG Outbox.

This function sets the outgoing message in the selected JTAG outbox. The corresponding JTAG outbox flag is cleared after this function, and set after the JTAG has read the message.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

outboxSelect is the chosen JTAG outbox that the message should be set it. Valid values are:

- **SYS_JTAGOUTBOX_0** - set the contents of JTAG outbox 0
- **SYS_JTAGOUTBOX_1** - set the contents of JTAG outbox 1

outgoingMessage is the message to send to the JTAG.

Modified bits are **MSGHI** and **MSGLO** of **SYSJMBOX** register.

Returns:

None

27.2.2.17 void SYS_setJTAGOutgoingMessage32Bit (uint32_t *outgoingMessage*)

Sets a 32 bit message in to both JTAG Outboxes.

This function sets the 32-bit outgoing message in both JTAG outboxes. The JTAG outbox flags are cleared after this function, and set after the JTAG has read the message.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	54
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:

outgoingMessage is the message to send to the JTAG.

Modified bits are **MSGHI** and **MSGLO** of **SYSJMBOX** register.

Returns:

None

27.2.2.18 void SYS_setRAMAssignedToBSL (uint8_t *BSLRAMAssignment*)

Sets RAM assignment to BSL area.

This function allows RAM to be assigned to BSL, based on the selection of the BSLRAMAssignment parameter.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	52
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

BSLRAMAssignment is the selection of if the BSL should be placed in RAM or not. Valid values are:

- **SYS_BSLRAMASSIGN_NORAM** [Default]
- **SYS_BSLRAMASSIGN_LOWEST16BYTES**
Modified bits are **SYSBSLR** of **SYSBSLC** register.

Returns:

None

27.3 Programming Example

The following example shows how to initialize and use the SYS API

```
SYS_enableBSLProtect (SYS_BASE);
```

28 16-Bit Timer_A (TIMER_A)

Introduction	311
API Functions	312
Programming Example	337

28.1 Introduction

TIMER_A is a 16-bit timer/counter with multiple capture/compare registers. TIMER_A can support multiple capture/compares, PWM outputs, and interval timing. TIMER_A also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

This peripheral API handles Timer A hardware peripheral.

TIMER_A features include:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Up to seven configurable capture/compare registers
- Configurable outputs with pulse width modulation (PWM) capability
- Asynchronous input and output latching
- Interrupt vector register for fast decoding of all Timer interrupts

TIMER_A can operate in 3 modes

- Continuous Mode
- Up Mode
- Down Mode

TIMER_A Interrupts may be generated on counter overflow conditions and during capture compare events.

The TIMER_A may also be used to generate PWM outputs. PWM outputs can be generated by initializing the compare mode with [TIMER_A_initCompare\(\)](#) and the necessary parameters. The PWM may be customized by selecting a desired timer mode (continuous/up/upDown), duty cycle, output mode, timer period etc. The library also provides a simpler way to generate PWM using [TIMER_A_generatePWM\(\)](#) API. However the level of customization and the kinds of PWM generated are limited in this API. Depending on how complex the PWM is and what level of customization is required, the user can use [TIMER_A_generatePWM\(\)](#) or a combination of [Timer_initCompare\(\)](#) and timer start APIs

The TIMER_A API provides a set of functions for dealing with the TIMER_A module. Functions are provided to configure and control the timer, along with functions to modify timer/counter values, and to manage interrupt handling for the timer.

Control is also provided over interrupt sources and events. Interrupts can be generated to indicate that an event has been captured.

This driver is contained in `TIMER_A.c`, with `TIMER_A.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	1992
TI Compiler 4.2.1	Size	1072
TI Compiler 4.2.1	Speed	1074
IAR 5.51.6	None	1500
IAR 5.51.6	Size	660
IAR 5.51.6	Speed	1094
MSPGCC 4.8.0	None	2716
MSPGCC 4.8.0	Size	1312
MSPGCC 4.8.0	Speed	1380

28.2 API Functions

Functions

- void [TIMER_A_clear](#) (uint16_t baseAddress)
- void [TIMER_A_clearCaptureCompareInterruptFlag](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- void [TIMER_A_clearTimerInterruptFlag](#) (uint16_t baseAddress)
- void [TIMER_A_configureContinuousMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TAIE, uint16_t timerClear)
- void [TIMER_A_configureUpDownMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t timerInterruptEnable_TAIE, uint16_t captureCompareInterruptEnable_CCR0_CCIE, uint16_t timerClear)
- void [TIMER_A_configureUpMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t timerInterruptEnable_TAIE, uint16_t captureCompareInterruptEnable_CCR0_CCIE, uint16_t timerClear)
- void [TIMER_A_disableCaptureCompareInterrupt](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- void [TIMER_A_disableInterrupt](#) (uint16_t baseAddress)
- void [TIMER_A_enableCaptureCompareInterrupt](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- void [TIMER_A_enableInterrupt](#) (uint16_t baseAddress)
- void [TIMER_A_generatePWM](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t compareRegister, uint16_t compareOutputMode, uint16_t dutyCycle)
- uint16_t [TIMER_A_getCaptureCompareCount](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- uint32_t [TIMER_A_getCaptureCompareInterruptStatus](#) (uint16_t baseAddress, uint16_t captureCompareRegister, uint16_t mask)
- uint16_t [TIMER_A_getCounterValue](#) (uint16_t baseAddress)
- uint32_t [TIMER_A_getInterruptStatus](#) (uint16_t baseAddress)
- uint8_t [TIMER_A_getOutputForOutputModeOutBitValue](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- uint8_t [TIMER_A_getSynchronizedCaptureCompareInput](#) (uint16_t baseAddress, uint16_t captureCompareRegister, uint16_t synchronized)
- void [TIMER_A_initCapture](#) (uint16_t baseAddress, uint16_t captureRegister, uint16_t captureMode, uint16_t captureInputSelect, uint16_t synchronizeCaptureSource, uint16_t captureInterruptEnable, uint16_t captureOutputMode)
- void [TIMER_A_initCaptureMode](#) (uint16_t baseAddress, [TIMER_A_initCaptureModeParam](#) *param)
- void [TIMER_A_initCompare](#) (uint16_t baseAddress, uint16_t compareRegister, uint16_t compareInterruptEnable, uint16_t compareOutputMode, uint16_t compareValue)
- void [TIMER_A_initCompareMode](#) (uint16_t baseAddress, [TIMER_A_initCompareModeParam](#) *param)
- void [TIMER_A_initContinuousMode](#) (uint16_t baseAddress, [TIMER_A_initContinuousModeParam](#) *param)
- void [TIMER_A_initUpDownMode](#) (uint16_t baseAddress, [TIMER_A_initUpDownModeParam](#) *param)
- void [TIMER_A_initUpMode](#) (uint16_t baseAddress, [TIMER_A_initUpModeParam](#) *param)
- void [TIMER_A_outputPWM](#) (uint16_t baseAddress, [TIMER_A_outputPWMParm](#) *param)
- void [TIMER_A_setCompareValue](#) (uint16_t baseAddress, uint16_t compareRegister, uint16_t compareValue)
- void [TIMER_A_setOutputForOutputModeOutBitValue](#) (uint16_t baseAddress, uint16_t captureCompareRegister, uint8_t outputModeOutBitValue)
- void [TIMER_A_startContinuousMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TAIE, uint16_t timerClear)

- void [TIMER_A_startContinuousMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TAIE, uint16_t timerClear)
- void [TIMER_A_startCounter](#) (uint16_t baseAddress, uint16_t timerMode)
- void [TIMER_A_startUpDownMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t timerInterruptEnable_TAIE, uint16_t captureCompareInterruptEnable_CCR0_CCIE, uint16_t timerClear)
- void [TIMER_A_startUpMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t timerInterruptEnable_TAIE, uint16_t captureCompareInterruptEnable_CCR0_CCIE, uint16_t timerClear)
- void [TIMER_A_stop](#) (uint16_t baseAddress)

28.2.1 Detailed Description

The TIMER_A API is broken into three groups of functions: those that deal with timer configuration and control, those that deal with timer contents, and those that deal with interrupt handling.

TIMER_A configuration and initialization is handled by

- [TIMER_A_startCounter\(\)](#)
- [TIMER_A_configureContinuousMode\(\)](#)
- [TIMER_A_configureUpMode\(\)](#)
- [TIMER_A_configureUpDownMode\(\)](#)
- [TIMER_A_startContinuousMode\(\)](#)
- [TIMER_A_startUpMode\(\)](#)
- [TIMER_A_startUpDownMode\(\)](#)
- [TIMER_A_initCapture\(\)](#)
- [TIMER_A_initCompare\(\)](#)
- [TIMER_A_clear\(\)](#)
- [TIMER_A_stop\(\)](#)

TIMER_A outputs are handled by

- [TIMER_A_getSynchronizedCaptureCompareInput\(\)](#)
- [TIMER_A_getOutputForOutputModeOutBitValue\(\)](#)
- [TIMER_A_setOutputForOutputModeOutBitValue\(\)](#)
- [TIMER_A_generatePWM\(\)](#)
- [TIMER_A_getCaptureCompareCount\(\)](#)
- [TIMER_A_setCompareValue\(\)](#)
- [TIMER_A_getCounterValue\(\)](#)

The interrupt handler for the TIMER_A interrupt is managed with

- [TIMER_A_enableInterrupt\(\)](#)
- [TIMER_A_disableInterrupt\(\)](#)
- [TIMER_A_getInterruptStatus\(\)](#)
- [TIMER_A_enableCaptureCompareInterrupt\(\)](#)
- [TIMER_A_disableCaptureCompareInterrupt\(\)](#)
- [TIMER_A_getCaptureCompareInterruptStatus\(\)](#)
- [TIMER_A_clearCaptureCompareInterruptFlag\(\)](#)
- [TIMER_A_clearTimerInterruptFlag\(\)](#)

28.2.2 Function Documentation

28.2.2.1 void TIMER_A_clear (uint16_t *baseAddress*)

Reset/Clear the timer clock divider, count direction, count.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the TIMER_A module.

Modified bits of **TAxCTL** register.

Returns:

None

28.2.2.2 void TIMER_A_clearCaptureCompareInterruptFlag (uint16_t *baseAddress*, uint16_t *captureCompareRegister*)

Clears the capture-compare interrupt flag.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the TIMER_A module.

captureCompareRegister selects the Capture-compare register being used. Valid values are:

- **TIMER_A_CAPTURECOMPARE_REGISTER_0**
- **TIMER_A_CAPTURECOMPARE_REGISTER_1**
- **TIMER_A_CAPTURECOMPARE_REGISTER_2**
- **TIMER_A_CAPTURECOMPARE_REGISTER_3**
- **TIMER_A_CAPTURECOMPARE_REGISTER_4**
- **TIMER_A_CAPTURECOMPARE_REGISTER_5**
- **TIMER_A_CAPTURECOMPARE_REGISTER_6**

Modified bits are **CCIFG** of **TAxCTLn** register.

Returns:

None

28.2.2.3 void TIMER_A_clearTimerInterruptFlag (uint16_t *baseAddress*)

Clears the Timer TAIFG interrupt flag.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the TIMER_A module.

Modified bits are **TAIFG** of **TAxCTL** register.

Returns:

None

28.2.2.4 void TIMER_A_configureContinuousMode (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerInterruptEnable_TAIE*, uint16_t *timerClear*)

DEPRECATED - Configures TIMER_A in continuous mode.

This API does not start the timer. Timer needs to be started when required using the TIMER_A_startCounter API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	80
TI Compiler 4.2.1	Size	38
TI Compiler 4.2.1	Speed	38
IAR 5.51.6	None	68
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	60
MSPGCC 4.8.0	Speed	52

Parameters:

baseAddress is the base address of the TIMER_A module.

clockSource selects Clock source. Valid values are:

- **TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]

- **TIMER_A_CLOCKSOURCE_ACLK**
- **TIMER_A_CLOCKSOURCE_SMCLK**
- **TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the desired divider for the clock source Valid values are:

- **TIMER_A_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_A_CLOCKSOURCE_DIVIDER_2**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_10**
- **TIMER_A_CLOCKSOURCE_DIVIDER_12**
- **TIMER_A_CLOCKSOURCE_DIVIDER_14**
- **TIMER_A_CLOCKSOURCE_DIVIDER_16**
- **TIMER_A_CLOCKSOURCE_DIVIDER_20**
- **TIMER_A_CLOCKSOURCE_DIVIDER_24**
- **TIMER_A_CLOCKSOURCE_DIVIDER_28**
- **TIMER_A_CLOCKSOURCE_DIVIDER_32**
- **TIMER_A_CLOCKSOURCE_DIVIDER_40**
- **TIMER_A_CLOCKSOURCE_DIVIDER_48**
- **TIMER_A_CLOCKSOURCE_DIVIDER_56**
- **TIMER_A_CLOCKSOURCE_DIVIDER_64**

timerInterruptEnable_TAIE is to enable or disable TIMER_A interrupt Valid values are:

- **TIMER_A_TAIE_INTERRUPT_ENABLE**
- **TIMER_A_TAIE_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_A clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_A_DO_CLEAR**
- **TIMER_A_SKIP_CLEAR** [Default]

Modified bits of **TAxCTL** register.

Returns:

None

28.2.2.5 void **TIMER_A_configureUpDownMode** (uint16_t *baseAddress*,
uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t
timerPeriod, uint16_t *timerInterruptEnable_TAIE*, uint16_t
captureCompareInterruptEnable_CCR0_CCIE, uint16_t *timerClear*)

DEPRECATED - Configures TIMER_A in up down mode.

This API does not start the timer. Timer needs to be started when required using the **TIMER_A_startCounter** API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	92
TI Compiler 4.2.1	Size	54
TI Compiler 4.2.1	Speed	54
IAR 5.51.6	None	84
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	88

Parameters:

baseAddress is the base address of the TIMER_A module.

clockSource selects Clock source. Valid values are:

- **TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_A_CLOCKSOURCE_ACLK**
- **TIMER_A_CLOCKSOURCE_SMCLK**
- **TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the desired divider for the clock source Valid values are:

- **TIMER_A_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_A_CLOCKSOURCE_DIVIDER_2**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_10**
- **TIMER_A_CLOCKSOURCE_DIVIDER_12**
- **TIMER_A_CLOCKSOURCE_DIVIDER_14**
- **TIMER_A_CLOCKSOURCE_DIVIDER_16**
- **TIMER_A_CLOCKSOURCE_DIVIDER_20**
- **TIMER_A_CLOCKSOURCE_DIVIDER_24**
- **TIMER_A_CLOCKSOURCE_DIVIDER_28**
- **TIMER_A_CLOCKSOURCE_DIVIDER_32**
- **TIMER_A_CLOCKSOURCE_DIVIDER_40**
- **TIMER_A_CLOCKSOURCE_DIVIDER_48**
- **TIMER_A_CLOCKSOURCE_DIVIDER_56**
- **TIMER_A_CLOCKSOURCE_DIVIDER_64**

timerPeriod is the specified TIMER_A period

timerInterruptEnable_TAIE is to enable or disable TIMER_A interrupt Valid values are:

- **TIMER_A_TAIE_INTERRUPT_ENABLE**
- **TIMER_A_TAIE_INTERRUPT_DISABLE** [Default]

captureCompareInterruptEnable_CCR0_CCIE is to enable or disable TIMER_A CCR0 captureCompare interrupt.

Valid values are:

- **TIMER_A_CCIE_CCR0_INTERRUPT_ENABLE**
- **TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_A clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_A_DO_CLEAR**
- **TIMER_A_SKIP_CLEAR** [Default]

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register and bits of **TAxCCR0** register.

Returns:

None

28.2.2.6 void **TIMER_A_configureUpMode** (uint16_t *baseAddress*,
uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t
timerPeriod, uint16_t *timerInterruptEnable_TAIE*, uint16_t
captureCompareInterruptEnable_CCR0_CCIE, uint16_t *timerClear*)

DEPRECATED - Configures TIMER_A in up mode.

This API does not start the timer. Timer needs to be started when required using the **TIMER_A_startCounter** API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	92
TI Compiler 4.2.1	Size	54
TI Compiler 4.2.1	Speed	54
IAR 5.51.6	None	84
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	88

Parameters:

baseAddress is the base address of the TIMER_A module.

clockSource selects Clock source. Valid values are:

- **TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_A_CLOCKSOURCE_ACLK**
- **TIMER_A_CLOCKSOURCE_SMCLK**
- **TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the desired divider for the clock source Valid values are:

- **TIMER_A_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_A_CLOCKSOURCE_DIVIDER_2**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_10**
- **TIMER_A_CLOCKSOURCE_DIVIDER_12**
- **TIMER_A_CLOCKSOURCE_DIVIDER_14**
- **TIMER_A_CLOCKSOURCE_DIVIDER_16**
- **TIMER_A_CLOCKSOURCE_DIVIDER_20**
- **TIMER_A_CLOCKSOURCE_DIVIDER_24**
- **TIMER_A_CLOCKSOURCE_DIVIDER_28**
- **TIMER_A_CLOCKSOURCE_DIVIDER_32**
- **TIMER_A_CLOCKSOURCE_DIVIDER_40**
- **TIMER_A_CLOCKSOURCE_DIVIDER_48**
- **TIMER_A_CLOCKSOURCE_DIVIDER_56**
- **TIMER_A_CLOCKSOURCE_DIVIDER_64**

timerPeriod is the specified TIMER_A period. This is the value that gets written into the CCR0. Limited to 16 bits[uint16_t]

timerInterruptEnable_TAIE is to enable or disable TIMER_A interrupt Valid values are:

- **TIMER_A_TAIE_INTERRUPT_ENABLE**
- **TIMER_A_TAIE_INTERRUPT_DISABLE** [Default]

captureCompareInterruptEnable_CCR0_CCIE is to enable or disable TIMER_A CCR0 captureCompare interrupt. Valid values are:

- **TIMER_A_CCIE_CCR0_INTERRUPT_ENABLE**
- **TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_A clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_A_DO_CLEAR**
- **TIMER_A_SKIP_CLEAR** [Default]

Modified bits of **TAxCTL** register, bits of **TAxCCTL0** register and bits of **TAxCCR0** register.

Returns:

None

28.2.2.7 void TIMER_A_disableCaptureCompareInterrupt (uint16_t *baseAddress*, uint16_t *captureCompareRegister*)

Disable capture compare interrupt.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the TIMER_A module.

captureCompareRegister is the selected capture compare register Valid values are:

- **TIMER_A_CAPTURECOMPARE_REGISTER_0**
- **TIMER_A_CAPTURECOMPARE_REGISTER_1**
- **TIMER_A_CAPTURECOMPARE_REGISTER_2**
- **TIMER_A_CAPTURECOMPARE_REGISTER_3**
- **TIMER_A_CAPTURECOMPARE_REGISTER_4**
- **TIMER_A_CAPTURECOMPARE_REGISTER_5**
- **TIMER_A_CAPTURECOMPARE_REGISTER_6**

Modified bits of **TAxCCTLn** register.

Returns:

None

28.2.2.8 void TIMER_A_disableInterrupt (uint16_t *baseAddress*)

Disable timer interrupt.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the TIMER_A module.

Modified bits of **TAxCTL** register.

Returns:

None

28.2.2.9 void TIMER_A_enableCaptureCompareInterrupt (uint16_t *baseAddress*, uint16_t *captureCompareRegister*)

Enable capture compare interrupt.

Does not clear interrupt flags

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the TIMER_A module.

captureCompareRegister is the selected capture compare register Valid values are:

- **TIMER_A_CAPTURECOMPARE_REGISTER_0**
- **TIMER_A_CAPTURECOMPARE_REGISTER_1**
- **TIMER_A_CAPTURECOMPARE_REGISTER_2**
- **TIMER_A_CAPTURECOMPARE_REGISTER_3**
- **TIMER_A_CAPTURECOMPARE_REGISTER_4**
- **TIMER_A_CAPTURECOMPARE_REGISTER_5**
- **TIMER_A_CAPTURECOMPARE_REGISTER_6**

Modified bits of **TAxCCTLn** register.

Returns:

None

28.2.2.10 void TIMER_A_enableInterrupt (uint16_t *baseAddress*)

Enable timer interrupt.

Does not clear interrupt flags

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the TIMER_A module.

Modified bits of **TAxCTL** register.

Returns:

None

28.2.2.11 void **TIMER_A_generatePWM** (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerPeriod*, uint16_t *compareRegister*, uint16_t *compareOutputMode*, uint16_t *dutyCycle*)

DEPRECATED - Generate a PWM with timer running in up mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	88
TI Compiler 4.2.1	Size	50
TI Compiler 4.2.1	Speed	50
IAR 5.51.6	None	80
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	68
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	76

Parameters:

baseAddress is the base address of the **TIMER_A** module.

clockSource selects Clock source. Valid values are:

- **TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_A_CLOCKSOURCE_ACLK**
- **TIMER_A_CLOCKSOURCE_SMCLK**
- **TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the desired divider for the clock source Valid values are:

- **TIMER_A_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_A_CLOCKSOURCE_DIVIDER_2**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_10**
- **TIMER_A_CLOCKSOURCE_DIVIDER_12**
- **TIMER_A_CLOCKSOURCE_DIVIDER_14**
- **TIMER_A_CLOCKSOURCE_DIVIDER_16**
- **TIMER_A_CLOCKSOURCE_DIVIDER_20**
- **TIMER_A_CLOCKSOURCE_DIVIDER_24**
- **TIMER_A_CLOCKSOURCE_DIVIDER_28**
- **TIMER_A_CLOCKSOURCE_DIVIDER_32**
- **TIMER_A_CLOCKSOURCE_DIVIDER_40**
- **TIMER_A_CLOCKSOURCE_DIVIDER_48**
- **TIMER_A_CLOCKSOURCE_DIVIDER_56**
- **TIMER_A_CLOCKSOURCE_DIVIDER_64**

timerPeriod selects the desired timer period

compareRegister selects the compare register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- `TIMER_A_CAPTURECOMPARE_REGISTER_0`
- `TIMER_A_CAPTURECOMPARE_REGISTER_1`
- `TIMER_A_CAPTURECOMPARE_REGISTER_2`
- `TIMER_A_CAPTURECOMPARE_REGISTER_3`
- `TIMER_A_CAPTURECOMPARE_REGISTER_4`
- `TIMER_A_CAPTURECOMPARE_REGISTER_5`
- `TIMER_A_CAPTURECOMPARE_REGISTER_6`

compareOutputMode specifies the output mode. Valid values are:

- `TIMER_A_OUTPUTMODE_OUTBITVALUE` [Default]
- `TIMER_A_OUTPUTMODE_SET`
- `TIMER_A_OUTPUTMODE_TOGGLE_RESET`
- `TIMER_A_OUTPUTMODE_SET_RESET`
- `TIMER_A_OUTPUTMODE_TOGGLE`
- `TIMER_A_OUTPUTMODE_RESET`
- `TIMER_A_OUTPUTMODE_TOGGLE_SET`
- `TIMER_A_OUTPUTMODE_RESET_SET`

dutyCycle specifies the dutycycle for the generated waveform

Modified bits of **TAxCTL** register, bits of **TAxCCTL0** register, bits of **TAxCCR0** register and bits of **TAxCCTLn** register.

Returns:

None

28.2.2.12 `uint16_t TIMER_A_getCaptureCompareCount (uint16_t baseAddress, uint16_t captureCompareRegister)`

Get current capturecompare count.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the `TIMER_A` module.

captureCompareRegister Valid values are:

- `TIMER_A_CAPTURECOMPARE_REGISTER_0`
- `TIMER_A_CAPTURECOMPARE_REGISTER_1`
- `TIMER_A_CAPTURECOMPARE_REGISTER_2`
- `TIMER_A_CAPTURECOMPARE_REGISTER_3`
- `TIMER_A_CAPTURECOMPARE_REGISTER_4`
- `TIMER_A_CAPTURECOMPARE_REGISTER_5`
- `TIMER_A_CAPTURECOMPARE_REGISTER_6`

Returns:

Current count as an `uint16_t`

28.2.2.13 uint32_t TIMER_A_getCaptureCompareInterruptStatus (uint16_t *baseAddress*, uint16_t *captureCompareRegister*, uint16_t *mask*)

Return capture compare interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the TIMER_A module.

captureCompareRegister is the selected capture compare register Valid values are:

- **TIMER_A_CAPTURECOMPARE_REGISTER_0**
- **TIMER_A_CAPTURECOMPARE_REGISTER_1**
- **TIMER_A_CAPTURECOMPARE_REGISTER_2**
- **TIMER_A_CAPTURECOMPARE_REGISTER_3**
- **TIMER_A_CAPTURECOMPARE_REGISTER_4**
- **TIMER_A_CAPTURECOMPARE_REGISTER_5**
- **TIMER_A_CAPTURECOMPARE_REGISTER_6**

mask is the mask for the interrupt status Mask value is the logical OR of any of the following:

- **TIMER_A_CAPTURE_OVERFLOW**
- **TIMER_A_CAPTURECOMPARE_INTERRUPT_FLAG**

Returns:

Logical OR of any of the following:

- **TIMER_A_CAPTURE_OVERFLOW**
- **TIMER_A_CAPTURECOMPARE_INTERRUPT_FLAG**
indicating the status of the masked interrupts

28.2.2.14 uint16_t TIMER_A_getCounterValue (uint16_t *baseAddress*)

Reads the current timer count value.

Reads the current count value of the timer. There is a majority vote system in place to confirm an accurate value is returned. The **TIMER_A_THRESHOLD** define in the corresponding header file can be modified so that the votes must be closer together for a consensus to occur.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	88
TI Compiler 4.2.1	Size	38
TI Compiler 4.2.1	Speed	38
IAR 5.51.6	None	58
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	102
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	48

Parameters:

baseAddress is the base address of the TIMER_A module.

Returns:

Majority vote of timer count value

28.2.2.15 uint32_t TIMER_A_getInterruptStatus (uint16_t baseAddress)

Get timer interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the TIMER_A module.

Returns:

One of the following:

- **TIMER_A_INTERRUPT_NOT_PENDING**
- **TIMER_A_INTERRUPT_PENDING**
indicating the TIMER_A interrupt status

28.2.2.16 uint8_t TIMER_A_getOutputForOutputModeOutBitValue (uint16_t baseAddress, uint16_t captureCompareRegister)

Get output bit for output mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	20
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	16
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the TIMER_A module.

captureCompareRegister Valid values are:

- **TIMER_A_CAPTURECOMPARE_REGISTER_0**
- **TIMER_A_CAPTURECOMPARE_REGISTER_1**

- `TIMER_A_CAPTURECOMPARE_REGISTER_2`
- `TIMER_A_CAPTURECOMPARE_REGISTER_3`
- `TIMER_A_CAPTURECOMPARE_REGISTER_4`
- `TIMER_A_CAPTURECOMPARE_REGISTER_5`
- `TIMER_A_CAPTURECOMPARE_REGISTER_6`

Returns:

One of the following:

- `TIMER_A_OUTPUTMODE_OUTBITVALUE_HIGH`
- `TIMER_A_OUTPUTMODE_OUTBITVALUE_LOW`

28.2.2.17 `uint8_t TIMER_A_getSynchronizedCaptureCompareInput (uint16_t baseAddress, uint16_t captureCompareRegister, uint16_t synchronized)`

Get synchronized capturecompare input.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	36
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	16
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:

baseAddress is the base address of the `TIMER_A` module.

captureCompareRegister Valid values are:

- `TIMER_A_CAPTURECOMPARE_REGISTER_0`
- `TIMER_A_CAPTURECOMPARE_REGISTER_1`
- `TIMER_A_CAPTURECOMPARE_REGISTER_2`
- `TIMER_A_CAPTURECOMPARE_REGISTER_3`
- `TIMER_A_CAPTURECOMPARE_REGISTER_4`
- `TIMER_A_CAPTURECOMPARE_REGISTER_5`
- `TIMER_A_CAPTURECOMPARE_REGISTER_6`

synchronized Valid values are:

- `TIMER_A_READ_SYNCHRONIZED_CAPTURECOMPAREINPUT`
- `TIMER_A_READ_CAPTURE_COMPARE_INPUT`

Returns:

One of the following:

- `TIMER_A_CAPTURECOMPARE_INPUT_HIGH`
- `TIMER_A_CAPTURECOMPARE_INPUT_LOW`

28.2.2.18 `void TIMER_A_initCapture (uint16_t baseAddress, uint16_t captureRegister, uint16_t captureMode, uint16_t captureInputSelect, uint16_t synchronizeCaptureSource, uint16_t captureInterruptEnable, uint16_t captureOutputMode)`

DEPRECATED - Initializes Capture Mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	88
TI Compiler 4.2.1	Size	50
TI Compiler 4.2.1	Speed	50
IAR 5.51.6	None	80
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	36

Parameters:

baseAddress is the base address of the TIMER_A module.

captureRegister selects the Capture register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_A_CAPTURECOMPARE_REGISTER_0**
- **TIMER_A_CAPTURECOMPARE_REGISTER_1**
- **TIMER_A_CAPTURECOMPARE_REGISTER_2**
- **TIMER_A_CAPTURECOMPARE_REGISTER_3**
- **TIMER_A_CAPTURECOMPARE_REGISTER_4**
- **TIMER_A_CAPTURECOMPARE_REGISTER_5**
- **TIMER_A_CAPTURECOMPARE_REGISTER_6**

captureMode is the capture mode selected. Valid values are:

- **TIMER_A_CAPTUREMODE_NO_CAPTURE** [Default]
- **TIMER_A_CAPTUREMODE_RISING_EDGE**
- **TIMER_A_CAPTUREMODE_FALLING_EDGE**
- **TIMER_A_CAPTUREMODE_RISING_AND_FALLING_EDGE**

captureInputSelect decides the Input Select Valid values are:

- **TIMER_A_CAPTURE_INPUTSELECT_CC1xA**
- **TIMER_A_CAPTURE_INPUTSELECT_CC1xB**
- **TIMER_A_CAPTURE_INPUTSELECT_GND**
- **TIMER_A_CAPTURE_INPUTSELECT_Vcc**

synchronizeCaptureSource decides if capture source should be synchronized with timer clock Valid values are:

- **TIMER_A_CAPTURE_ASYNCHRONOUS** [Default]
- **TIMER_A_CAPTURE_SYNCHRONOUS**

captureInterruptEnable is to enable or disable timer captureCompare interrupt. Valid values are:

- **TIMER_A_CAPTURECOMPARE_INTERRUPT_DISABLE** [Default]
- **TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE**

captureOutputMode specifies the output mode. Valid values are:

- **TIMER_A_OUTPUTMODE_OUTBITVALUE** [Default]
- **TIMER_A_OUTPUTMODE_SET**
- **TIMER_A_OUTPUTMODE_TOGGLE_RESET**
- **TIMER_A_OUTPUTMODE_SET_RESET**
- **TIMER_A_OUTPUTMODE_TOGGLE**
- **TIMER_A_OUTPUTMODE_RESET**
- **TIMER_A_OUTPUTMODE_TOGGLE_SET**
- **TIMER_A_OUTPUTMODE_RESET_SET**

Modified bits of **TAxCCTLn** register.

Returns:

None

28.2.2.19 void TIMER_A_initCaptureMode (uint16_t *baseAddress*,
TIMER_A_initCaptureModeParam * *param*)

Initializes Capture Mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	70
TI Compiler 4.2.1	Size	48
TI Compiler 4.2.1	Speed	50
IAR 5.51.6	None	50
IAR 5.51.6	Size	48
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	128
MSPGCC 4.8.0	Size	48
MSPGCC 4.8.0	Speed	48

Parameters:

baseAddress is the base address of the TIMER_A module.

param is the pointer to struct for capture mode initialization.

Modified bits of **TAxCTLn** register.

Returns:

None

28.2.2.20 void TIMER_A_initCompare (uint16_t *baseAddress*, uint16_t *compareRegister*,
uint16_t *compareInterruptEnable*, uint16_t *compareOutputMode*, uint16_t
compareValue)

DEPRECATED - Initializes Compare Mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	82
TI Compiler 4.2.1	Size	30
TI Compiler 4.2.1	Speed	30
IAR 5.51.6	None	64
IAR 5.51.6	Size	54
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	28

Parameters:

baseAddress is the base address of the TIMER_A module.

compareRegister selects the Capture register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_A_CAPTURECOMPARE_REGISTER_0**
- **TIMER_A_CAPTURECOMPARE_REGISTER_1**
- **TIMER_A_CAPTURECOMPARE_REGISTER_2**
- **TIMER_A_CAPTURECOMPARE_REGISTER_3**
- **TIMER_A_CAPTURECOMPARE_REGISTER_4**

- **TIMER_A_CAPTURECOMPARE_REGISTER_5**
- **TIMER_A_CAPTURECOMPARE_REGISTER_6**

compareInterruptEnable is to enable or disable timer captureCompare interrupt. Valid values are:

- **TIMER_A_CAPTURECOMPARE_INTERRUPT_DISABLE** [Default]
- **TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE**

compareOutputMode specifies the output mode. Valid values are:

- **TIMER_A_OUTPUTMODE_OUTBITVALUE** [Default]
- **TIMER_A_OUTPUTMODE_SET**
- **TIMER_A_OUTPUTMODE_TOGGLE_RESET**
- **TIMER_A_OUTPUTMODE_SET_RESET**
- **TIMER_A_OUTPUTMODE_TOGGLE**
- **TIMER_A_OUTPUTMODE_RESET**
- **TIMER_A_OUTPUTMODE_TOGGLE_SET**
- **TIMER_A_OUTPUTMODE_RESET_SET**

compareValue is the count to be compared with in compare mode

Modified bits of **TAxCCRn** register and bits of **TAxCCTLn** register.

Returns:

None

28.2.2.21 void **TIMER_A_initCompareMode** (uint16_t *baseAddress*,
TIMER_A_initCompareModeParam * *param*)

Initializes Compare Mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	72
TI Compiler 4.2.1	Size	46
TI Compiler 4.2.1	Speed	46
IAR 5.51.6	None	52
IAR 5.51.6	Size	50
IAR 5.51.6	Speed	50
MSPGCC 4.8.0	None	126
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

Parameters:

baseAddress is the base address of the **TIMER_A** module.

param is the pointer to struct for compare mode initialization.

Modified bits of **TAxCCRn** register and bits of **TAxCCTLn** register.

Returns:

None

28.2.2.22 void **TIMER_A_initContinuousMode** (uint16_t *baseAddress*,
TIMER_A_initContinuousModeParam * *param*)

Configures **TIMER_A** in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	106
TI Compiler 4.2.1	Size	70
TI Compiler 4.2.1	Speed	70
IAR 5.51.6	None	78
IAR 5.51.6	Size	70
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	170
MSPGCC 4.8.0	Size	82
MSPGCC 4.8.0	Speed	82

Parameters:

baseAddress is the base address of the TIMER_A module.

param is the pointer to struct for continuous mode initialization.

Modified bits of **TAxCTL** register.

Returns:

None

28.2.2.23 void TIMER_A_initUpDownMode (uint16_t *baseAddress*,
TIMER_A_initUpDownModeParam * *param*)

Configures TIMER_A in up down mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	152
TI Compiler 4.2.1	Size	98
TI Compiler 4.2.1	Speed	98
IAR 5.51.6	None	118
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	72
MSPGCC 4.8.0	None	248
MSPGCC 4.8.0	Size	116
MSPGCC 4.8.0	Speed	118

Parameters:

baseAddress is the base address of the TIMER_A module.

param is the pointer to struct for up-down mode initialization.

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register and bits of **TAxCCR0** register.

Returns:

None

28.2.2.24 void TIMER_A_initUpMode (uint16_t *baseAddress*, TIMER_A_initUpModeParam
* *param*)

Configures TIMER_A in up mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	152
TI Compiler 4.2.1	Size	98
TI Compiler 4.2.1	Speed	98
IAR 5.51.6	None	120
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	74
MSPGCC 4.8.0	None	248
MSPGCC 4.8.0	Size	116
MSPGCC 4.8.0	Speed	118

Parameters:

baseAddress is the base address of the TIMER_A module.

param is the pointer to struct for up mode initialization.

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register and bits of **TAxCCR0** register.

Returns:

None

28.2.2.25 void TIMER_A_outputPWM (uint16_t *baseAddress*, TIMER_A_outputPWMPParam * *param*)

Generate a PWM with timer running in up mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	136
TI Compiler 4.2.1	Size	86
TI Compiler 4.2.1	Speed	86
IAR 5.51.6	None	108
IAR 5.51.6	Size	100
IAR 5.51.6	Speed	100
MSPGCC 4.8.0	None	228
MSPGCC 4.8.0	Size	100
MSPGCC 4.8.0	Speed	100

Parameters:

baseAddress is the base address of the TIMER_A module.

param is the pointer to struct for PWM configuration.

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register, bits of **TAxCCR0** register and bits of **TAxCTLn** register.

Returns:

None

28.2.2.26 void TIMER_A_setCompareValue (uint16_t *baseAddress*, uint16_t *compareRegister*, uint16_t *compareValue*)

Sets the value of the capture-compare register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the TIMER_A module.

compareRegister selects the Capture register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_A_CAPTURECOMPARE_REGISTER_0**
- **TIMER_A_CAPTURECOMPARE_REGISTER_1**
- **TIMER_A_CAPTURECOMPARE_REGISTER_2**
- **TIMER_A_CAPTURECOMPARE_REGISTER_3**
- **TIMER_A_CAPTURECOMPARE_REGISTER_4**
- **TIMER_A_CAPTURECOMPARE_REGISTER_5**
- **TIMER_A_CAPTURECOMPARE_REGISTER_6**

compareValue is the count to be compared with in compare mode

Modified bits of **TAxCCRn** register.

Returns:

None

28.2.2.27 void **TIMER_A_setOutputForOutputModeOutBitValue** (uint16_t *baseAddress*, uint16_t *captureCompareRegister*, uint8_t *outputModeOutBitValue*)

Set output bit for output mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	42
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	74
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

baseAddress is the base address of the TIMER_A module.

captureCompareRegister Valid values are:

- **TIMER_A_CAPTURECOMPARE_REGISTER_0**
- **TIMER_A_CAPTURECOMPARE_REGISTER_1**
- **TIMER_A_CAPTURECOMPARE_REGISTER_2**
- **TIMER_A_CAPTURECOMPARE_REGISTER_3**
- **TIMER_A_CAPTURECOMPARE_REGISTER_4**

- **TIMER_A_CAPTURECOMPARE_REGISTER_5**
- **TIMER_A_CAPTURECOMPARE_REGISTER_6**

outputModeOutBitValue is the value to be set for out bit Valid values are:

- **TIMER_A_OUTPUTMODE_OUTBITVALUE_HIGH**
- **TIMER_A_OUTPUTMODE_OUTBITVALUE_LOW**

Modified bits of **TAxCCTLn** register.

Returns:
None

28.2.2.28 void **TIMER_A_startContinuousMode** (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerInterruptEnable_TAIE*, uint16_t *timerClear*)

DEPRECATED - Spelling Error Fixed. Starts timer in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	52
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	34
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	52
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	58

Parameters:

baseAddress is the base address of the **TIMER_A** module.

clockSource selects Clock source. Valid values are:

- **TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_A_CLOCKSOURCE_ACLK**
- **TIMER_A_CLOCKSOURCE_SMCLK**
- **TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the desired divider for the clock source Valid values are:

- **TIMER_A_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_A_CLOCKSOURCE_DIVIDER_2**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_10**
- **TIMER_A_CLOCKSOURCE_DIVIDER_12**
- **TIMER_A_CLOCKSOURCE_DIVIDER_14**
- **TIMER_A_CLOCKSOURCE_DIVIDER_16**
- **TIMER_A_CLOCKSOURCE_DIVIDER_20**
- **TIMER_A_CLOCKSOURCE_DIVIDER_24**
- **TIMER_A_CLOCKSOURCE_DIVIDER_28**
- **TIMER_A_CLOCKSOURCE_DIVIDER_32**
- **TIMER_A_CLOCKSOURCE_DIVIDER_40**

- **TIMER_A_CLOCKSOURCE_DIVIDER_48**
- **TIMER_A_CLOCKSOURCE_DIVIDER_56**
- **TIMER_A_CLOCKSOURCE_DIVIDER_64**

timerInterruptEnable_TAIE is to enable or disable timer interrupt Valid values are:

- **TIMER_A_TAIE_INTERRUPT_ENABLE**
- **TIMER_A_TAIE_INTERRUPT_DISABLE** [Default]

timerClear decides if timer clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_A_DO_CLEAR**
- **TIMER_A_SKIP_CLEAR** [Default]

Modified bits of **TAxCTL** register.

Returns:

None

28.2.2.29 void **TIMER_A_startContinuousMode** (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerInterruptEnable_TAIE*, uint16_t *timerClear*)

DEPRECATED - Starts timer in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	80
TI Compiler 4.2.1	Size	38
TI Compiler 4.2.1	Speed	38
IAR 5.51.6	None	68
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	64
MSPGCC 4.8.0	Speed	58

Parameters:

baseAddress is the base address of the **TIMER_A** module.

clockSource selects Clock source. Valid values are:

- **TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_A_CLOCKSOURCE_ACLK**
- **TIMER_A_CLOCKSOURCE_SMCLK**
- **TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the desired divider for the clock source Valid values are:

- **TIMER_A_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_A_CLOCKSOURCE_DIVIDER_2**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_10**
- **TIMER_A_CLOCKSOURCE_DIVIDER_12**
- **TIMER_A_CLOCKSOURCE_DIVIDER_14**
- **TIMER_A_CLOCKSOURCE_DIVIDER_16**
- **TIMER_A_CLOCKSOURCE_DIVIDER_20**

- **TIMER_A_CLOCKSOURCE_DIVIDER_24**
- **TIMER_A_CLOCKSOURCE_DIVIDER_28**
- **TIMER_A_CLOCKSOURCE_DIVIDER_32**
- **TIMER_A_CLOCKSOURCE_DIVIDER_40**
- **TIMER_A_CLOCKSOURCE_DIVIDER_48**
- **TIMER_A_CLOCKSOURCE_DIVIDER_56**
- **TIMER_A_CLOCKSOURCE_DIVIDER_64**

timerInterruptEnable_TAIE is to enable or disable timer interrupt Valid values are:

- **TIMER_A_TAIE_INTERRUPT_ENABLE**
- **TIMER_A_TAIE_INTERRUPT_DISABLE** [Default]

timerClear decides if timer clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_A_DO_CLEAR**
- **TIMER_A_SKIP_CLEAR** [Default]

Modified bits of **TAxCTL** register.

Returns:

None

28.2.2.30 void TIMER_A_startCounter (uint16_t *baseAddress*, uint16_t *timerMode*)

Starts **TIMER_A** counter.

This function assumes that the timer has been previously configured using **TIMER_A_configureContinuousMode**, **TIMER_A_configureUpMode** or **TIMER_A_configureUpDownMode**.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	30
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the **TIMER_A** module.

timerMode mode to put the timer in Valid values are:

- **TIMER_A_STOP_MODE**
- **TIMER_A_UP_MODE**
- **TIMER_A_CONTINUOUS_MODE** [Default]
- **TIMER_A_UPDOWN_MODE**

Modified bits of **TAxCTL** register.

Returns:

None

28.2.2.31 void TIMER_A_startUpDownMode (uint16_t *baseAddress*,
uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t
timerPeriod, uint16_t *timerInterruptEnable_TAIE*, uint16_t
captureCompareInterruptEnable_CCR0_CCIE, uint16_t *timerClear*)

DEPRECATED - Replaced by TIMER_A_configureUpMode and TIMER_A_startCounter API. Starts timer in up down mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	92
TI Compiler 4.2.1	Size	54
TI Compiler 4.2.1	Speed	54
IAR 5.51.6	None	84
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	76
MSPGCC 4.8.0	Speed	94

Parameters:

baseAddress is the base address of the TIMER_A module.

clockSource selects Clock source. Valid values are:

- **TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_A_CLOCKSOURCE_ACLK**
- **TIMER_A_CLOCKSOURCE_SMCLK**
- **TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the desired divider for the clock source Valid values are:

- **TIMER_A_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_A_CLOCKSOURCE_DIVIDER_2**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_10**
- **TIMER_A_CLOCKSOURCE_DIVIDER_12**
- **TIMER_A_CLOCKSOURCE_DIVIDER_14**
- **TIMER_A_CLOCKSOURCE_DIVIDER_16**
- **TIMER_A_CLOCKSOURCE_DIVIDER_20**
- **TIMER_A_CLOCKSOURCE_DIVIDER_24**
- **TIMER_A_CLOCKSOURCE_DIVIDER_28**
- **TIMER_A_CLOCKSOURCE_DIVIDER_32**
- **TIMER_A_CLOCKSOURCE_DIVIDER_40**
- **TIMER_A_CLOCKSOURCE_DIVIDER_48**
- **TIMER_A_CLOCKSOURCE_DIVIDER_56**
- **TIMER_A_CLOCKSOURCE_DIVIDER_64**

timerPeriod is the specified timer period

timerInterruptEnable_TAIE is to enable or disable timer interrupt Valid values are:

- **TIMER_A_TAIE_INTERRUPT_ENABLE**
- **TIMER_A_TAIE_INTERRUPT_DISABLE** [Default]

captureCompareInterruptEnable_CCR0_CCIE is to enable or disable timer CCR0 captureCompare interrupt. Valid values are:

- **TIMER_A_CCIE_CCR0_INTERRUPT_ENABLE**

- **TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE** [Default]

timerClear decides if timer clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_A_DO_CLEAR**
- **TIMER_A_SKIP_CLEAR** [Default]

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register and bits of **TAxCCR0** register.

Returns:
None

28.2.2.32 void **TIMER_A_startUpMode** (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerPeriod*, uint16_t *timerInterruptEnable_TAIE*, uint16_t *captureCompareInterruptEnable_CCR0_CCIE*, uint16_t *timerClear*)

DEPRECATED - Replaced by **TIMER_A_configureUpMode** and **TIMER_A_startCounter** API. Starts timer in up mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	92
TI Compiler 4.2.1	Size	54
TI Compiler 4.2.1	Speed	54
IAR 5.51.6	None	84
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	62
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	76
MSPGCC 4.8.0	Speed	94

Parameters:

baseAddress is the base address of the **TIMER_A** module.

clockSource selects Clock source. Valid values are:

- **TIMER_A_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_A_CLOCKSOURCE_ACLK**
- **TIMER_A_CLOCKSOURCE_SMCLK**
- **TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the desired divider for the clock source Valid values are:

- **TIMER_A_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_A_CLOCKSOURCE_DIVIDER_2**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_10**
- **TIMER_A_CLOCKSOURCE_DIVIDER_12**
- **TIMER_A_CLOCKSOURCE_DIVIDER_14**
- **TIMER_A_CLOCKSOURCE_DIVIDER_16**
- **TIMER_A_CLOCKSOURCE_DIVIDER_20**
- **TIMER_A_CLOCKSOURCE_DIVIDER_24**
- **TIMER_A_CLOCKSOURCE_DIVIDER_28**
- **TIMER_A_CLOCKSOURCE_DIVIDER_32**
- **TIMER_A_CLOCKSOURCE_DIVIDER_40**

- **TIMER_A_CLOCKSOURCE_DIVIDER_48**
- **TIMER_A_CLOCKSOURCE_DIVIDER_56**
- **TIMER_A_CLOCKSOURCE_DIVIDER_64**

timerPeriod is the specified timer period. This is the value that gets written into the CCR0. Limited to 16 bits[uint16_t]

timerInterruptEnable_TAIE is to enable or disable timer interrupt Valid values are:

- **TIMER_A_TAIE_INTERRUPT_ENABLE**
- **TIMER_A_TAIE_INTERRUPT_DISABLE** [Default]

captureCompareInterruptEnable_CCR0_CCIE is to enable or disable timer CCR0 captureCompare interrupt. Valid values are:

- **TIMER_A_CCIE_CCR0_INTERRUPT_ENABLE**
- **TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE** [Default]

timerClear decides if timer clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_A_DO_CLEAR**
- **TIMER_A_SKIP_CLEAR** [Default]

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register and bits of **TAxCCR0** register.

Returns:

None

28.2.2.33 void TIMER_A_stop (uint16_t baseAddress)

Stops the timer.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the TIMER_A module.

Modified bits of **TAxCTL** register.

Returns:

None

28.3 Programming Example

The following example shows some TIMER_A operations using the APIs

```
{
    //Start TIMER_A
    TIMER_A_configureUpDownMode( TIMER_A1_BASE,
    TIMER_A_CLOCKSOURCE_SMCLK,
    TIMER_A_CLOCKSOURCE_DIVIDER_1,
    TIMER_PERIOD,
    TIMER_A_TAIE_INTERRUPT_DISABLE,
```

```
    TIMER_A_CCIE_CCR0_INTERRUPT_DISABLE,  
    TIMER_A_DO_CLEAR  
);  
  
TIMER_A_startCounter( TIMER_A1_BASE,  
    TIMER_A_UPDOWN_MODE  
);  
  
//Initialize compare registers to generate PWM1  
TIMER_A_initCompare(TIMER_A1_BASE,  
    TIMER_A_CAPTURECOMPARE_REGISTER_1,  
    TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE,  
    TIMER_A_OUTPUTMODE_TOGGLE_SET,  
    DUTY_CYCLE1  
);  
//Initialize compare registers to generate PWM2  
TIMER_A_initCompare(TIMER_A1_BASE,  
    TIMER_A_CAPTURECOMPARE_REGISTER_2,  
    TIMER_A_CAPTURECOMPARE_INTERRUPT_DISABLE,  
    TIMER_A_OUTPUTMODE_TOGGLE_SET,  
    DUTY_CYCLE2  
);  
  
//Enter LPM0  
__bis_SR_register(LPM0_bits);  
  
//For debugger  
__no_operation();  
}
```

29 16-Bit Timer_B (TIMER_B)

Introduction	339
API Functions	340
Programming Example	367

29.1 Introduction

TIMER_B is a 16-bit timer/counter with multiple capture/compare registers. TIMER_B can support multiple capture/compares, PWM outputs, and interval timing. TIMER_B also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions and from each of the capture/compare registers.

This peripheral API handles Timer B hardware peripheral.

TIMER_B features include:

- Asynchronous 16-bit timer/counter with four operating modes
- Selectable and configurable clock source
- Up to seven configurable capture/compare registers
- Configurable outputs with pulse width modulation (PWM) capability
- Asynchronous input and output latching
- Interrupt vector register for fast decoding of all Timer_B interrupts

Differences From Timer_A Timer_B is identical to Timer_A with the following exceptions:

- The length of Timer_B is programmable to be 8, 10, 12, or 16 bits
- Timer_B TBxCCRn registers are double-buffered and can be grouped
- All Timer_B outputs can be put into a high-impedance state
- The SCCI bit function is not implemented in Timer_B

TIMER_B can operate in 3 modes

- Continuous Mode
- Up Mode
- Down Mode

TIMER_B Interrupts may be generated on counter overflow conditions and during capture compare events.

The TIMER_B may also be used to generate PWM outputs. PWM outputs can be generated by initializing the compare mode with [TIMER_B_initCompare\(\)](#) and the necessary parameters. The PWM may be customized by selecting a desired timer mode (continuous/up/upDown), duty cycle, output mode, timer period etc. The library also provides a simpler way to generate PWM using [TIMER_B_generatePWM\(\)](#) API. However the level of customization and the kinds of PWM generated are limited in this API. Depending on how complex the PWM is and what level of customization is required, the user can use [TIMER_B_generatePWM\(\)](#) or a combination of [Timer_initCompare\(\)](#) and timer start APIs

The TIMER_B API provides a set of functions for dealing with the TIMER_B module. Functions are provided to configure and control the timer, along with functions to modify timer/counter values, and to manage interrupt handling for the timer.

Control is also provided over interrupt sources and events. Interrupts can be generated to indicate that an event has been captured.

This driver is contained in `TIMER_B.c`, with `TIMER_B.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	2094
TI Compiler 4.2.1	Size	1110
TI Compiler 4.2.1	Speed	1112
IAR 5.51.6	None	1544
IAR 5.51.6	Size	674
IAR 5.51.6	Speed	1110
MSPGCC 4.8.0	None	2884
MSPGCC 4.8.0	Size	1350
MSPGCC 4.8.0	Speed	1418

29.2 API Functions

Functions

- void [TIMER_B_clear](#) (uint16_t baseAddress)
- void [TIMER_B_clearCaptureCompareInterruptFlag](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- void [TIMER_B_clearTimerInterruptFlag](#) (uint16_t baseAddress)
- void [TIMER_B_configureContinuousMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TBIE, uint16_t timerClear)
- void [TIMER_B_configureUpDownMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t timerInterruptEnable_TBIE, uint16_t captureCompareInterruptEnable_CCR0_CCIE, uint16_t timerClear)
- void [TIMER_B_configureUpMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t timerInterruptEnable_TBIE, uint16_t captureCompareInterruptEnable_CCR0_CCIE, uint16_t timerClear)
- void [TIMER_B_disableCaptureCompareInterrupt](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- void [TIMER_B_disableInterrupt](#) (uint16_t baseAddress)
- void [TIMER_B_enableCaptureCompareInterrupt](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- void [TIMER_B_enableInterrupt](#) (uint16_t baseAddress)
- void [TIMER_B_generatePWM](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t compareRegister, uint16_t compareOutputMode, uint16_t dutyCycle)
- uint16_t [TIMER_B_getCaptureCompareCount](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- uint32_t [TIMER_B_getCaptureCompareInterruptStatus](#) (uint16_t baseAddress, uint16_t captureCompareRegister, uint16_t mask)
- uint16_t [TIMER_B_getCounterValue](#) (uint16_t baseAddress)
- uint32_t [TIMER_B_getInterruptStatus](#) (uint16_t baseAddress)
- uint8_t [TIMER_B_getOutputForOutputModeOutBitValue](#) (uint16_t baseAddress, uint16_t captureCompareRegister)
- uint8_t [TIMER_B_getSynchronizedCaptureCompareInput](#) (uint16_t baseAddress, uint16_t captureCompareRegister, uint16_t synchronized)
- void [TIMER_B_initCapture](#) (uint16_t baseAddress, uint16_t captureRegister, uint16_t captureMode, uint16_t captureInputSelect, uint16_t synchronizeCaptureSource, uint16_t captureInterruptEnable, uint16_t captureOutputMode)
- void [TIMER_B_initCaptureMode](#) (uint16_t baseAddress, TIMER_B_initCaptureModeParam *param)
- void [TIMER_B_initCompare](#) (uint16_t baseAddress, uint16_t compareRegister, uint16_t compareInterruptEnable, uint16_t compareOutputMode, uint16_t compareValue)
- void [TIMER_B_initCompareLatchLoadEvent](#) (uint16_t baseAddress, uint16_t compareRegister, uint16_t compareLatchLoadEvent)
- void [TIMER_B_initCompareMode](#) (uint16_t baseAddress, TIMER_B_initCompareModeParam *param)
- void [TIMER_B_initContinuousMode](#) (uint16_t baseAddress, TIMER_B_initContinuousModeParam *param)
- void [TIMER_B_initUpDownMode](#) (uint16_t baseAddress, TIMER_B_initUpDownModeParam *param)
- void [TIMER_B_initUpMode](#) (uint16_t baseAddress, TIMER_B_initUpModeParam *param)
- void [TIMER_B_outputPWM](#) (uint16_t baseAddress, TIMER_B_outputPWMParm *param)
- void [TIMER_B_selectCounterLength](#) (uint16_t baseAddress, uint16_t counterLength)
- void [TIMER_B_selectLatchingGroup](#) (uint16_t baseAddress, uint16_t groupLatch)
- void [TIMER_B_setCompareValue](#) (uint16_t baseAddress, uint16_t compareRegister, uint16_t compareValue)

- void [TIMER_B_setOutputForOutputModeOutBitValue](#) (uint16_t baseAddress, uint16_t captureCompareRegister, uint8_t outputModeOutBitValue)
- void [TIMER_B_startContinuousMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TBIE, uint16_t timerClear)
- void [TIMER_B_startContinuousMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TBIE, uint16_t timerClear)
- void [TIMER_B_startCounter](#) (uint16_t baseAddress, uint16_t timerMode)
- void [TIMER_B_startUpDownMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t timerInterruptEnable_TBIE, uint16_t captureCompareInterruptEnable_CCR0_CCIE, uint16_t timerClear)
- void [TIMER_B_startUpMode](#) (uint16_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerPeriod, uint16_t timerInterruptEnable_TBIE, uint16_t captureCompareInterruptEnable_CCR0_CCIE, uint16_t timerClear)
- void [TIMER_B_stop](#) (uint16_t baseAddress)

29.2.1 Detailed Description

The TIMER_B API is broken into three groups of functions: those that deal with timer configuration and control, those that deal with timer contents, and those that deal with interrupt handling.

TIMER_B configuration and initialization is handled by

- [TIMER_B_startCounter\(\)](#)
- [TIMER_B_configureContinuousMode\(\)](#)
- [TIMER_B_configureUpMode\(\)](#)
- [TIMER_B_configureUpDownMode\(\)](#)
- [TIMER_B_startContinuousMode\(\)](#)
- [TIMER_B_startUpMode\(\)](#)
- [TIMER_B_startUpDownMode\(\)](#)
- [TIMER_B_initCapture\(\)](#)
- [TIMER_B_initCompare\(\)](#)
- [TIMER_B_clear\(\)](#)
- [TIMER_B_stop\(\)](#)
- [TIMER_B_initCompareLatchLoadEvent\(\)](#)
- [TIMER_B_selectLatchingGroup\(\)](#)
- [TIMER_B_selectCounterLength\(\)](#)

TIMER_B outputs are handled by

- [TIMER_B_getSynchronizedCaptureCompareInput\(\)](#)
- [TIMER_B_getOutputForOutputModeOutBitValue\(\)](#)
- [TIMER_B_setOutputForOutputModeOutBitValue\(\)](#)
- [TIMER_B_generatePWM\(\)](#)
- [TIMER_B_getCaptureCompareCount\(\)](#)
- [TIMER_B_setCompareValue\(\)](#)
- [TIMER_B_getCounterValue\(\)](#)

The interrupt handler for the TIMER_B interrupt is managed with

- [TIMER_B_enableInterrupt\(\)](#)
- [TIMER_B_disableInterrupt\(\)](#)
- [TIMER_B_getInterruptStatus\(\)](#)
- [TIMER_B_enableCaptureCompareInterrupt\(\)](#)
- [TIMER_B_disableCaptureCompareInterrupt\(\)](#)
- [TIMER_B_getCaptureCompareInterruptStatus\(\)](#)
- [TIMER_B_clearCaptureCompareInterruptFlag\(\)](#)
- [TIMER_B_clearTimerInterruptFlag\(\)](#)

29.2.2 Function Documentation

29.2.2.1 void TIMER_B_clear (uint16_t *baseAddress*)

Reset/Clear the TIMER_B clock divider, count direction, count.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the TIMER_B module.

Modified bits of **TBxCTL** register.

Returns:

None

29.2.2.2 void TIMER_B_clearCaptureCompareInterruptFlag (uint16_t *baseAddress*, uint16_t *captureCompareRegister*)

Clears the capture-compare interrupt flag.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the TIMER_B module.

captureCompareRegister selects the capture compare register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

Modified bits are **CCIFG** of **TBxCCTLn** register.

Returns:
None

29.2.2.3 void TIMER_B_clearTimerInterruptFlag (uint16_t *baseAddress*)

Clears the TIMER_B TBIFG interrupt flag.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the TIMER_B module.

Modified bits are **TBIFG** of **TBxCTL** register.

Returns:
None

29.2.2.4 void TIMER_B_configureContinuousMode (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerInterruptEnable_TBIE*, uint16_t *timerClear*)

DEPRECATED - Configures TIMER_B in continuous mode.

This API does not start the timer. Timer needs to be started when required using the TIMER_B_startCounter API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	80
TI Compiler 4.2.1	Size	38
TI Compiler 4.2.1	Speed	38
IAR 5.51.6	None	68
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	60
MSPGCC 4.8.0	Speed	52

Parameters:

baseAddress is the base address of the TIMER_B module.

clockSource selects the clock source Valid values are:

■ **TIMER_B_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]

- **TIMER_B_CLOCKSOURCE_ACLK**
- **TIMER_B_CLOCKSOURCE_SMCLK**
- **TIMER_B_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the divider for Clock source. Valid values are:

- **TIMER_B_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_B_CLOCKSOURCE_DIVIDER_2**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_10**
- **TIMER_B_CLOCKSOURCE_DIVIDER_12**
- **TIMER_B_CLOCKSOURCE_DIVIDER_14**
- **TIMER_B_CLOCKSOURCE_DIVIDER_16**
- **TIMER_B_CLOCKSOURCE_DIVIDER_20**
- **TIMER_B_CLOCKSOURCE_DIVIDER_24**
- **TIMER_B_CLOCKSOURCE_DIVIDER_28**
- **TIMER_B_CLOCKSOURCE_DIVIDER_32**
- **TIMER_B_CLOCKSOURCE_DIVIDER_40**
- **TIMER_B_CLOCKSOURCE_DIVIDER_48**
- **TIMER_B_CLOCKSOURCE_DIVIDER_56**
- **TIMER_B_CLOCKSOURCE_DIVIDER_64**

timerInterruptEnable_TBIE is to enable or disable TIMER_B interrupt Valid values are:

- **TIMER_B_TBIE_INTERRUPT_ENABLE**
- **TIMER_B_TBIE_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_B clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_B_DO_CLEAR**
- **TIMER_B_SKIP_CLEAR** [Default]

Modified bits of **TBxCTL** register.

Returns:

None

29.2.2.5 void **TIMER_B_configureUpDownMode** (uint16_t *baseAddress*,
uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t
timerPeriod, uint16_t *timerInterruptEnable_TBIE*, uint16_t
captureCompareInterruptEnable_CCR0_CCIE, uint16_t *timerClear*)

DEPRECATED - Configures TIMER_B in up down mode.

This API does not start the timer. Timer needs to be started when required using the **TIMER_B_startCounter** API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	92
TI Compiler 4.2.1	Size	54
TI Compiler 4.2.1	Speed	54
IAR 5.51.6	None	84
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	88

Parameters:

baseAddress is the base address of the TIMER_B module.

clockSource selects the clock source Valid values are:

- **TIMER_B_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_B_CLOCKSOURCE_ACLK**
- **TIMER_B_CLOCKSOURCE_SMCLK**
- **TIMER_B_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the divider for Clock source. Valid values are:

- **TIMER_B_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_B_CLOCKSOURCE_DIVIDER_2**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_10**
- **TIMER_B_CLOCKSOURCE_DIVIDER_12**
- **TIMER_B_CLOCKSOURCE_DIVIDER_14**
- **TIMER_B_CLOCKSOURCE_DIVIDER_16**
- **TIMER_B_CLOCKSOURCE_DIVIDER_20**
- **TIMER_B_CLOCKSOURCE_DIVIDER_24**
- **TIMER_B_CLOCKSOURCE_DIVIDER_28**
- **TIMER_B_CLOCKSOURCE_DIVIDER_32**
- **TIMER_B_CLOCKSOURCE_DIVIDER_40**
- **TIMER_B_CLOCKSOURCE_DIVIDER_48**
- **TIMER_B_CLOCKSOURCE_DIVIDER_56**
- **TIMER_B_CLOCKSOURCE_DIVIDER_64**

timerPeriod is the specified TIMER_B period

timerInterruptEnable_TBIE is to enable or disable TIMER_B interrupt Valid values are:

- **TIMER_B_TBIE_INTERRUPT_ENABLE**
- **TIMER_B_TBIE_INTERRUPT_DISABLE** [Default]

captureCompareInterruptEnable_CCR0_CCIE is to enable or disable TIMER_B CCR0 capture compare interrupt.

Valid values are:

- **TIMER_B_CCIE_CCR0_INTERRUPT_ENABLE**
- **TIMER_B_CCIE_CCR0_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_B clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_B_DO_CLEAR**
- **TIMER_B_SKIP_CLEAR** [Default]

Modified bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:

None

29.2.2.6 void **TIMER_B_configureUpMode** (uint16_t *baseAddress*,
uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t
timerPeriod, uint16_t *timerInterruptEnable_TBIE*, uint16_t
captureCompareInterruptEnable_CCR0_CCIE, uint16_t *timerClear*)

DEPRECATED - Configures TIMER_B in up mode.

This API does not start the timer. Timer needs to be started when required using the **TIMER_B_startCounter** API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	92
TI Compiler 4.2.1	Size	54
TI Compiler 4.2.1	Speed	54
IAR 5.51.6	None	84
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	88

Parameters:

baseAddress is the base address of the TIMER_B module.

clockSource selects the clock source Valid values are:

- **TIMER_B_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_B_CLOCKSOURCE_ACLK**
- **TIMER_B_CLOCKSOURCE_SMCLK**
- **TIMER_B_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the divider for Clock source. Valid values are:

- **TIMER_B_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_B_CLOCKSOURCE_DIVIDER_2**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_10**
- **TIMER_B_CLOCKSOURCE_DIVIDER_12**
- **TIMER_B_CLOCKSOURCE_DIVIDER_14**
- **TIMER_B_CLOCKSOURCE_DIVIDER_16**
- **TIMER_B_CLOCKSOURCE_DIVIDER_20**
- **TIMER_B_CLOCKSOURCE_DIVIDER_24**
- **TIMER_B_CLOCKSOURCE_DIVIDER_28**
- **TIMER_B_CLOCKSOURCE_DIVIDER_32**
- **TIMER_B_CLOCKSOURCE_DIVIDER_40**
- **TIMER_B_CLOCKSOURCE_DIVIDER_48**
- **TIMER_B_CLOCKSOURCE_DIVIDER_56**
- **TIMER_B_CLOCKSOURCE_DIVIDER_64**

timerPeriod is the specified TIMER_B period. This is the value that gets written into the CCR0. Limited to 16 bits[uint16_t]

timerInterruptEnable_TBIE is to enable or disable TIMER_B interrupt Valid values are:

- **TIMER_B_TBIE_INTERRUPT_ENABLE**
- **TIMER_B_TBIE_INTERRUPT_DISABLE** [Default]

captureCompareInterruptEnable_CCR0_CCIE is to enable or disable TIMER_B CCR0 capture compare interrupt. Valid values are:

- **TIMER_B_CCIE_CCR0_INTERRUPT_ENABLE**
- **TIMER_B_CCIE_CCR0_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_B clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_B_DO_CLEAR**
- **TIMER_B_SKIP_CLEAR** [Default]

Modified bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:

None

29.2.2.7 void TIMER_B_disableCaptureCompareInterrupt (uint16_t *baseAddress*, uint16_t *captureCompareRegister*)

Disable capture compare interrupt.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the TIMER_B module.

captureCompareRegister selects the capture compare register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

Modified bits of **TBxCCTLn** register.

Returns:

None

29.2.2.8 void TIMER_B_disableInterrupt (uint16_t *baseAddress*)

Disable TIMER_B interrupt.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the TIMER_B module.

Modified bits of **TBxCTL** register.

Returns:

None

29.2.2.9 void TIMER_B_enableCaptureCompareInterrupt (uint16_t *baseAddress*, uint16_t *captureCompareRegister*)

Enable capture compare interrupt.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the TIMER_B module.

captureCompareRegister selects the capture compare register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

Modified bits of **TBxCCTLn** register.

Returns:

None

29.2.2.10 void TIMER_B_enableInterrupt (uint16_t *baseAddress*)

Enable TIMER_B interrupt.

Enables TIMER_B interrupt. Does not clear interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	22
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the TIMER_B module.

Modified bits of **TBxCTL** register.

Returns:
None

29.2.2.11 void **TIMER_B_generatePWM** (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerPeriod*, uint16_t *compareRegister*, uint16_t *compareOutputMode*, uint16_t *dutyCycle*)

DEPRECATED - Generate a PWM with **TIMER_B** running in up mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	88
TI Compiler 4.2.1	Size	50
TI Compiler 4.2.1	Speed	50
IAR 5.51.6	None	80
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	68
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	76

Parameters:

baseAddress is the base address of the **TIMER_B** module.

clockSource selects the clock source Valid values are:

- **TIMER_B_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_B_CLOCKSOURCE_ACLK**
- **TIMER_B_CLOCKSOURCE_SMCLK**
- **TIMER_B_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the divider for Clock source. Valid values are:

- **TIMER_B_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_B_CLOCKSOURCE_DIVIDER_2**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_10**
- **TIMER_B_CLOCKSOURCE_DIVIDER_12**
- **TIMER_B_CLOCKSOURCE_DIVIDER_14**
- **TIMER_B_CLOCKSOURCE_DIVIDER_16**
- **TIMER_B_CLOCKSOURCE_DIVIDER_20**
- **TIMER_B_CLOCKSOURCE_DIVIDER_24**
- **TIMER_B_CLOCKSOURCE_DIVIDER_28**
- **TIMER_B_CLOCKSOURCE_DIVIDER_32**
- **TIMER_B_CLOCKSOURCE_DIVIDER_40**
- **TIMER_B_CLOCKSOURCE_DIVIDER_48**
- **TIMER_B_CLOCKSOURCE_DIVIDER_56**
- **TIMER_B_CLOCKSOURCE_DIVIDER_64**

timerPeriod selects the desired **TIMER_B** period

compareRegister selects the compare register being used. Refer to datasheet to ensure the device has the compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

compareOutputMode specifies the output mode. Valid values are:

- **TIMER_B_OUTPUTMODE_OUTBITVALUE** [Default]
- **TIMER_B_OUTPUTMODE_SET**
- **TIMER_B_OUTPUTMODE_TOGGLE_RESET**
- **TIMER_B_OUTPUTMODE_SET_RESET**
- **TIMER_B_OUTPUTMODE_TOGGLE**
- **TIMER_B_OUTPUTMODE_RESET**
- **TIMER_B_OUTPUTMODE_TOGGLE_SET**
- **TIMER_B_OUTPUTMODE_RESET_SET**

dutyCycle specifies the dutycycle for the generated waveform

Modified bits of **TBxCCTLn** register, bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:

None

29.2.2.12 uint16_t TIMER_B_getCaptureCompareCount (uint16_t *baseAddress*, uint16_t *captureCompareRegister*)

Get current capturecompare count.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	20
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	26
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the TIMER_B module.

captureCompareRegister selects the capture compare register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

Returns:

Current count as uint16_t

29.2.2.13 uint32_t TIMER_B_getCaptureCompareInterruptStatus (uint16_t *baseAddress*, uint16_t *captureCompareRegister*, uint16_t *mask*)

Return capture compare interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	10
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Parameters:

baseAddress is the base address of the TIMER_B module.

captureCompareRegister selects the capture compare register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

mask is the mask for the interrupt status Mask value is the logical OR of any of the following:

- **TIMER_B_CAPTURE_OVERFLOW**
- **TIMER_B_CAPTURECOMPARE_INTERRUPT_FLAG**

Returns:

Logical OR of any of the following:

- **TIMER_B_CAPTURE_OVERFLOW**
 - **TIMER_B_CAPTURECOMPARE_INTERRUPT_FLAG**
- indicating the status of the masked interrupts

29.2.2.14 uint16_t TIMER_B_getCounterValue (uint16_t *baseAddress*)

Reads the current timer count value.

Reads the current count value of the timer. There is a majority vote system in place to confirm an accurate value is returned. The **TIMER_B_THRESHOLD** define in the associated header file can be modified so that the votes must be closer together for a consensus to occur.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	88
TI Compiler 4.2.1	Size	38
TI Compiler 4.2.1	Speed	38
IAR 5.51.6	None	58
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	102
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	48

Parameters:

baseAddress is the base address of the Timer module.

Returns:

Majority vote of timer count value

29.2.2.15 uint32_t TIMER_B_getInterruptStatus (uint16_t baseAddress)

Get TIMER_B interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	18
TI Compiler 4.2.1	Size	10
TI Compiler 4.2.1	Speed	10
IAR 5.51.6	None	8
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the TIMER_B module.

Returns:

One of the following:

- **TIMER_B_INTERRUPT_NOT_PENDING**
- **TIMER_B_INTERRUPT_PENDING**
indicating the status of the TIMER_B interrupt

29.2.2.16 uint8_t TIMER_B_getOutputForOutputModeOutBitValue (uint16_t baseAddress, uint16_t captureCompareRegister)

Get output bit for output mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	28
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	20
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	16
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the TIMER_B module.

captureCompareRegister selects the capture compare register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**

- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

Returns:

One of the following:

- **TIMER_B_OUTPUTMODE_OUTBITVALUE_HIGH**
- **TIMER_B_OUTPUTMODE_OUTBITVALUE_LOW**

29.2.2.17 uint8_t TIMER_B_getSynchronizedCaptureCompareInput (uint16_t *baseAddress*, uint16_t *captureCompareRegister*, uint16_t *synchronized*)

Get synchronized capturecompare input.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	36
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	16
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:

baseAddress is the base address of the TIMER_B module.

captureCompareRegister selects the capture compare register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

synchronized selects the type of capture compare input Valid values are:

- **TIMER_B_READ_SYNCHRONIZED_CAPTURECOMPAREINPUT**
- **TIMER_B_READ_CAPTURE_COMPARE_INPUT**

Returns:

One of the following:

- **TIMER_B_CAPTURECOMPARE_INPUT_HIGH**
- **TIMER_B_CAPTURECOMPARE_INPUT_LOW**

29.2.2.18 void TIMER_B_initCapture (uint16_t *baseAddress*, uint16_t *captureRegister*, uint16_t *captureMode*, uint16_t *captureInputSelect*, uint16_t *synchronizeCaptureSource*, uint16_t *captureInterruptEnable*, uint16_t *captureOutputMode*)

DEPRECATED - Initializes Capture Mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	88
TI Compiler 4.2.1	Size	50
TI Compiler 4.2.1	Speed	50
IAR 5.51.6	None	80
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	36

Parameters:

baseAddress is the base address of the TIMER_B module.

captureRegister selects the capture register being used. Refer to datasheet to ensure the device has the capture register being used. Valid values are:

- TIMER_B_CAPTURECOMPARE_REGISTER_0
- TIMER_B_CAPTURECOMPARE_REGISTER_1
- TIMER_B_CAPTURECOMPARE_REGISTER_2
- TIMER_B_CAPTURECOMPARE_REGISTER_3
- TIMER_B_CAPTURECOMPARE_REGISTER_4
- TIMER_B_CAPTURECOMPARE_REGISTER_5
- TIMER_B_CAPTURECOMPARE_REGISTER_6

captureMode is the capture mode selected. Valid values are:

- TIMER_B_CAPTUREMODE_NO_CAPTURE [Default]
- TIMER_B_CAPTUREMODE_RISING_EDGE
- TIMER_B_CAPTUREMODE_FALLING_EDGE
- TIMER_B_CAPTUREMODE_RISING_AND_FALLING_EDGE

captureInputSelect decides the Input Select Valid values are:

- TIMER_B_CAPTURE_INPUTSELECT_CC1xA [Default]
- TIMER_B_CAPTURE_INPUTSELECT_CC1xB
- TIMER_B_CAPTURE_INPUTSELECT_GND
- TIMER_B_CAPTURE_INPUTSELECT_Vcc

synchronizeCaptureSource decides if capture source should be synchronized with TIMER_B clock Valid values are:

- TIMER_B_CAPTURE_ASYNCHRONOUS [Default]
- TIMER_B_CAPTURE_SYNCHRONOUS

captureInterruptEnable is to enable or disable TIMER_B capture compare interrupt. Valid values are:

- TIMER_B_CAPTURECOMPARE_INTERRUPT_DISABLE [Default]
- TIMER_B_CAPTURECOMPARE_INTERRUPT_ENABLE

captureOutputMode specifies the output mode. Valid values are:

- TIMER_B_OUTPUTMODE_OUTBITVALUE [Default]
- TIMER_B_OUTPUTMODE_SET
- TIMER_B_OUTPUTMODE_TOGGLE_RESET
- TIMER_B_OUTPUTMODE_SET_RESET
- TIMER_B_OUTPUTMODE_TOGGLE
- TIMER_B_OUTPUTMODE_RESET
- TIMER_B_OUTPUTMODE_TOGGLE_SET
- TIMER_B_OUTPUTMODE_RESET_SET

Modified bits of **TBxCCTLn** register.

Returns:
None

29.2.2.19 void TIMER_B_initCaptureMode (uint16_t *baseAddress*,
TIMER_B_initCaptureModeParam * *param*)

Initializes Capture Mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	70
TI Compiler 4.2.1	Size	48
TI Compiler 4.2.1	Speed	50
IAR 5.51.6	None	50
IAR 5.51.6	Size	48
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	128
MSPGCC 4.8.0	Size	48
MSPGCC 4.8.0	Speed	48

Parameters:

baseAddress is the base address of the TIMER_B module.

param is the pointer to struct for capture mode initialization.

Modified bits of **TBxCCTLn** register.

Returns:
None

29.2.2.20 void TIMER_B_initCompare (uint16_t *baseAddress*, uint16_t *compareRegister*,
uint16_t *compareInterruptEnable*, uint16_t *compareOutputMode*, uint16_t
compareValue)

DEPRECATED - Initializes Compare Mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	82
TI Compiler 4.2.1	Size	30
TI Compiler 4.2.1	Speed	30
IAR 5.51.6	None	64
IAR 5.51.6	Size	54
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	28

Parameters:

baseAddress is the base address of the TIMER_B module.

compareRegister selects the compare register being used. Refer to datasheet to ensure the device has the compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

compareInterruptEnable is to enable or disable TIMER_B capture compare interrupt. Valid values are:

- **TIMER_B_CAPTURECOMPARE_INTERRUPT_DISABLE** [Default]
- **TIMER_B_CAPTURECOMPARE_INTERRUPT_ENABLE**

compareOutputMode specifies the output mode. Valid values are:

- **TIMER_B_OUTPUTMODE_OUTBITVALUE** [Default]
- **TIMER_B_OUTPUTMODE_SET**
- **TIMER_B_OUTPUTMODE_TOGGLE_RESET**
- **TIMER_B_OUTPUTMODE_SET_RESET**
- **TIMER_B_OUTPUTMODE_TOGGLE**
- **TIMER_B_OUTPUTMODE_RESET**
- **TIMER_B_OUTPUTMODE_TOGGLE_SET**
- **TIMER_B_OUTPUTMODE_RESET_SET**

compareValue is the count to be compared with in compare mode

Modified bits of **TBxCCTLn** register and bits of **TBxCCRn** register.

Returns:

None

29.2.2.21 void TIMER_B_initCompareLatchLoadEvent (uint16_t *baseAddress*, uint16_t *compareRegister*, uint16_t *compareLatchLoadEvent*)

Selects Compare Latch Load Event.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	42
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Parameters:

baseAddress is the base address of the TIMER_B module.

compareRegister selects the compare register being used. Refer to datasheet to ensure the device has the compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

compareLatchLoadEvent selects the latch load event Valid values are:

- **TIMER_B_LATCH_ON_WRITE_TO_TBxCCRn_COMPARE_REGISTER** [Default]
- **TIMER_B_LATCH_WHEN_COUNTER_COUNTS_TO_0_IN_UP_OR_CONT_MODE**
- **TIMER_B_LATCH_WHEN_COUNTER_COUNTS_TO_0_IN_UPDOWN_MODE**
- **TIMER_B_LATCH_WHEN_COUNTER_COUNTS_TO_CURRENT_COMPARE_LATCH_VALUE**

Modified bits are **CLLD** of **TBxCCTLn** register.

Returns:

None

29.2.2.22 void **TIMER_B_initCompareMode** (uint16_t *baseAddress*,
TIMER_B_initCompareModeParam * *param*)

Initializes Compare Mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	72
TI Compiler 4.2.1	Size	46
TI Compiler 4.2.1	Speed	46
IAR 5.51.6	None	52
IAR 5.51.6	Size	50
IAR 5.51.6	Speed	50
MSPGCC 4.8.0	None	126
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

Parameters:

baseAddress is the base address of the **TIMER_B** module.

param is the pointer to struct for compare mode initialization.

Modified bits of **TBxCCTLn** register and bits of **TBxCCRn** register.

Returns:

None

29.2.2.23 void **TIMER_B_initContinuousMode** (uint16_t *baseAddress*,
TIMER_B_initContinuousModeParam * *param*)

Configures **TIMER_B** in continuous mode.

This API does not start the timer. Timer needs to be started when required using the **TIMER_B_startCounter** API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	106
TI Compiler 4.2.1	Size	70
TI Compiler 4.2.1	Speed	70
IAR 5.51.6	None	78
IAR 5.51.6	Size	70
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	170
MSPGCC 4.8.0	Size	82
MSPGCC 4.8.0	Speed	82

Parameters:

baseAddress is the base address of the TIMER_B module.

param is the pointer to struct for continuous mode initialization.

Modified bits of **TBxCTL** register.

Returns:

None

29.2.2.24 void TIMER_B_initUpDownMode (uint16_t *baseAddress*,
TIMER_B_initUpDownModeParam * *param*)

Configures TIMER_B in up down mode.

This API does not start the timer. Timer needs to be started when required using the TIMER_B_startCounter API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	152
TI Compiler 4.2.1	Size	98
TI Compiler 4.2.1	Speed	98
IAR 5.51.6	None	118
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	72
MSPGCC 4.8.0	None	248
MSPGCC 4.8.0	Size	116
MSPGCC 4.8.0	Speed	118

Parameters:

baseAddress is the base address of the TIMER_B module.

param is the pointer to struct for up-down mode initialization.

Modified bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:

None

29.2.2.25 void TIMER_B_initUpMode (uint16_t *baseAddress*, TIMER_B_initUpModeParam
* *param*)

Configures TIMER_B in up mode.

This API does not start the timer. Timer needs to be started when required using the TIMER_B_startCounter API.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	152
TI Compiler 4.2.1	Size	98
TI Compiler 4.2.1	Speed	98
IAR 5.51.6	None	120
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	74
MSPGCC 4.8.0	None	248
MSPGCC 4.8.0	Size	116
MSPGCC 4.8.0	Speed	118

Parameters:

baseAddress is the base address of the TIMER_B module.

param is the pointer to struct for up mode initialization.

Modified bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:

None

29.2.2.26 void TIMER_B_outputPWM (uint16_t *baseAddress*, TIMER_B_outputPWMPParam * *param*)

Generate a PWM with TIMER_B running in up mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	136
TI Compiler 4.2.1	Size	86
TI Compiler 4.2.1	Speed	86
IAR 5.51.6	None	108
IAR 5.51.6	Size	100
IAR 5.51.6	Speed	100
MSPGCC 4.8.0	None	228
MSPGCC 4.8.0	Size	100
MSPGCC 4.8.0	Speed	100

Parameters:

baseAddress is the base address of the TIMER_B module.

param is the pointer to struct for PWM configuration.

Modified bits of **TBxCCTLn** register, bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:

None

29.2.2.27 void TIMER_B_selectCounterLength (uint16_t *baseAddress*, uint16_t *counterLength*)

Selects TIMER_B counter length.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the TIMER_B module.

counterLength selects the value of counter length. Valid values are:

- **TIMER_B_COUNTER_16BIT** [Default]
- **TIMER_B_COUNTER_12BIT**
- **TIMER_B_COUNTER_10BIT**
- **TIMER_B_COUNTER_8BIT**

Modified bits are **CNTL** of **TBxCTL** register.

Returns:

None

29.2.2.28 void TIMER_B_selectLatchingGroup (uint16_t baseAddress, uint16_t groupLatch)

Selects TIMER_B Latching Group.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the TIMER_B module.

groupLatch selects the latching group. Valid values are:

- **TIMER_B_GROUP_NONE** [Default]
- **TIMER_B_GROUP_CL12_CL23_CL56**
- **TIMER_B_GROUP_CL123_CL456**
- **TIMER_B_GROUP_ALL**

Modified bits are **TBCLGRP** of **TBxCTL** register.

Returns:

None

29.2.2.29 void TIMER_B_setCompareValue (uint16_t baseAddress, uint16_t compareRegister, uint16_t compareValue)

Sets the value of the capture-compare register.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	30
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the TIMER_B module.

compareRegister selects the compare register being used. Refer to datasheet to ensure the device has the compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**
- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

compareValue is the count to be compared with in compare mode

Modified bits of **TBxCCRn** register.

Returns:

None

29.2.2.30 void **TIMER_B_setOutputForOutputModeOutBitValue** (uint16_t *baseAddress*, uint16_t *captureCompareRegister*, uint8_t *outputModeOutBitValue*)

Set output bit for output mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	42
TI Compiler 4.2.1	Size	14
TI Compiler 4.2.1	Speed	14
IAR 5.51.6	None	20
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	74
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

baseAddress is the base address of the TIMER_B module.

captureCompareRegister selects the capture compare register being used. Refer to datasheet to ensure the device has the capture compare register being used. Valid values are:

- **TIMER_B_CAPTURECOMPARE_REGISTER_0**
- **TIMER_B_CAPTURECOMPARE_REGISTER_1**
- **TIMER_B_CAPTURECOMPARE_REGISTER_2**
- **TIMER_B_CAPTURECOMPARE_REGISTER_3**
- **TIMER_B_CAPTURECOMPARE_REGISTER_4**

- **TIMER_B_CAPTURECOMPARE_REGISTER_5**
- **TIMER_B_CAPTURECOMPARE_REGISTER_6**

outputModeOutBitValue the value to be set for out bit Valid values are:

- **TIMER_B_OUTPUTMODE_OUTBITVALUE_HIGH**
- **TIMER_B_OUTPUTMODE_OUTBITVALUE_LOW**

Modified bits of **TBxCCTLn** register.

Returns:

None

29.2.2.31 void **TIMER_B_startContinuousMode** (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerInterruptEnable_TBIE*, uint16_t *timerClear*)

DEPRECATED - Spelling Error Fixed. Starts **TIMER_B** in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	52
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	34
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	52
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	58

Parameters:

baseAddress is the base address of the **TIMER_B** module.

clockSource selects the clock source Valid values are:

- **TIMER_B_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_B_CLOCKSOURCE_ACLK**
- **TIMER_B_CLOCKSOURCE_SMCLK**
- **TIMER_B_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the divider for Clock source. Valid values are:

- **TIMER_B_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_B_CLOCKSOURCE_DIVIDER_2**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_10**
- **TIMER_B_CLOCKSOURCE_DIVIDER_12**
- **TIMER_B_CLOCKSOURCE_DIVIDER_14**
- **TIMER_B_CLOCKSOURCE_DIVIDER_16**
- **TIMER_B_CLOCKSOURCE_DIVIDER_20**
- **TIMER_B_CLOCKSOURCE_DIVIDER_24**
- **TIMER_B_CLOCKSOURCE_DIVIDER_28**
- **TIMER_B_CLOCKSOURCE_DIVIDER_32**
- **TIMER_B_CLOCKSOURCE_DIVIDER_40**

- **TIMER_B_CLOCKSOURCE_DIVIDER_48**
- **TIMER_B_CLOCKSOURCE_DIVIDER_56**
- **TIMER_B_CLOCKSOURCE_DIVIDER_64**

timerInterruptEnable_TBIE is to enable or disable TIMER_B interrupt Valid values are:

- **TIMER_B_TBIE_INTERRUPT_ENABLE**
- **TIMER_B_TBIE_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_B clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_B_DO_CLEAR**
- **TIMER_B_SKIP_CLEAR** [Default]

Modified bits of **TBxCTL** register.

Returns:

None

29.2.2.32 void **TIMER_B_startContinuousMode** (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerInterruptEnable_TBIE*, uint16_t *timerClear*)

DEPRECATED - Replaced by **TIMER_B_configureContinuousMode** and **TIMER_B_startCounter** API. Starts **TIMER_B** in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	80
TI Compiler 4.2.1	Size	38
TI Compiler 4.2.1	Speed	38
IAR 5.51.6	None	68
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	64
MSPGCC 4.8.0	Speed	58

Parameters:

baseAddress is the base address of the **TIMER_B** module.

clockSource selects the clock source Valid values are:

- **TIMER_B_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_B_CLOCKSOURCE_ACLK**
- **TIMER_B_CLOCKSOURCE_SMCLK**
- **TIMER_B_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the divider for Clock source. Valid values are:

- **TIMER_B_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_B_CLOCKSOURCE_DIVIDER_2**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_10**
- **TIMER_B_CLOCKSOURCE_DIVIDER_12**
- **TIMER_B_CLOCKSOURCE_DIVIDER_14**
- **TIMER_B_CLOCKSOURCE_DIVIDER_16**

- **TIMER_B_CLOCKSOURCE_DIVIDER_20**
- **TIMER_B_CLOCKSOURCE_DIVIDER_24**
- **TIMER_B_CLOCKSOURCE_DIVIDER_28**
- **TIMER_B_CLOCKSOURCE_DIVIDER_32**
- **TIMER_B_CLOCKSOURCE_DIVIDER_40**
- **TIMER_B_CLOCKSOURCE_DIVIDER_48**
- **TIMER_B_CLOCKSOURCE_DIVIDER_56**
- **TIMER_B_CLOCKSOURCE_DIVIDER_64**

timerInterruptEnable_TBIE is to enable or disable TIMER_B interrupt Valid values are:

- **TIMER_B_TBIE_INTERRUPT_ENABLE**
- **TIMER_B_TBIE_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_B clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_B_DO_CLEAR**
- **TIMER_B_SKIP_CLEAR** [Default]

Modified bits of **TBxCTL** register.

Returns:

None

29.2.2.33 void TIMER_B_startCounter (uint16_t *baseAddress*, uint16_t *timerMode*)

Starts TIMER_B counter.

This function assumes that the timer has been previously configured using **TIMER_B_configureContinuousMode**, **TIMER_B_configureUpMode** or **TIMER_B_configureUpDownMode**.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	22
TI Compiler 4.2.1	Size	6
TI Compiler 4.2.1	Speed	6
IAR 5.51.6	None	6
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	30
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Parameters:

baseAddress is the base address of the TIMER_B module.

timerMode selects the mode of the timer Valid values are:

- **TIMER_B_STOP_MODE**
- **TIMER_B_UP_MODE**
- **TIMER_B_CONTINUOUS_MODE** [Default]
- **TIMER_B_UPDOWN_MODE**

Modified bits of **TBxCTL** register.

Returns:

None

29.2.2.34 void TIMER_B_startUpDownMode (uint16_t *baseAddress*,
uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t
timerPeriod, uint16_t *timerInterruptEnable_TBIE*, uint16_t
captureCompareInterruptEnable_CCR0_CCIE, uint16_t *timerClear*)

DEPRECATED - Replaced by TIMER_B_configureUpDownMode and TIMER_B_startCounter API. Starts TIMER_B in up down mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	92
TI Compiler 4.2.1	Size	54
TI Compiler 4.2.1	Speed	54
IAR 5.51.6	None	84
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	76
MSPGCC 4.8.0	Speed	94

Parameters:

baseAddress is the base address of the TIMER_B module.

clockSource selects the clock source Valid values are:

- **TIMER_B_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_B_CLOCKSOURCE_ACLK**
- **TIMER_B_CLOCKSOURCE_SMCLK**
- **TIMER_B_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the divider for Clock source. Valid values are:

- **TIMER_B_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_B_CLOCKSOURCE_DIVIDER_2**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_10**
- **TIMER_B_CLOCKSOURCE_DIVIDER_12**
- **TIMER_B_CLOCKSOURCE_DIVIDER_14**
- **TIMER_B_CLOCKSOURCE_DIVIDER_16**
- **TIMER_B_CLOCKSOURCE_DIVIDER_20**
- **TIMER_B_CLOCKSOURCE_DIVIDER_24**
- **TIMER_B_CLOCKSOURCE_DIVIDER_28**
- **TIMER_B_CLOCKSOURCE_DIVIDER_32**
- **TIMER_B_CLOCKSOURCE_DIVIDER_40**
- **TIMER_B_CLOCKSOURCE_DIVIDER_48**
- **TIMER_B_CLOCKSOURCE_DIVIDER_56**
- **TIMER_B_CLOCKSOURCE_DIVIDER_64**

timerPeriod is the specified TIMER_B period

timerInterruptEnable_TBIE is to enable or disable TIMER_B interrupt Valid values are:

- **TIMER_B_TBIE_INTERRUPT_ENABLE**
- **TIMER_B_TBIE_INTERRUPT_DISABLE** [Default]

captureCompareInterruptEnable_CCR0_CCIE is to enable or disable TIMER_B CCR0 capture compare interrupt.

Valid values are:

- **TIMER_B_CCIE_CCR0_INTERRUPT_ENABLE**

- **TIMER_B_CCIE_CCR0_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_B clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_B_DO_CLEAR**
- **TIMER_B_SKIP_CLEAR** [Default]

Modified bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:
None

29.2.2.35 void **TIMER_B_startUpMode** (uint16_t *baseAddress*, uint16_t *clockSource*, uint16_t *clockSourceDivider*, uint16_t *timerPeriod*, uint16_t *timerInterruptEnable_TBIE*, uint16_t *captureCompareInterruptEnable_CCR0_CCIE*, uint16_t *timerClear*)

DEPRECATED - Replaced by **TIMER_B_configureUpMode** and **TIMER_B_startCounter** API. Starts **TIMER_B** in up mode.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	92
TI Compiler 4.2.1	Size	54
TI Compiler 4.2.1	Speed	54
IAR 5.51.6	None	84
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	62
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	76
MSPGCC 4.8.0	Speed	94

Parameters:

baseAddress is the base address of the **TIMER_B** module.

clockSource selects the clock source Valid values are:

- **TIMER_B_CLOCKSOURCE_EXTERNAL_TXCLK** [Default]
- **TIMER_B_CLOCKSOURCE_ACLK**
- **TIMER_B_CLOCKSOURCE_SMCLK**
- **TIMER_B_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK**

clockSourceDivider is the divider for Clock source. Valid values are:

- **TIMER_B_CLOCKSOURCE_DIVIDER_1** [Default]
- **TIMER_B_CLOCKSOURCE_DIVIDER_2**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_10**
- **TIMER_B_CLOCKSOURCE_DIVIDER_12**
- **TIMER_B_CLOCKSOURCE_DIVIDER_14**
- **TIMER_B_CLOCKSOURCE_DIVIDER_16**
- **TIMER_B_CLOCKSOURCE_DIVIDER_20**
- **TIMER_B_CLOCKSOURCE_DIVIDER_24**
- **TIMER_B_CLOCKSOURCE_DIVIDER_28**
- **TIMER_B_CLOCKSOURCE_DIVIDER_32**
- **TIMER_B_CLOCKSOURCE_DIVIDER_40**

- **TIMER_B_CLOCKSOURCE_DIVIDER_48**
- **TIMER_B_CLOCKSOURCE_DIVIDER_56**
- **TIMER_B_CLOCKSOURCE_DIVIDER_64**

timerPeriod is the specified TIMER_B period. This is the value that gets written into the CCR0. Limited to 16 bits[uint16_t]

timerInterruptEnable_TBIE is to enable or disable TIMER_B interrupt Valid values are:

- **TIMER_B_TBIE_INTERRUPT_ENABLE**
- **TIMER_B_TBIE_INTERRUPT_DISABLE** [Default]

captureCompareInterruptEnable_CCR0_CCIE is to enable or disable TIMER_B CCR0 capture compare interrupt. Valid values are:

- **TIMER_B_CCIE_CCR0_INTERRUPT_ENABLE**
- **TIMER_B_CCIE_CCR0_INTERRUPT_DISABLE** [Default]

timerClear decides if TIMER_B clock divider, count direction, count need to be reset. Valid values are:

- **TIMER_B_DO_CLEAR**
- **TIMER_B_SKIP_CLEAR** [Default]

Modified bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:

None

29.2.2.36 void TIMER_B_stop (uint16_t baseAddress)

Stops the TIMER_B.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	24
TI Compiler 4.2.1	Size	12
TI Compiler 4.2.1	Speed	12
IAR 5.51.6	None	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	34
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the TIMER_B module.

Modified bits of **TBxCTL** register.

Returns:

None

29.3 Programming Example

The following example shows some TIMER_B operations using the APIs

```
{
    //Start TIMER_B
    TIMER_B_configureUpMode(    TIMER_B0_BASE,
        TIMER_B_CLOCKSOURCE_SMCLK,
        TIMER_B_CLOCKSOURCE_DIVIDER_1,
        511,
```

```
TIMER_B_TBIE_INTERRUPT_DISABLE,  
TIMER_B_CCIE_CCR0_INTERRUPT_DISABLE,  
TIMER_B_DO_CLEAR  
);  
  
TIMER_B_startCounter(  TIMER_B0_BASE,  
                       TIMER_B_UP_MODE  
                       );  
  
//Initialize compare mode to generate PWM1  
TIMER_B_initCompare(TIMER_B0_BASE,  
                    TIMER_B_CAPTURECOMPARE_REGISTER_1,  
                    TIMER_B_CAPTURECOMPARE_INTERRUPT_DISABLE,  
                    TIMER_B_OUTPUTMODE_RESET_SET,  
                    383  
                    );  
  
//Initialize compare mode to generate PWM2  
TIMER_B_initCompare(TIMER_B0_BASE,  
                    TIMER_B_CAPTURECOMPARE_REGISTER_2,  
                    TIMER_B_CAPTURECOMPARE_INTERRUPT_ENABLE,  
                    TIMER_B_OUTPUTMODE_RESET_SET,  
                    128  
                    );  
}
```

30 Tag Length Value

Introduction	369
API Functions	369
Programming Example	374

30.1 Introduction

The TLV structure is a table stored in flash memory that contains device-specific information. This table is read-only and is write-protected. It contains important information for using and calibrating the device. A list of the contents of the TLV is available in the device-specific data sheet (in the Device Descriptors section), and an explanation on its functionality is available in the MSP430x5xx/MSP430x6xx Family User's Guide

This driver is contained in `tlv.c`, with `tlv.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	602
TI Compiler 4.2.1	Size	364
TI Compiler 4.2.1	Speed	368
IAR 5.51.6	None	456
IAR 5.51.6	Size	302
IAR 5.51.6	Speed	354
MSPGCC 4.8.0	None	750
MSPGCC 4.8.0	Size	432
MSPGCC 4.8.0	Speed	706

30.2 API Functions

Functions

- `uint16_t TLV_getDeviceType ()`
- `void TLV_getInfo (uint8_t tag, uint8_t instance, uint8_t *length, uint16_t **data_address)`
- `uint8_t TLV_getInterrupt (uint8_t tag)`
- `uint16_t TLV_getMemory (uint8_t instance)`
- `uint16_t TLV_getPeripheral (uint8_t tag, uint8_t instance)`

30.2.1 Detailed Description

The APIs that help in querying the information in the TLV structure are listed

- `TLV_getInfo()` This function retrieves the value of a tag and the length of the tag.
- `TLV_getDeviceType()` This function retrieves the unique device ID from the TLV structure.
- `TLV_getMemory()` The returned value is zero if the end of the memory list is reached.
- `TLV_getPeripheral()` The returned value is zero if the specified tag value (peripheral) is not available in the device.
- `TLV_getInterrupt()` The returned value is zero if the specified interrupt vector is not defined.

30.2.2 Function Documentation

30.2.2.1 uint16_t TLV_getDeviceType (void)

Retrieves the unique device ID from the TLV structure.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	16
TI Compiler 4.2.1	Size	8
TI Compiler 4.2.1	Speed	8
IAR 5.51.6	None	8
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	16
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:

The device ID is returned as type uint16_t.

30.2.2.2 void TLV_getInfo (uint8_t tag, uint8_t instance, uint8_t * length, uint16_t ** data_address)

Gets TLV Info.

The TLV structure uses a tag or base address to identify segments of the table where information is stored. Some examples of TLV tags are Peripheral Descriptor, Interrupts, Info Block and Die Record. This function retrieves the value of a tag and the length of the tag.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	126
TI Compiler 4.2.1	Size	72
TI Compiler 4.2.1	Speed	76
IAR 5.51.6	None	82
IAR 5.51.6	Size	70
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	168
MSPGCC 4.8.0	Size	100
MSPGCC 4.8.0	Speed	110

Parameters:

tag represents the tag for which the information needs to be retrieved. Valid values are:

- TLV_TAG_LD TAG
- TLV_TAG_PD TAG
- TLV_TAG_Reserved3
- TLV_TAG_Reserved4
- TLV_TAG_BLANK
- TLV_TAG_Reserved6
- TLV_TAG_Reserved7
- TLV_TAG_TAGEND
- TLV_TAG_TAGEXT
- TLV_TAG_TIMER_D_CAL

- TLV_DEVICE_ID_0
- TLV_DEVICE_ID_1
- TLV_TAG_DIERECORD
- TLV_TAG_ADCCAL
- TLV_TAG_ADC12CAL
- TLV_TAG_ADC10CAL
- TLV_TAG_REFCAL

instance In some cases a specific tag may have more than one instance. For example there may be multiple instances of timer calibration data present under a single Timer Cal tag. This variable specifies the instance for which information is to be retrieved (0, 1, etc.). When only one instance exists; 0 is passed.

length Acts as a return through indirect reference. The function retrieves the value of the TLV tag length. This value is pointed to by *length and can be used by the application level once the function is called. If the specified tag is not found then the pointer is null 0.

data_address acts as a return through indirect reference. Once the function is called data_address points to the pointer that holds the value retrieved from the specified TLV tag. If the specified tag is not found then the pointer is null 0.

Returns:
None

30.2.2.3 uint8_t TLV_getInterrupt (uint8_t tag)

Get interrupt information from the TLV.

This function is used to retrieve information on available interrupt vectors. It allows the user to check if a specific interrupt vector is defined in a given device.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	160
TI Compiler 4.2.1	Size	98
TI Compiler 4.2.1	Speed	98
IAR 5.51.6	None	128
IAR 5.51.6	Size	82
IAR 5.51.6	Speed	102
MSPGCC 4.8.0	None	200
MSPGCC 4.8.0	Size	106
MSPGCC 4.8.0	Speed	228

Parameters:

tag represents the tag for the interrupt vector. Interrupt vector tags number from 0 to N depending on the number of available interrupts. Refer to the device datasheet for a list of available interrupts.

Returns:

The returned value is zero if the specified interrupt vector is not defined.

30.2.2.4 uint16_t TLV_getMemory (uint8_t instance)

Gets memory information.

The Peripheral Descriptor tag is split into two portions a list of the available flash memory blocks followed by a list of available peripherals. This function is used to parse through the first portion and calculate the total flash memory available in a device. The typical usage is to call the TLV_getMemory which returns a non-zero value until the entire memory list has been parsed. When a zero is returned, it indicates that all the memory blocks have been counted and the next address holds the beginning of the device peripheral list.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	114
TI Compiler 4.2.1	Size	70
TI Compiler 4.2.1	Speed	70
IAR 5.51.6	None	92
IAR 5.51.6	Size	54
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	128
MSPGCC 4.8.0	Size	86
MSPGCC 4.8.0	Speed	120

Parameters:

instance In some cases a specific tag may have more than one instance. This variable specifies the instance for which information is to be retrieved (0, 1 etc). When only one instance exists; 0 is passed.

Returns:

The returned value is zero if the end of the memory list is reached.

30.2.2.5 uint16_t TLV_getPeripheral (uint8_t tag, uint8_t instance)

Gets peripheral information from the TLV.

The Peripheral Descriptor tag is split into two portions a list of the available flash memory blocks followed by a list of available peripherals. This function is used to parse through the second portion and can be used to check if a specific peripheral is present in a device. The function calls [TLV_getPeripheral\(\)](#) recursively until the end of the memory list and consequently the beginning of the peripheral list is reached. <

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	186
TI Compiler 4.2.1	Size	116
TI Compiler 4.2.1	Speed	116
IAR 5.51.6	None	146
IAR 5.51.6	Size	90
IAR 5.51.6	Speed	106
MSPGCC 4.8.0	None	238
MSPGCC 4.8.0	Size	134
MSPGCC 4.8.0	Speed	242

Parameters:

tag represents represents the tag for a specific peripheral for which the information needs to be retrieved. In the header file tlv.h specific peripheral tags are pre-defined, for example USCIA_B and TA0 are defined as TLV_PID_USCI_AB and TLV_PID_TA2 respectively. Valid values are:

- TLV_PID_NO_MODULE - No Module
- TLV_PID_PORTMAPPING - Port Mapping
- TLV_PID_MSP430CPUXV2 - MSP430CPUXV2
- TLV_PID_JTAG - JTAG
- TLV_PID_SBW - SBW
- TLV_PID_EEM_XS - EEM X-Small
- TLV_PID_EEM_S - EEM Small
- TLV_PID_EEM_M - EEM Medium
- TLV_PID_EEM_L - EEM Large
- TLV_PID_PMM - PMM
- TLV_PID_PMM_FR - PMM FRAM
- TLV_PID_FCTL - Flash
- TLV_PID_CRC16 - CRC16

- TLV_PID_CRC16_RB - CRC16 Reverse
- TLV_PID_WDT_A - WDT_A
- TLV_PID_SFR - SFR
- TLV_PID_SYS - SYS
- TLV_PID_RAMCTL - RAMCTL
- TLV_PID_DMA_1 - DMA 1
- TLV_PID_DMA_3 - DMA 3
- TLV_PID_UCS - UCS
- TLV_PID_DMA_6 - DMA 6
- TLV_PID_DMA_2 - DMA 2
- TLV_PID_PORT1_2 - Port 1 + 2 / A
- TLV_PID_PORT3_4 - Port 3 + 4 / B
- TLV_PID_PORT5_6 - Port 5 + 6 / C
- TLV_PID_PORT7_8 - Port 7 + 8 / D
- TLV_PID_PORT9_10 - Port 9 + 10 / E
- TLV_PID_PORT11_12 - Port 11 + 12 / F
- TLV_PID_PORTU - Port U
- TLV_PID_PORTJ - Port J
- TLV_PID_TA2 - Timer A2
- TLV_PID_TA3 - Timer A1
- TLV_PID_TA5 - Timer A5
- TLV_PID_TA7 - Timer A7
- TLV_PID_TB3 - Timer B3
- TLV_PID_TB5 - Timer B5
- TLV_PID_TB7 - Timer B7
- TLV_PID_RTC - RTC
- TLV_PID_BT_RTC - BT + RTC
- TLV_PID_BBS - Battery Backup Switch
- TLV_PID_RTC_B - RTC_B
- TLV_PID_TD2 - Timer D2
- TLV_PID_TD3 - Timer D1
- TLV_PID_TD5 - Timer D5
- TLV_PID_TD7 - Timer D7
- TLV_PID_TEC - Timer Event Control
- TLV_PID_RTC_C - RTC_C
- TLV_PID_AES - AES
- TLV_PID_MPY16 - MPY16
- TLV_PID_MPY32 - MPY32
- TLV_PID_MPU - MPU
- TLV_PID_USCI_AB - USCI_AB
- TLV_PID_USCI_A - USCI_A
- TLV_PID_USCI_B - USCI_B
- TLV_PID_EUSCI_A - eUSCI_A
- TLV_PID_EUSCI_B - eUSCI_B
- TLV_PID_REF - Shared Reference
- TLV_PID_COMP_B - COMP_B
- TLV_PID_COMP_D - COMP_D
- TLV_PID_USB - USB
- TLV_PID_LCD_B - LCD_B
- TLV_PID_LCD_C - LCD_C
- TLV_PID_DAC12_A - DAC12_A
- TLV_PID_SD16_B_1 - SD16_B 1 Channel
- TLV_PID_SD16_B_2 - SD16_B 2 Channel
- TLV_PID_SD16_B_3 - SD16_B 3 Channel
- TLV_PID_SD16_B_4 - SD16_B 4 Channel

- TLV_PID_SD16_B_5 - SD16_B 5 Channel
- TLV_PID_SD16_B_6 - SD16_B 6 Channel
- TLV_PID_SD16_B_7 - SD16_B 7 Channel
- TLV_PID_SD16_B_8 - SD16_B 8 Channel
- TLV_PID_ADC12_A - ADC12_A
- TLV_PID_ADC10_A - ADC10_A
- TLV_PID_ADC10_B - ADC10_B
- TLV_PID_SD16_A - SD16_A
- TLV_PID_TI_BSL - BSL

instance In some cases a specific tag may have more than one instance. For example a device may have more than a single USCI module, each of which is defined by an instance number 0, 1, 2, etc. When only one instance exists; 0 is passed.

Returns:

The returned value is zero if the specified tag value (peripheral) is not available in the device.

30.3 Programming Example

The following example shows some tlv operations using the APIs

```
struct s_TLV_Die_Record * pDIEREC;
unsigned char bDieRecord_bytes;

TLV_getInfo(TLV_TAG_DIERECORD,
            0,
            &bDieRecord_bytes,
            (unsigned int **)&pDIEREC
            );
```

31 WatchDog Timer (WDT_A)

Introduction	375
API Functions	375
Programming Example	379

31.1 Introduction

The Watchdog Timer (WDT_A) API provides a set of functions for using the MSP430Ware WDT_A modules. Functions are provided to initialize the Watchdog in either timer interval mode, or watchdog mode, with selectable clock sources and dividers to define the timer interval.

The WDT_A module can generate only 1 kind of interrupt in timer interval mode. If in watchdog mode, then the WDT_A module will assert a reset once the timer has finished.

This driver is contained in `wdt_a.c`, with `wdt_a.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the TI Compiler 4.2.1 compiler, IAR 5.51.6 compiler and MSPGCC 4.8.0 compiler with different optimization settings. Users may see different code sizes depending on their project settings so it is best to perform your benchmarks within your project. These sizes contain all functions of the peripheral but only functions that are used will be linked into the application and added to the total code size. To see individual API code metrics see the specific API below.

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	172
TI Compiler 4.2.1	Size	90
TI Compiler 4.2.1	Speed	90
IAR 5.51.6	None	140
IAR 5.51.6	Size	54
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	236
MSPGCC 4.8.0	Size	104
MSPGCC 4.8.0	Speed	104

31.2 API Functions

Functions

- void [WDT_A_hold](#) (uint16_t baseAddress)
- void [WDT_A_intervalTimerInit](#) (uint16_t baseAddress, uint8_t clockSelect, uint8_t clockDivider)
- void [WDT_A_resetTimer](#) (uint16_t baseAddress)
- void [WDT_A_start](#) (uint16_t baseAddress)
- void [WDT_A_watchdogTimerInit](#) (uint16_t baseAddress, uint8_t clockSelect, uint8_t clockDivider)

31.2.1 Detailed Description

The WDT_A API is one group that controls the WDT_A module.

- [WDT_A_hold\(\)](#)
- [WDT_A_start\(\)](#)
- [WDT_A_clearCounter\(\)](#)

- [WDT_A_watchdogTimerInit\(\)](#)
- [WDT_A_intervalTimerInit\(\)](#)

31.2.2 Function Documentation

31.2.2.1 void WDT_A_hold (uint16_t *baseAddress*)

Holds the Watchdog Timer.

This function stops the watchdog timer from running, that way no interrupt or PUC is asserted.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	30
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	50
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Parameters:

baseAddress is the base address of the WDT_A module.

Returns:

None

31.2.2.2 void WDT_A_intervalTimerInit (uint16_t *baseAddress*, uint8_t *clockSelect*, uint8_t *clockDivider*)

Sets the clock source for the Watchdog Timer in timer interval mode.

This function sets the watchdog timer as timer interval mode, which will assert an interrupt without causing a PUC.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	36
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	26
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Parameters:

baseAddress is the base address of the WDT_A module.

clockSelect is the clock source that the watchdog timer will use. Valid values are:

- **WDT_A_CLOCKSOURCE_SMCLK** [Default]
- **WDT_A_CLOCKSOURCE_ACLK**

- WDT_A_CLOCKSOURCE_VLOCLK
- WDT_A_CLOCKSOURCE_XCLK

Modified bits are **WDTSEL** of **WDTCTL** register.

clockDivider is the divider of the clock source, in turn setting the watchdog timer interval. Valid values are:

- WDT_A_CLOCKDIVIDER_2G
- WDT_A_CLOCKDIVIDER_128M
- WDT_A_CLOCKDIVIDER_8192K
- WDT_A_CLOCKDIVIDER_512K
- WDT_A_CLOCKDIVIDER_32K [Default]
- WDT_A_CLOCKDIVIDER_8192
- WDT_A_CLOCKDIVIDER_512
- WDT_A_CLOCKDIVIDER_64

Modified bits are **WDTIS** and **WDTHOLD** of **WDTCTL** register.

Returns:

None

31.2.2.3 void WDT_A_resetTimer (uint16_t *baseAddress*)

Resets the timer counter of the Watchdog Timer.

This function resets the watchdog timer to 0x0000h.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	32
TI Compiler 4.2.1	Size	18
TI Compiler 4.2.1	Speed	18
IAR 5.51.6	None	28
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Parameters:

baseAddress is the base address of the WDT_A module.

Returns:

None

31.2.2.4 void WDT_A_start (uint16_t *baseAddress*)

Starts the Watchdog Timer.

This function starts the watchdog timer functionality to start counting again.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	34
TI Compiler 4.2.1	Size	20
TI Compiler 4.2.1	Speed	20
IAR 5.51.6	None	30
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	50
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Parameters:

baseAddress is the base address of the WDT_A module.

Returns:

None

31.2.2.5 void WDT_A_watchdogTimerInit (uint16_t baseAddress, uint8_t clockSelect, uint8_t clockDivider)

Sets the clock source for the Watchdog Timer in watchdog mode.

This function sets the watchdog timer in watchdog mode, which will cause a PUC when the timer overflows. When in the mode, a PUC can be avoided with a call to [WDT_A_resetTimer\(\)](#) before the timer runs out.

Code Metrics:

Compiler	Optimization	Code Size
TI Compiler 4.2.1	None	36
TI Compiler 4.2.1	Size	16
TI Compiler 4.2.1	Speed	16
IAR 5.51.6	None	26
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Parameters:

baseAddress is the base address of the WDT_A module.

clockSelect is the clock source that the watchdog timer will use. Valid values are:

- WDT_A_CLOCKSOURCE_SMCLK [Default]
 - WDT_A_CLOCKSOURCE_ACLK
 - WDT_A_CLOCKSOURCE_VLOCLK
 - WDT_A_CLOCKSOURCE_XCLK
- Modified bits are WDTSSSEL of WDTCTL register.

clockDivider is the divider of the clock source, in turn setting the watchdog timer interval. Valid values are:

- WDT_A_CLOCKDIVIDER_2G
 - WDT_A_CLOCKDIVIDER_128M
 - WDT_A_CLOCKDIVIDER_8192K
 - WDT_A_CLOCKDIVIDER_512K
 - WDT_A_CLOCKDIVIDER_32K [Default]
 - WDT_A_CLOCKDIVIDER_8192
 - WDT_A_CLOCKDIVIDER_512
 - WDT_A_CLOCKDIVIDER_64
- Modified bits are WDTIS and WDT_HOLD of WDTCTL register.

Returns:

None

31.3 Programming Example

The following example shows how to initialize and use the WDT_A API to interrupt about every 32 ms, toggling the LED in the ISR.

```
//Initialize WDT_A module in timer interval mode,
//with SMCLK as source at an interval of 32 ms.
WDT_A_intervalTimerInit(WDT_A_BASE,
    WDT_A_CLOCKSOURCE_SMCLK,
    WDT_A_CLOCKDIVIDER_32K);

//Enable Watchdog Interrupt
SFR_enableInterrupt(SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT);

//Set P1.0 to output direction
GPIO_setAsOutputPin(
    GPIO_PORT_P1,
    GPIO_PIN0
);

//Enter LPM0, enable interrupts
__bis_SR_register(LPM0_bits + GIE);
//For debugger
__no_operation();
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products

Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
RF/IF and ZigBee® Solutions	www.ti.com/lprf

Applications

Audio	www.ti.com/audio
Automotive	www.ti.com/automotive
Broadband	www.ti.com/broadband
Digital Control	www.ti.com/digitalcontrol
Medical	www.ti.com/medical
Military	www.ti.com/military
Optical Networking	www.ti.com/opticalnetwork
Security	www.ti.com/security
Telephony	www.ti.com/telephony
Video & Imaging	www.ti.com/video
Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2014, Texas Instruments Incorporated