MSP430® Peripheral Driver Library for FR57xx Devices

USER'S GUIDE

Copyright © 2014 Texas Instruments Incorporated.

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 1.80.00.18 of this document, last updated on 2014-03-06.

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 14
4 How to include driverlib into your existing project 16
5 10-Bit Analog-to-Digital Converter (ADC10_B) 18
5.1 Introduction 18
5.2 API Functions 19
5.3 Programming Example 40
6 Comparator (COMP_D) 41
6.1 Introduction 41
6.2 API Functions 41
6.3 Programming Example 54
7 Cyclical Redundancy Check (CRC) 56
7.1 Introduction 56
7.2 API Functions 56
7.3 Programming Example 61
8 Clock System (CS) 62
8.1 Introduction 62
8.2 API Functions 63
8.3 Programming Example 64
9 Direct Memory Access (DMA) 65
9.1 Introduction 65
9.2 API Functions 65
9.3 Programming Example 80
10 EUSCI Universal Asynchronous Receiver/Transmitter (EUSCI_A_UART) 81
10.1 Introduction 81
10.2 API Functions 81
10.3 Programming Example 94
11 EUSCI Synchronous Peripheral Interface (EUSCI_A_SPI) 95
11.1 Introduction 95
11.2 Functions 95
11.3 Programming Example 107
12 EUSCI Synchronous Peripheral Interface (EUSCI_B_SPI) 108
12.1 Introduction 108
12.2 Functions 108
12.3 Programming Example 120
13 EUSCI Inter-Integrated Circuit (EUSCI_B_I2C) 121
13.1 Introduction 121
13.2 API Functions 123
13.3 Programming Example 148
14 FRAM Controller 150
14.1 Introduction 150

14.2 API Functions . . . 150
14.3 Programming Example . . . 156
15 GPIO . . . 157
15.1 Introduction . . . 157
15.2 API Functions . . . 158
15.3 Programming Example . . . 175
16 Memory Protection Unit (MPU) . . . 177
16.1 Introduction . . . 177
16.2 API Functions . . . 177
16.3 Programming Example . . . 183
17 32-Bit Hardware Multiplier (MPY32) . . . 184
17.1 Introduction . . . 184
17.2 API Functions . . . 184
17.3 Programming Example . . . 196
18 Power Management Module (PMM) . . . 197
18.1 Introduction . . . 197
18.2 API Functions . . . 197
18.3 Programming Example . . . 204
19 Internal Reference (REF) . . . 206
19.1 Introduction . . . 206
19.2 API Functions . . . 206
19.3 Programming Example . . . 213
20 Real-Time Clock (RTC_B) . . . 214
20.1 Introduction . . . 214
20.2 API Functions . . . 214
20.3 Programming Example . . . 227
21 SFR Module . . . 229
21.1 Introduction . . . 229
21.2 API Functions . . . 229
21.3 Programming Example . . . 234
22 SYS Module . . . 236
22.1 Introduction . . . 236
22.2 API Functions . . . 236
22.3 Programming Example . . . 247
23 TIMER_A . . . 248
23.1 Introduction . . . 248
23.2 API Functions . . . 249
23.3 Programming Example . . . 273
24 TIMER_B . . . 275
24.1 Introduction . . . 275
24.2 API Functions . . . 276
24.3 Programming Example . . . 303
25 Tag Length Value . . . 304
25.1 Introduction . . . 304
25.2 API Functions . . . 304
25.3 Programming Example . . . 309
26 WatchDog Timer (WDT_A) . . . 310
26.1 Introduction . . . 310

26.2 API Functions 310
26.3 Programming Example 314
IMPORTANT NOTICE 315

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:

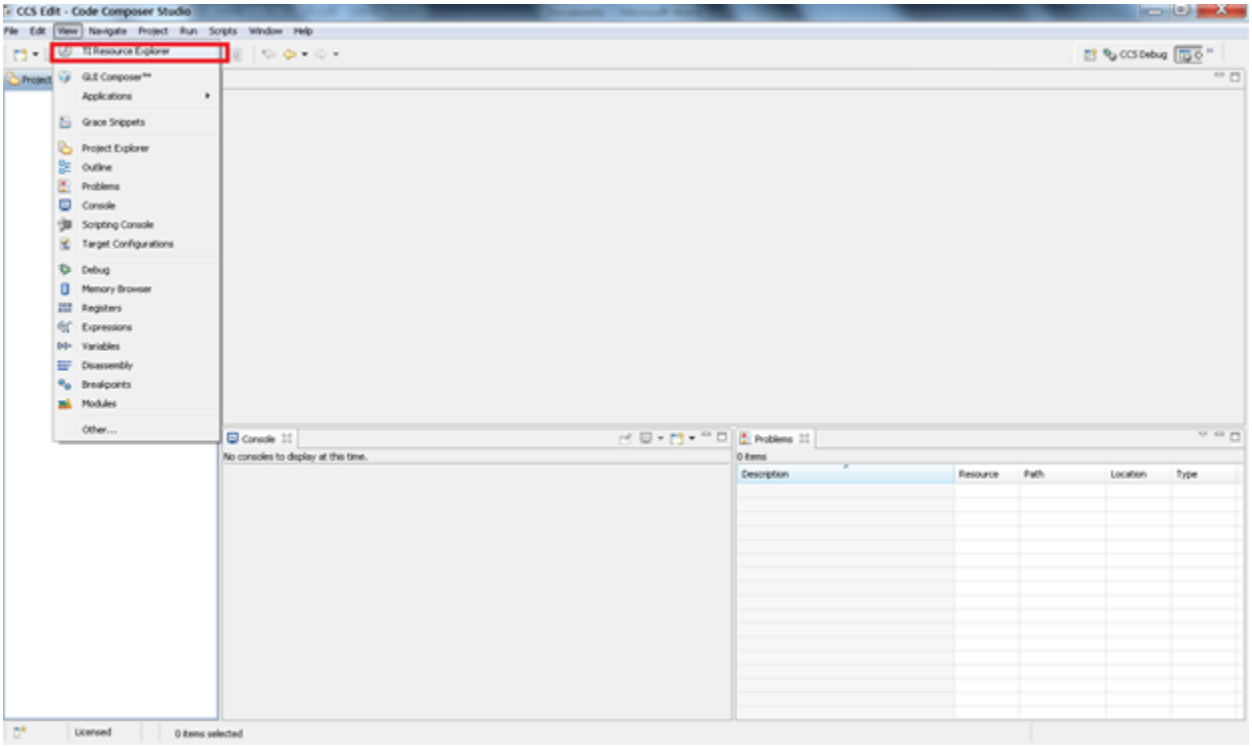
- IAR Embedded Workbench®
- Texas Instruments Code Composer Studio™

Using assert statements to debug

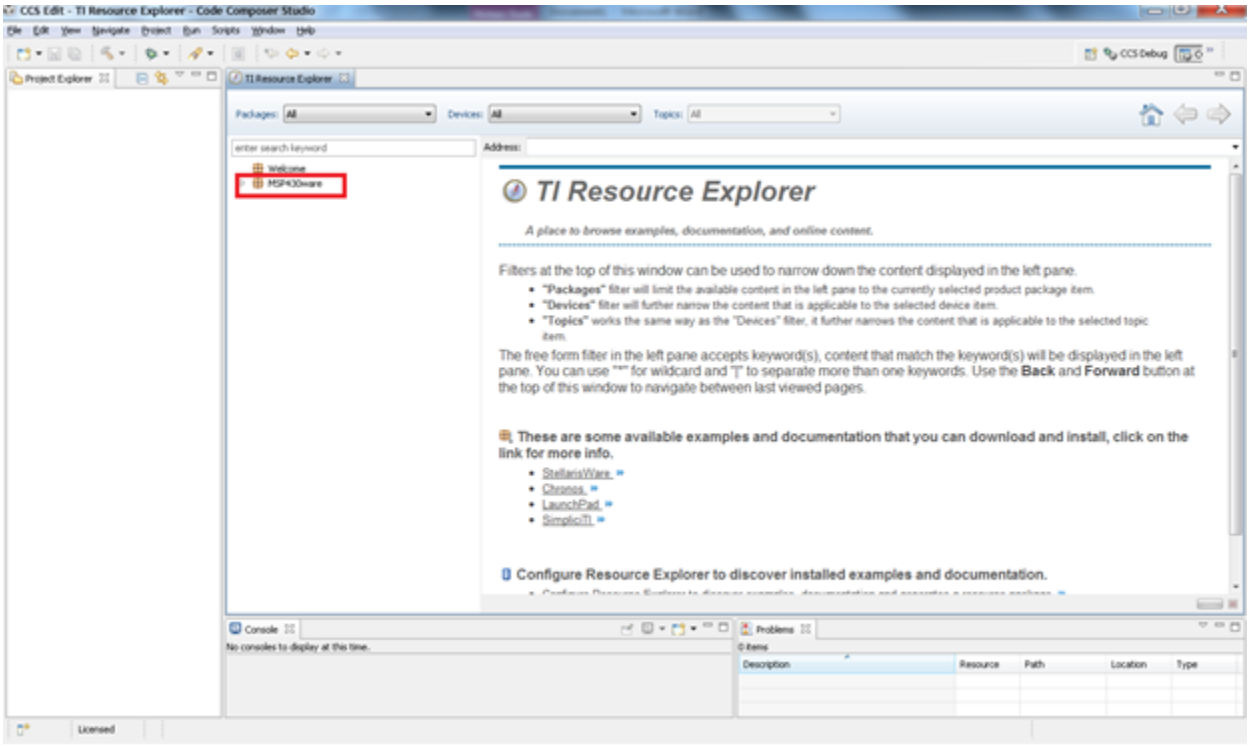
Assert statements are disabled by default. To enable the assert statement edit the hw_regaccess.h file in the inc folder. Comment out the statement define NDEBUG -> //define NDEBUG Asserts in CCS work only if the project is optimized for size.

2 Navigating to driverlib through CCS Resource Explorer

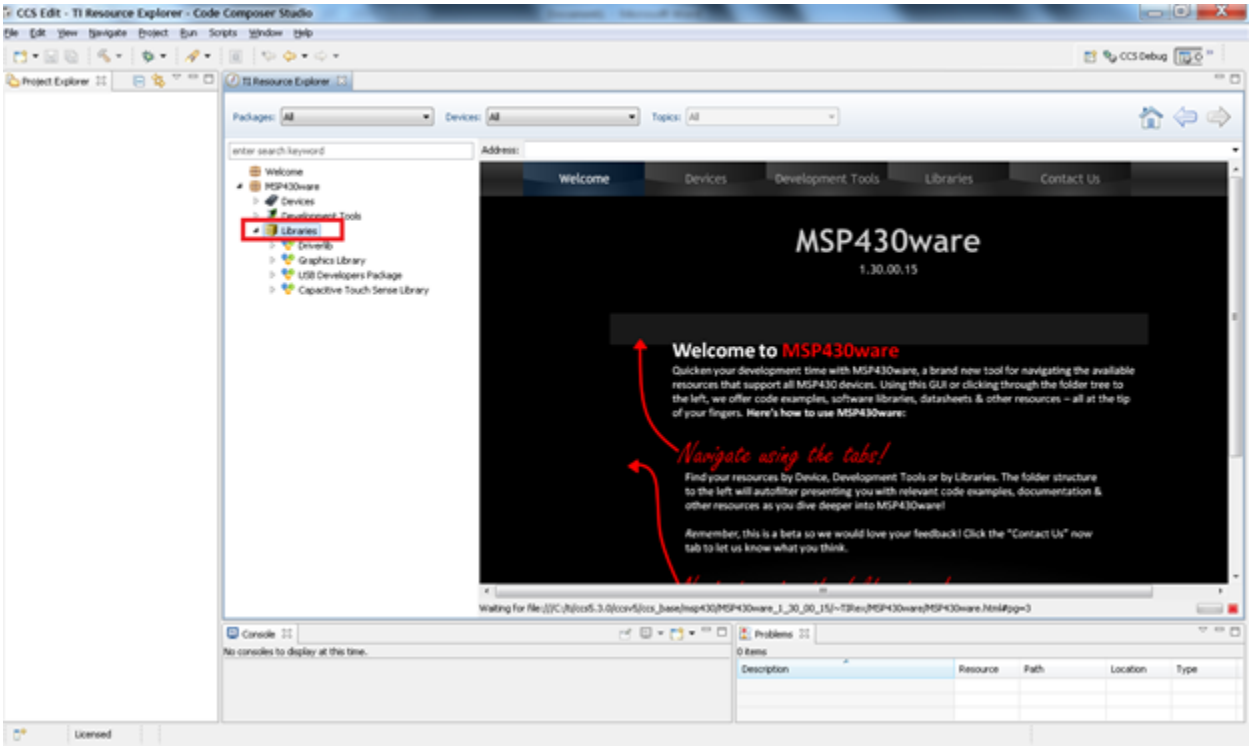
In CCS, click View->TI Resource Explorer

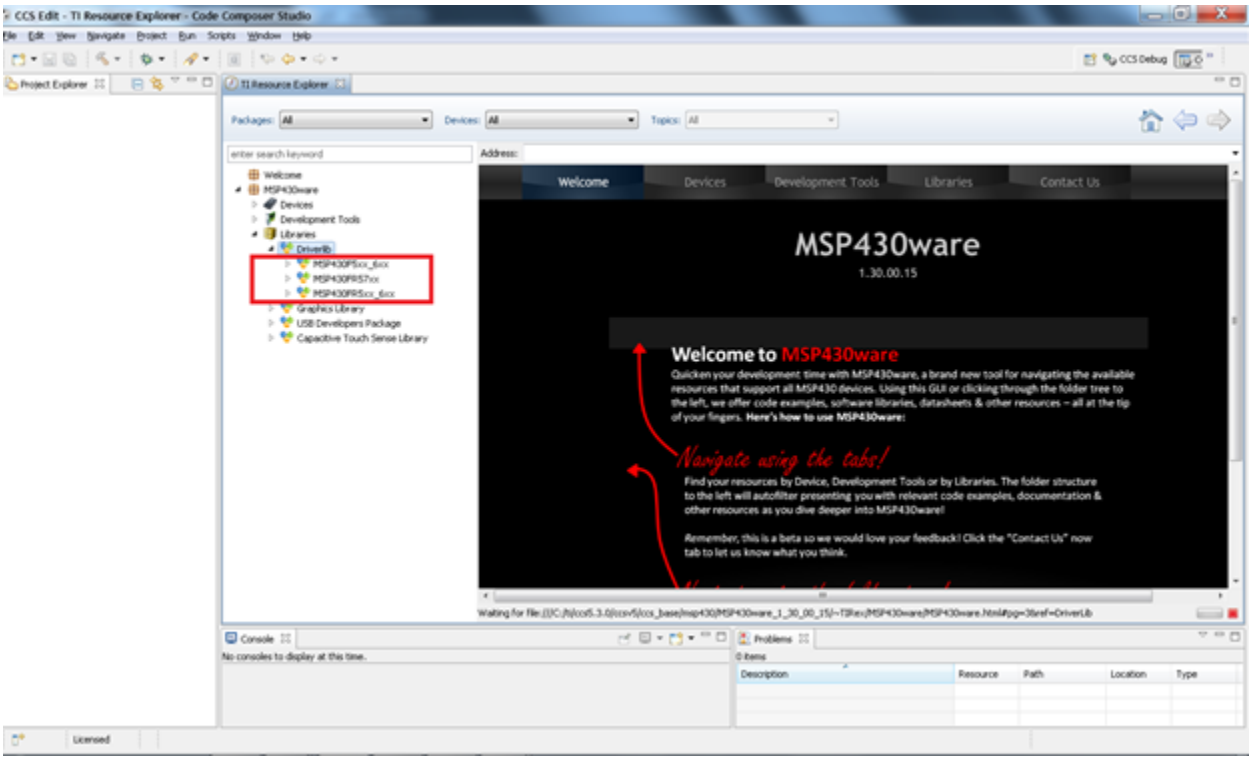


In Resource Explorer View, click on MSP430ware

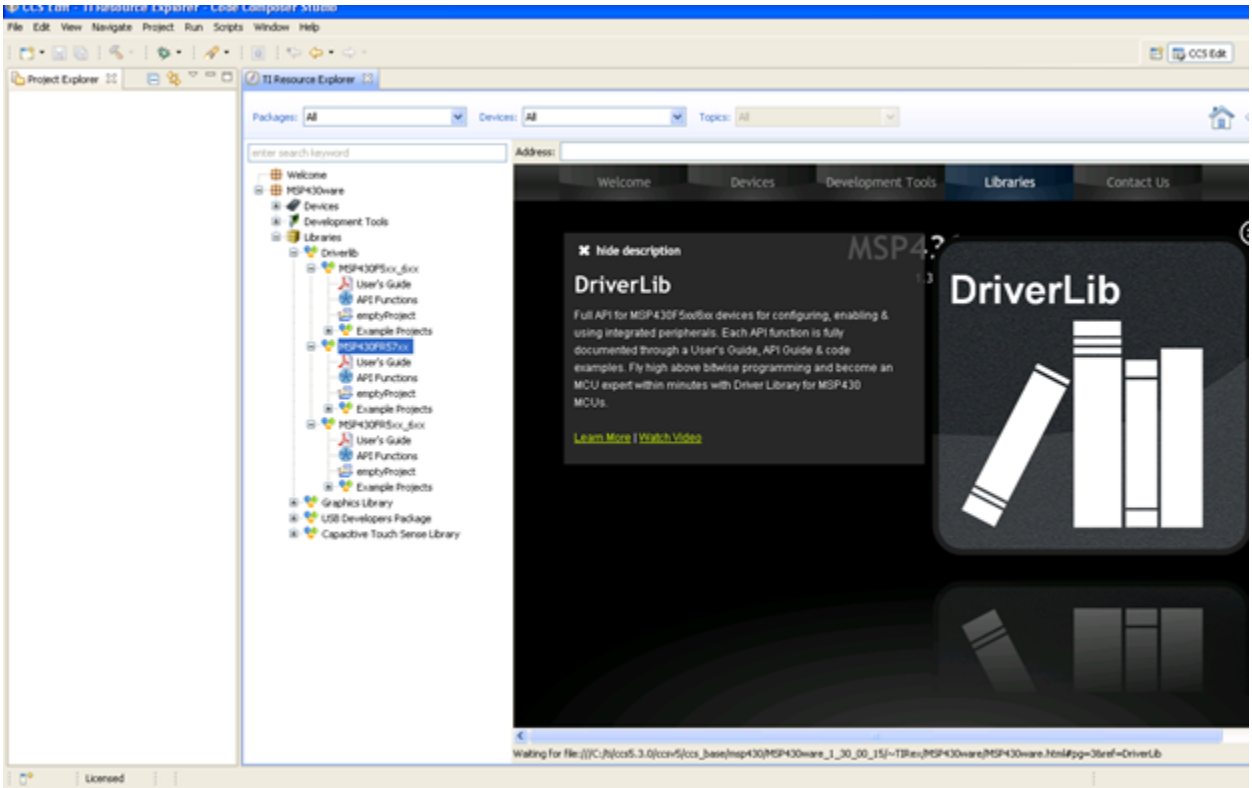


Clicking MSP430ware takes you to the introductory page. The version of the latest MSP430ware installed is available in this page. In this screenshot the version is 1.30.00.15 The various software, collateral, code examples, datasheets and user guides can be navigated by clicking the different topics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

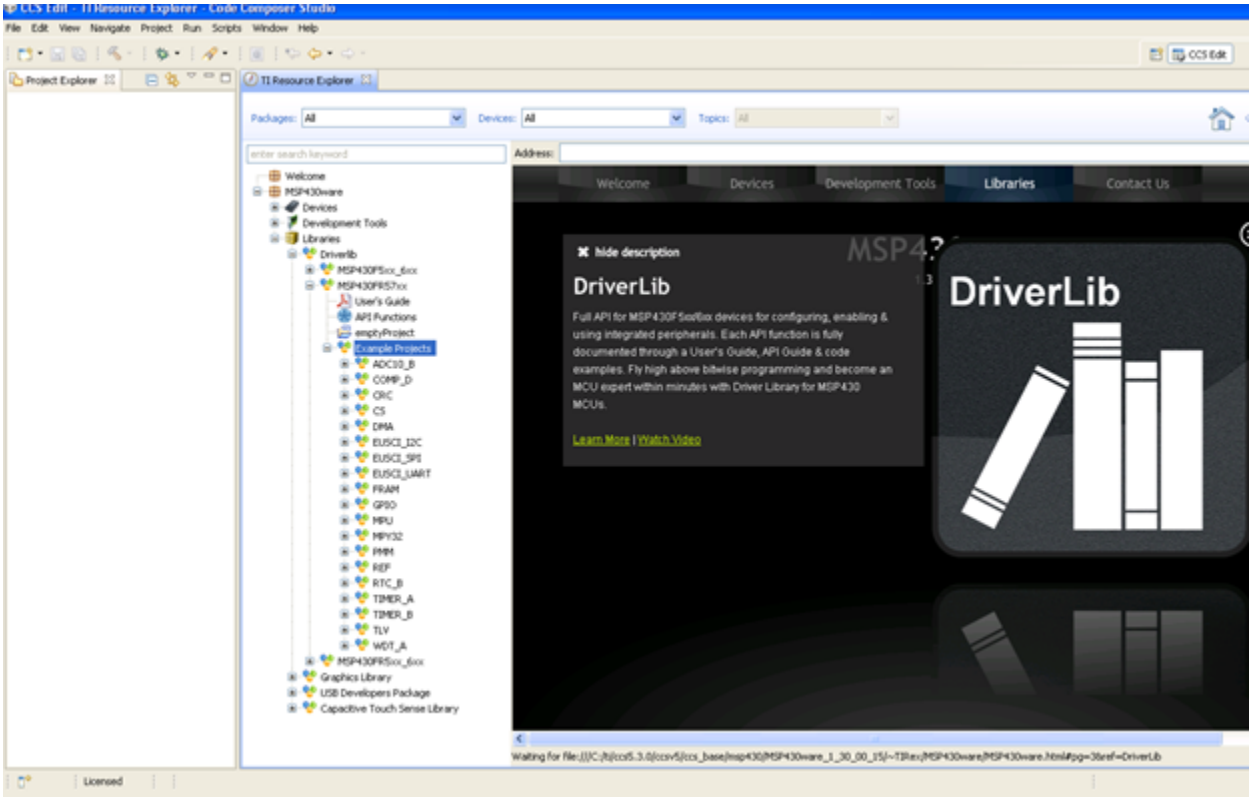




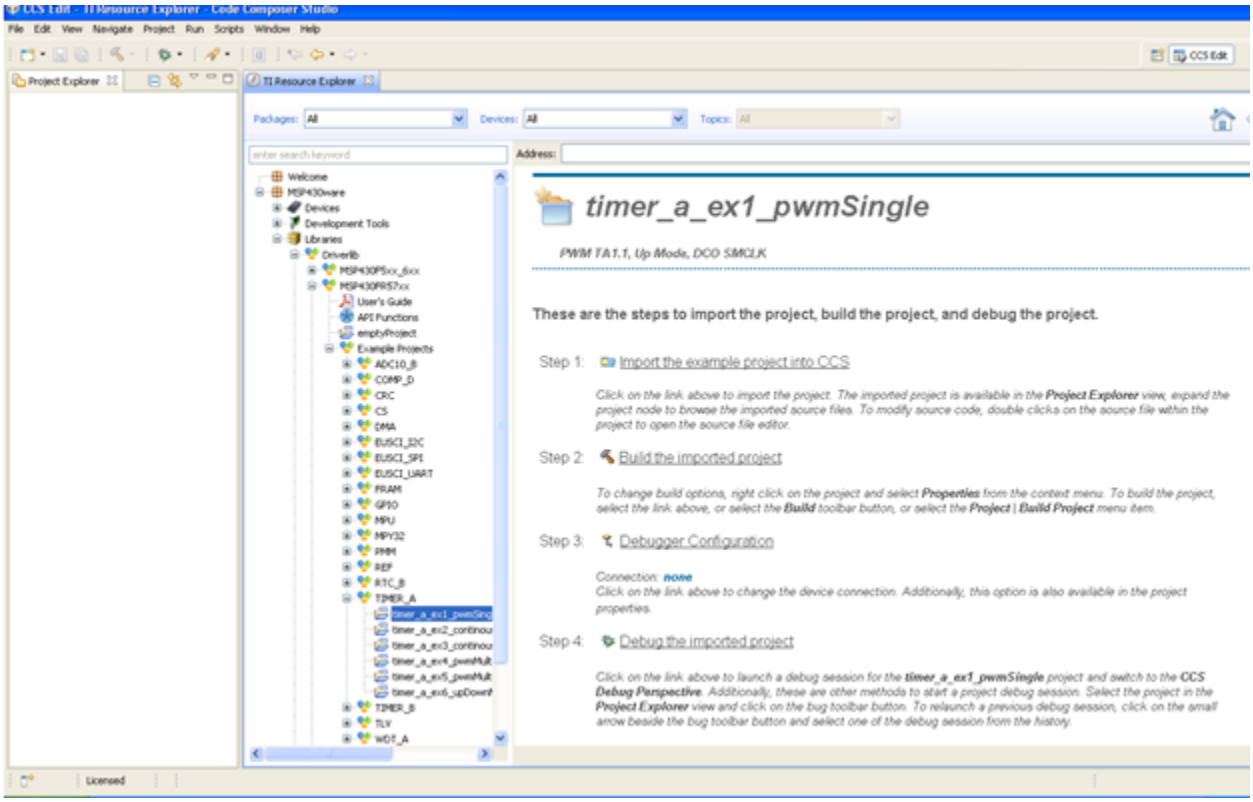
Driverlib is designed per Family. If a common device family userŠs guide exists for a group of devices, these devices belong to the same ŚfamilyŠ. Currently driverlib is available for the following family of devices. MSP430F5xx_6xx MSP430FR57xx MSP430FR5xx_6xx



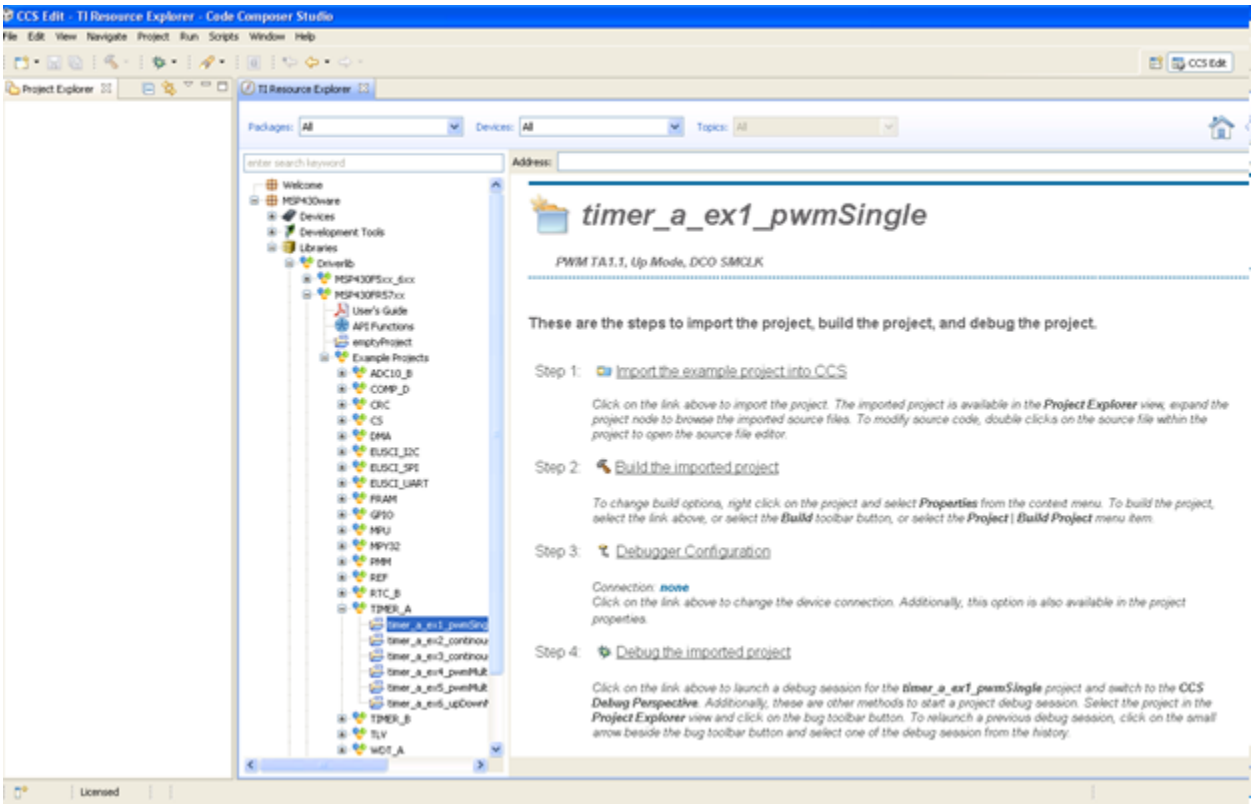
Click on the MSP430FR5xx_6xx to navigate to the driverlib based example code for that family.



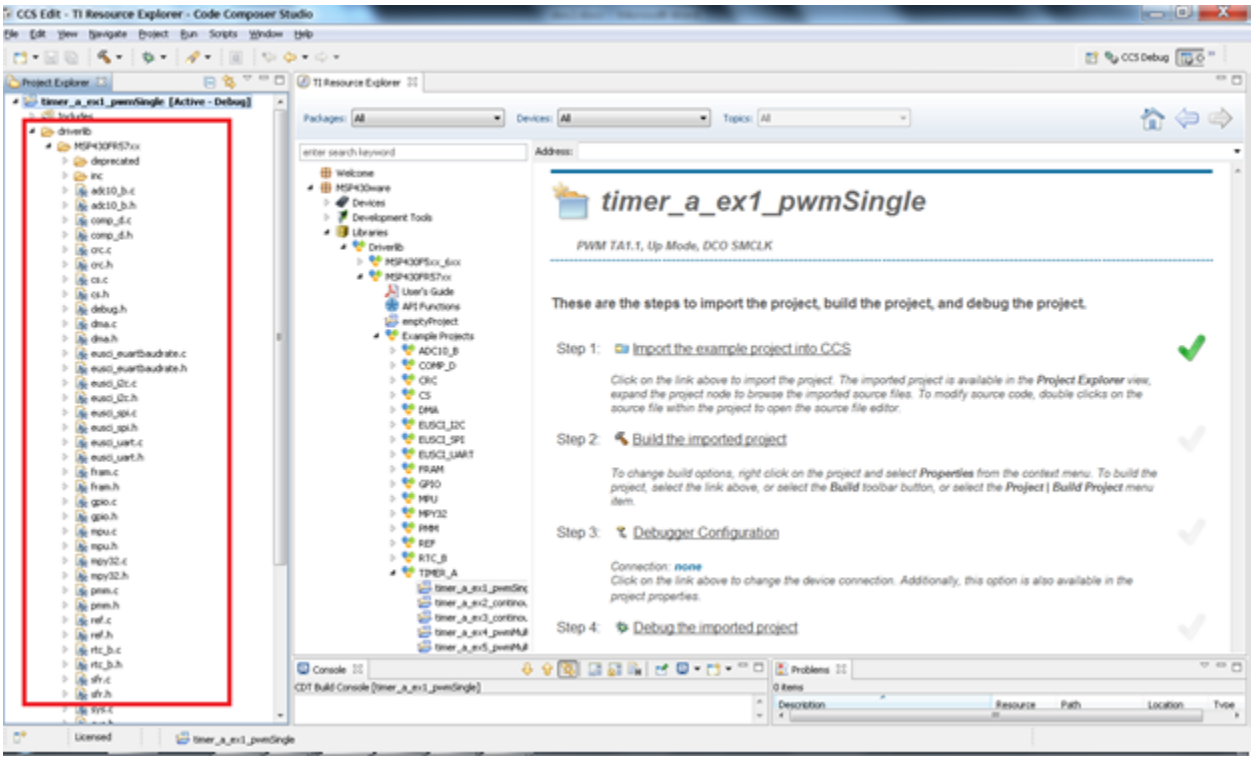
The various peripherals are listed in alphabetical order. The names of peripherals are as in device family userŠs guide. Clicking on a peripheral name lists the driverlib example code for that peripheral. The screenshot below shows an example when the user clicks on GPIO peripheral.



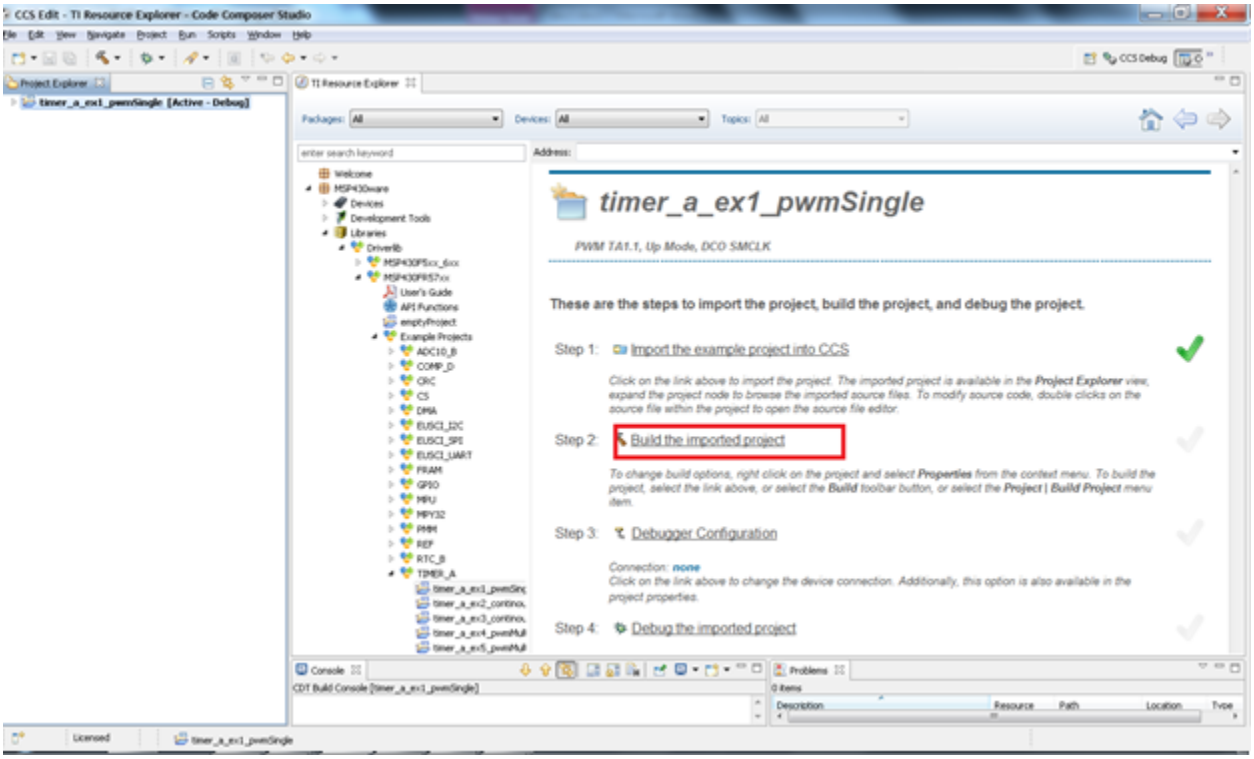
Now click on the specific example you are interested in. On the right side there are options to Import/Build/Download and Debug. Import the project by clicking on the "Import the example project into CCS"



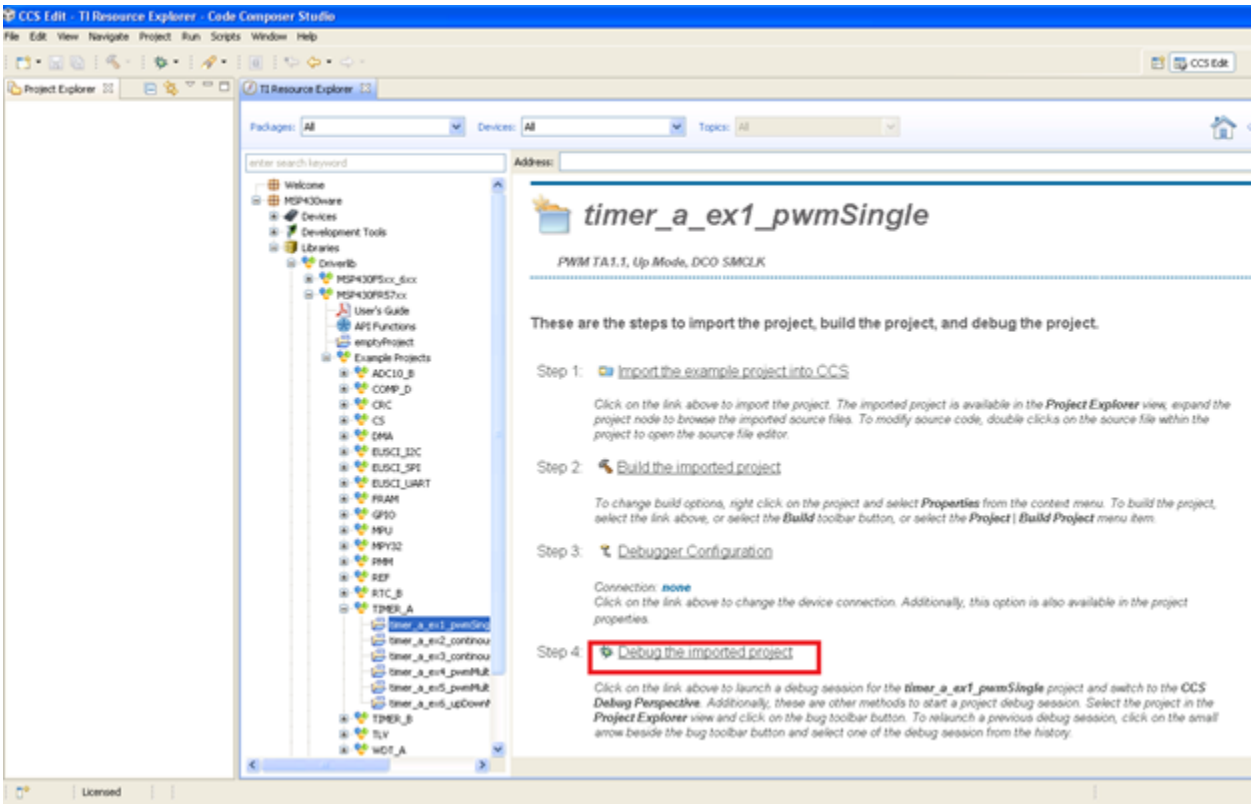
The imported project can be viewed on the left in the Project Explorer. All required driverlib source and header files are included inside the driverlib folder. All driverlib source and header files are linked to the example projects. So if the user modifies any of these source or header files, the original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

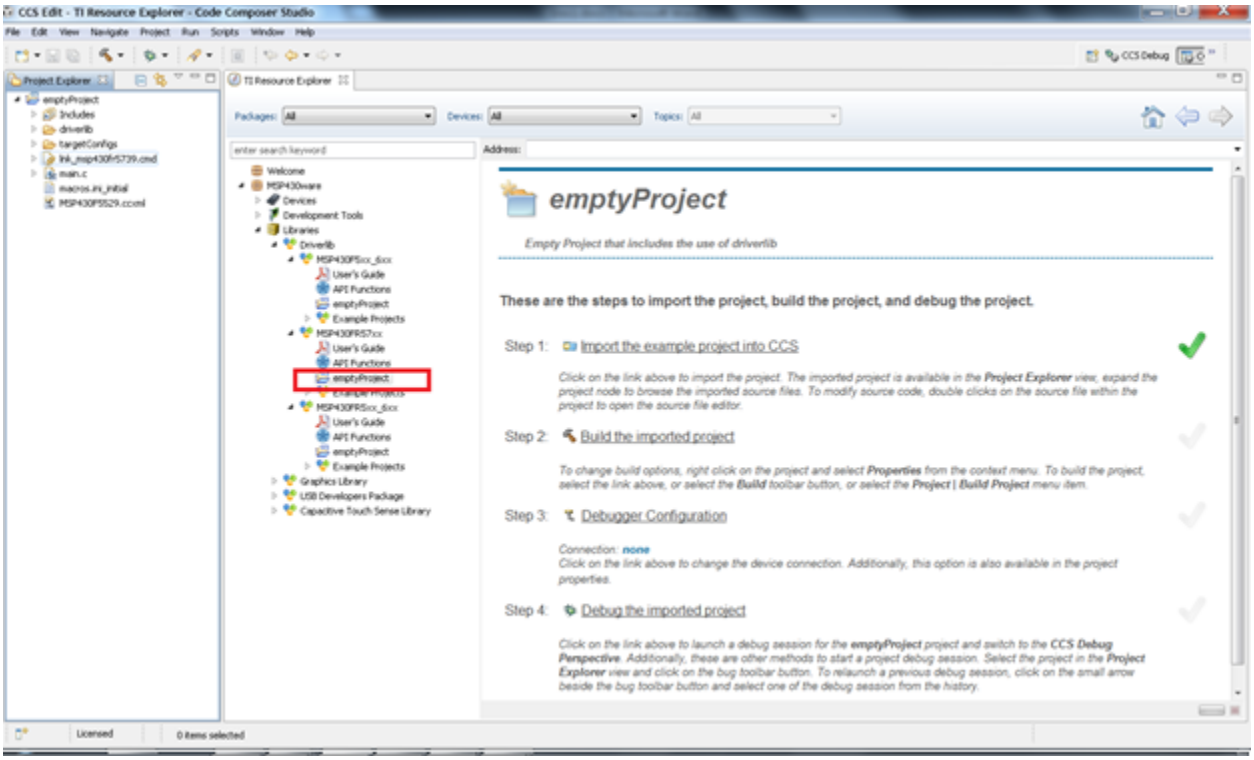


Now click on Build the imported project on the right to build the example project.



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.

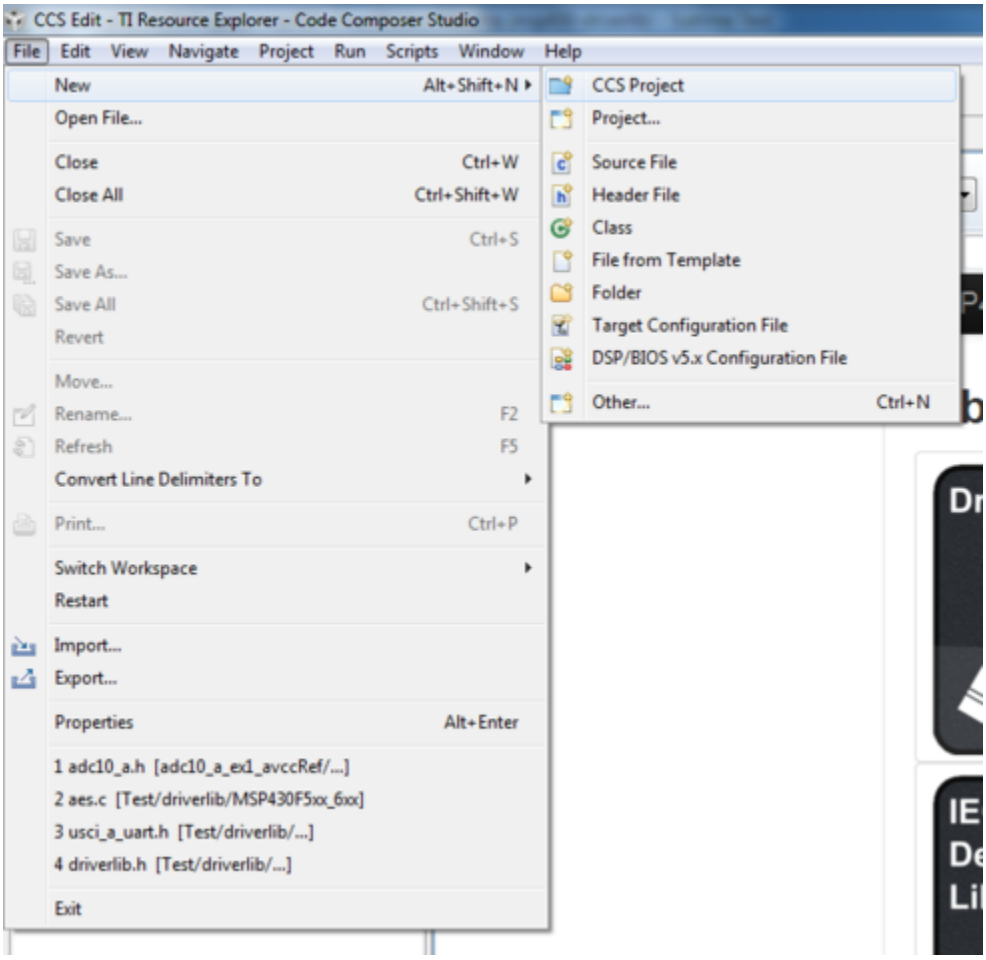


The main.c included with the empty project can be modified to include user code.

3 How to create a new user project that uses Driverlib

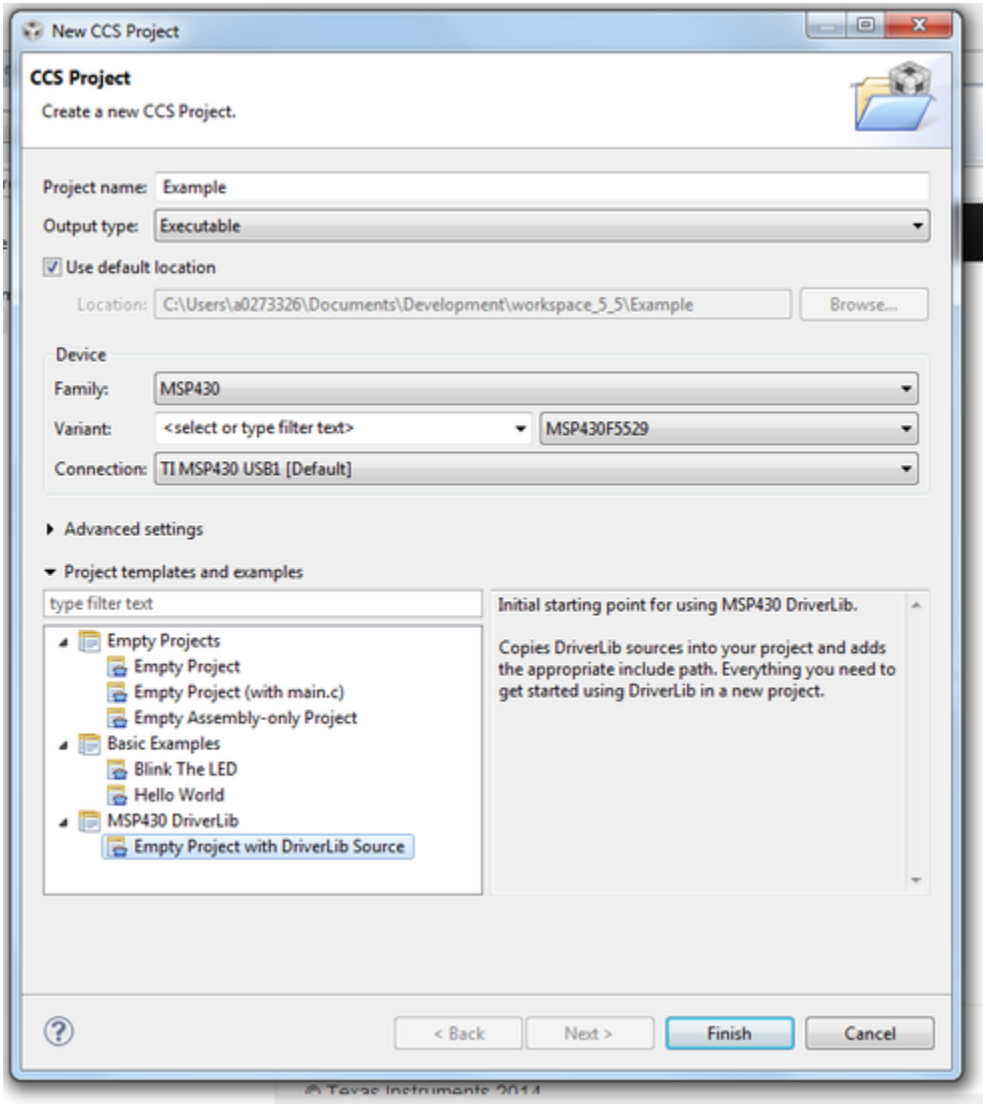
To get started on a new project we recommend using the new project wizard. For driver library to work with the new project wizard CCS must have discovered the driver library RTSC product. For more information refer to the installation steps of the release notes. The new project wizard adds the needed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

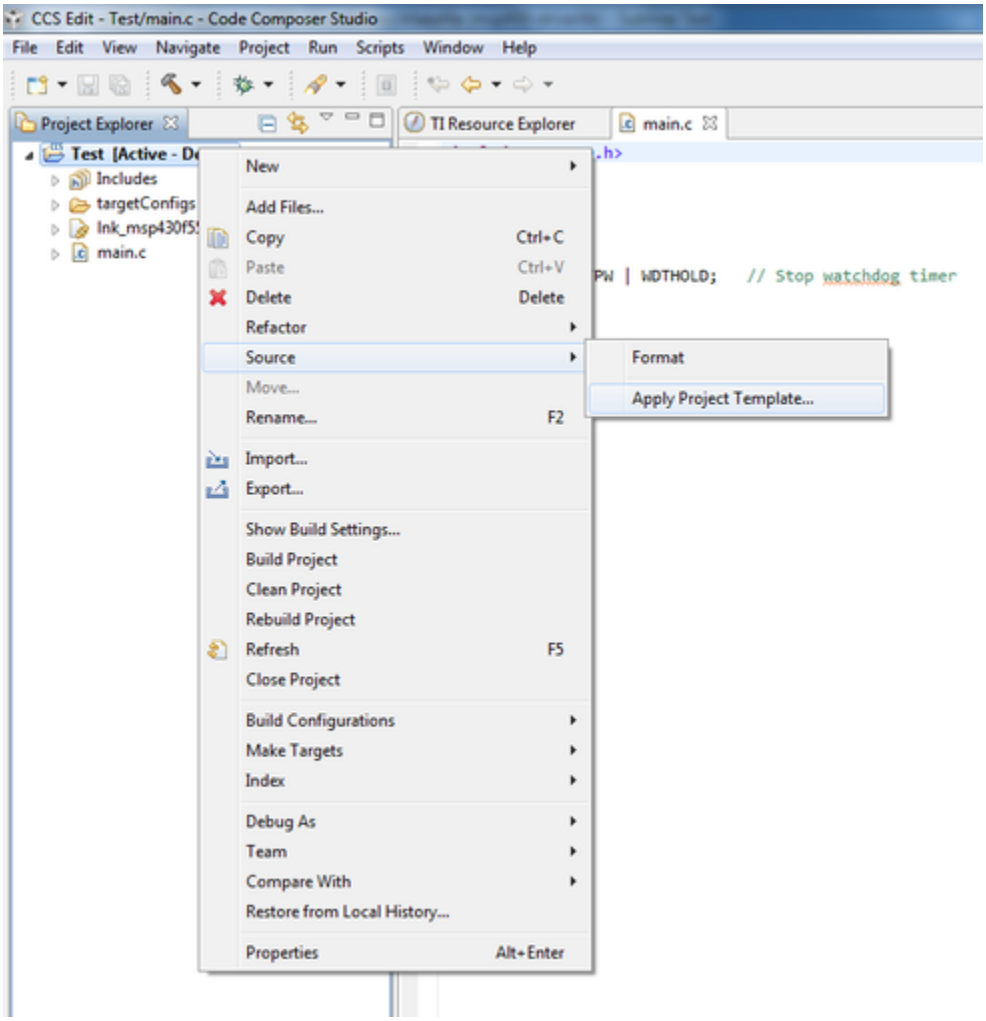


Finally click "Finish" and begin developing with your Driver Library enabled project.

4 How to include driverlib into your existing project

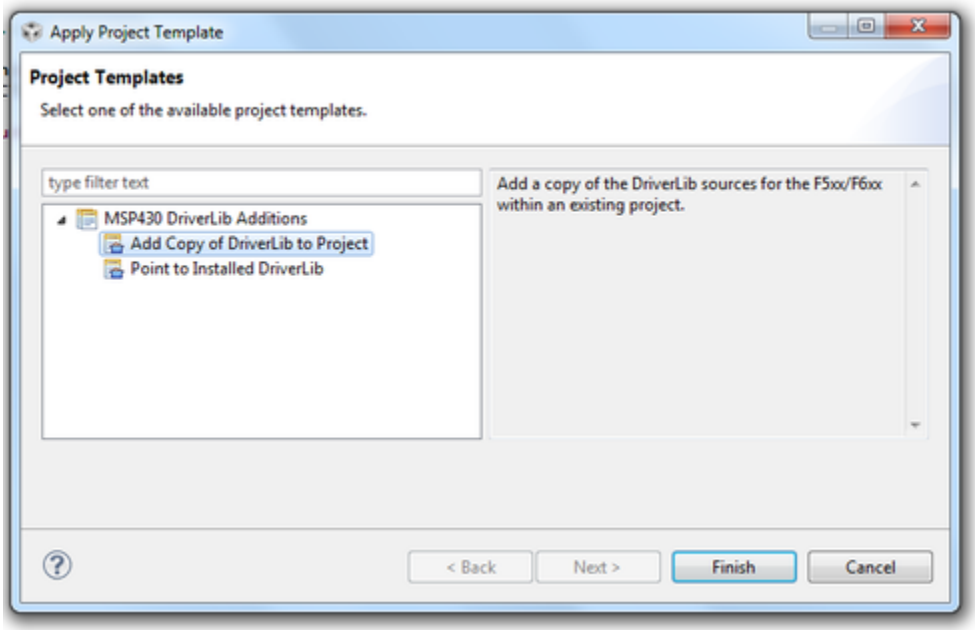
To add driver library to an existing project we recommend using CCS project templates. For driver library to work with project templates CCS must have discovered the driver library RTSC product. For more information refer to the installation steps of the release notes. CCS project templates adds the 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 advandced 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 10-Bit Analog-to-Digital Converter (ADC10_B)

Introduction 18
API Functions 19
Programming Example 40

5.1 Introduction

The 10-Bit Analog-to-Digital (ADC10_B) API provides a set of functions for using the MSP430Ware ADC10_B modules. Functions are provided to initialize the ADC10_B modules, setup signal sources and reference voltages, and manage interrupts for the ADC10_B modules.

The ADC10_B module supports fast 10-bit analog-to-digital conversions. The module implements a 10-bit SAR core together, sample select control and a window comparator.

ADC10_B features include:

- Greater than 200-ksps maximum conversion rate
- Monotonic 10-bit converter with no missing codes
- Sample-and-hold with programmable sampling periods controlled by software or timers
- Conversion initiation by software or different timers
- Software-selectable on chip reference using the REF module or external reference
- Twelve individually configurable external input channels
- Conversion channel for temperature sensor of the REF module
- Selectable conversion clock source
- Single-channel, repeat-single-channel, sequence, and repeat-sequence conversion modes
- Window comparator for low-power monitoring of input signals
- Interrupt vector register for fast decoding of six ADC interrupts (ADC10IFG0, ADC10TOVIFG, ADC10OVIFG, ADC10LOIFG, ADC10INIFG, ADC10HIIFG)

This driver is contained in `adc10_b.c`, with `adc10_b.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the CCS 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 withing 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
CCS 4.2.1	None	846
CCS 4.2.1	Size	294
CCS 4.2.1	Speed	292
IAR 5.51.6	None	448
IAR 5.51.6	Size	338
IAR 5.51.6	Speed	338
MSPGCC 4.8.0	None	1202
MSPGCC 4.8.0	Size	336
MSPGCC 4.8.0	Speed	338

5.2 API Functions

Functions

- void ADC10_B_clearInterrupt (uint32_t baseAddress, uint8_t interruptFlagMask)
- void ADC10_B_disable (uint32_t baseAddress)
- void ADC10_B_disableConversions (uint32_t baseAddress, bool preempt)
- void ADC10_B_disableInterrupt (uint32_t baseAddress, uint8_t interruptMask)
- void ADC10_B_disableReferenceBurst (uint32_t baseAddress)
- void ADC10_B_disableSamplingTimer (uint32_t baseAddress)
- void ADC10_B_enable (uint32_t baseAddress)
- void ADC10_B_enableInterrupt (uint32_t baseAddress, uint8_t interruptMask)
- void ADC10_B_enableReferenceBurst (uint32_t baseAddress)
- uint8_t ADC10_B_getInterruptStatus (uint32_t baseAddress, uint8_t interruptFlagMask)
- uint32_t ADC10_B_getMemoryAddressForDMA (uint32_t baseAddress)
- uint16_t ADC10_B_getResults (uint32_t baseAddress)
- bool ADC10_B_init (uint32_t baseAddress, uint16_t sampleHoldSignalSourceSelect, uint8_t clockSourceSelect, uint16_t clockSourceDivider)
- uint8_t ADC10_B_isBusy (uint32_t baseAddress)
- void ADC10_B_memoryConfigure (uint32_t baseAddress, uint8_t inputSourceSelect, uint8_t positiveRefVoltageSourceSelect, uint8_t negativeRefVoltageSourceSelect)
- void ADC10_B_setDataReadBackFormat (uint32_t baseAddress, uint16_t readBackFormat)
- void ADC10_B_setReferenceBufferSamplingRate (uint32_t baseAddress, uint16_t samplingRateSelect)
- void ADC10_B_setResolution (uint32_t baseAddress, uint8_t resolutionSelect)
- void ADC10_B_setSampleHoldSignalInversion (uint32_t baseAddress, uint16_t invertedSignal)
- void ADC10_B_setupSamplingTimer (uint32_t baseAddress, uint16_t clockCycleHoldCount, uint16_t multipleSamplesEnabled)
- void ADC10_B_setWindowComp (uint32_t baseAddress, uint16_t highThreshold, uint16_t lowThreshold)
- void ADC10_B_startConversion (uint32_t baseAddress, uint8_t conversionSequenceModeSelect)

5.2.1 Detailed Description

The ADC10_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 ADC10.

The ADC10_B initialization and conversion functions are

- ADC10_B_init()
- ADC10_B_memoryConfigure()
- ADC10_B_setupSamplingTimer()
- ADC10_B_disableSamplingTimer()
- ADC10_B_setWindowComp()
- ADC10_B_startConversion()
- ADC10_B_disableConversions()
- ADC10_B_getResults()
- ADC10_B_isBusy()

The ADC10_B interrupts are handled by

- ADC10_B_enableInterrupt()
- ADC10_B_disableInterrupt()
- ADC10_B_clearInterrupt()
- ADC10_B_getInterruptStatus()

Auxiliary features of the ADC10_B are handled by

- ADC10_B_setResolution()
- ADC10_B_setSampleHoldSignalInversion()
- ADC10_B_setDataReadBackFormat()
- ADC10_B_enableReferenceBurst()
- ADC10_B_disableReferenceBurst()
- ADC10_B_setReferenceBufferSamplingRate()
- ADC10_B_getMemoryAddressForDMA()
- ADC10_B_enable()
- ADC10_B_disable()

5.2.2 Function Documentation

5.2.2.1 ADC10_B_clearInterrupt

Clears ADC10B selected interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	54
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
ADC10_B_clearInterrupt(uint32_t baseAddress,
                       uint8_t interruptFlagMask)
```

Description:

The selected ADC10B interrupt flags are cleared, so that it no longer asserts. The memory buffer interrupt flags are only cleared when the memory buffer is accessed.

Parameters:

baseAddress is the base address of the ADC10B module.

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

- **ADC10_B_OVIFG** - Interrupt flag for when a new conversion is about to overwrite the previous one
- **ADC10_B_TOVIFG** - Interrupt flag for when a new conversion is starting before the previous one has finished
- **ADC10_B_HIIFG** - Interrupt flag for when the input signal has gone above the high threshold of the window comparator
- **ADC10_B_LOIFG** - Interrupt flag for when the input signal has gone below the low threshold of the window comparator
- **ADC10_B_INIFG** - Interrupt flag for when the input signal is in between the high and low thresholds of the window comparator
- **ADC10_B_IFG0** - Interrupt flag for new conversion data in the memory buffer

Modified bits of **ADC10IFG** register.

Returns:

None

5.2.2.2 ADC10_B_disable

Disables the ADC10B block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
ADC10_B_disable(uint32_t baseAddress)
```

Description:

This will disable operation of the ADC10B block.

Parameters:

baseAddress is the base address of the ADC10B module.

Modified bits are **ADC10ON** of **ADC10CTL0** register.

Returns:

None

5.2.2.3 ADC10_B_disableConversions

Disables the ADC from converting any more signals.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	68
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	40
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	94
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	34

Prototype:

```
void
ADC10_B_disableConversions(uint32_t baseAddress,
                           bool preempt)
```

Description:

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 ADC10_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 ADC10CTL1 and ADC10CTL0

Parameters:

baseAddress is the base address of the ADC10B module.

preempt specifies if the current conversion should be pre-empted stopped before the end of the conversion Valid values are:

- **ADC10_B_COMPLETECONVERSION** - Allows the ADC10B to end the current conversion before disabling conversions.
- **ADC10_B_PREEMPTCONVERSION** - Stops the ADC10B immediately, with unpredicatble results of the current conversion. Cannot be used with repeated conversion.

Modified bits of **ADC10CTL1** register and bits of **ADC10CTL0** register.

Returns:

None

5.2.2.4 ADC10_B_disableInterrupt

Disables selected ADC10B interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	54
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
ADC10_B_disableInterrupt(uint32_t baseAddress,
                         uint8_t interruptMask)
```

Description:

Disables the indicated ADC10B interrupt sources. 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 ADC10B module.

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:

- **ADC10_B_OVIE** - Interrupts when a new conversion is about to overwrite the previous one
- **ADC10_B_TOVIE** - Interrupts when a new conversion is starting before the previous one has finished
- **ADC10_B_HIIE** - Interrupts when the input signal has gone above the high threshold of the window comparator
- **ADC10_B_LOIE** - Interrupts when the input signal has gone below the low threshold of the low window comparator
- **ADC10_B_INIE** - Interrupts when the input signal is in between the high and low thresholds of the window comparator
- **ADC10_B_IE0** - Interrupt for new conversion data in the memory buffer

Modified bits of **ADC10IE** register.

Returns:

None

5.2.2.5 ADC10_B_disableReferenceBurst

Disables the reference buffer's burst ability.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	6
CCS 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	30
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
ADC10_B_disableReferenceBurst(uint32_t baseAddress)
```

Description:

Disables the reference buffer's burst ability, forcing the reference buffer to remain on continuously.

Parameters:

baseAddress is the base address of the ADC10B module.

Modified bits are **ADC10REFBURST** of **ADC10CTL2** register.

Returns:
None

5.2.2.6 ADC10_B_disableSamplingTimer

Disables Sampling Timer Pulse Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	8
CCS 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

Prototype:
```
void
ADC10_B_disableSamplingTimer(uint32_t baseAddress)
```

Description:
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.

Parameters:
baseAddress is the base address of the ADC10B module.

Modified bits are **ADC10SHP** of **ADC10CTL1** register.

Returns:
None

5.2.2.7 ADC10_B_enable

Enables the ADC10B block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
ADC10_B_enable(uint32_t baseAddress)
```

Description:

This will enable operation of the ADC10B block.

Parameters:

baseAddress is the base address of the ADC10B module.

Modified bits are **ADC10ON** of **ADC10CTL0** register.

Returns:

None

5.2.2.8 ADC10_B_enableInterrupt

Enables selected ADC10B interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	50
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
ADC10_B_enableInterrupt(uint32_t baseAddress,
                        uint8_t interruptMask)
```

Description:

Enables the indicated ADC10B 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 ADC10B module.

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

- **ADC10_B_OVIE** - Interrupts when a new conversion is about to overwrite the previous one
- **ADC10_B_TOVIE** - Interrupts when a new conversion is starting before the previous one has finished
- **ADC10_B_HIIE** - Interrupts when the input signal has gone above the high threshold of the window comparator
- **ADC10_B_LOIE** - Interrupts when the input signal has gone below the low threshold of the low window comparator
- **ADC10_B_INIE** - Interrupts when the input signal is in between the high and low thresholds of the window comparator
- **ADC10_B_IE0** - Interrupt for new conversion data in the memory buffer

Modified bits of **ADC10IE** register.

Returns:

None

5.2.2.9 ADC10_B_enableReferenceBurst

Enables the reference buffer's burst ability.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	6
CCS 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	30
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
ADC10_B_enableReferenceBurst(uint32_t baseAddress)
```

Description:

Enables the reference buffer's burst ability, allowing the reference buffer to turn off while the ADC is not converting, and automatically turning on when the ADC needs the generated reference voltage for a conversion.

Parameters:
baseAddress is the base address of the ADC10B module.

Modified bits are **ADC10REFBURST** of **ADC10CTL2** register.

Returns:
None

5.2.2.10 ADC10_B_getInterruptStatus

Returns the status of the selected memory interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:
```
uint8_t
ADC10_B_getInterruptStatus(uint32_t baseAddress,
                           uint8_t interruptFlagMask)
```

Description:
Returns the status of the selected interrupt flags.

Parameters:
baseAddress is the base address of the ADC10B module.
interruptFlagMask is a bit mask of the interrupt flags status to be returned. Mask value is the logical OR of any of the following:
- **ADC10_B_OVIFG** - Interrupt flag for when a new conversion is about to overwrite the previous one
- **ADC10_B_TOVIFG** - Interrupt flag for when a new conversion is starting before the previous one has finished
- **ADC10_B_HIIFG** - Interrupt flag for when the input signal has gone above the high threshold of the window comparator
- **ADC10_B_LOIFG** - Interrupt flag for when the input signal has gone below the low threshold of the window comparator
- **ADC10_B_INIFG** - Interrupt flag for when the input signal is in between the high and low thresholds of the window comparator
- **ADC10_B_IFG0** - Interrupt flag for new conversion data in the memory buffer

Modified bits of **ADC10IFG** register.

Returns:
The current interrupt flag status for the corresponding mask.

5.2.2.11 ADC10_B_getMemoryAddressForDMA

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	26
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:
```
uint32_t
ADC10_B_getMemoryAddressForDMA(uint32_t baseAddress)
```

Parameters:
baseAddress is the base address of the ADC10B module.

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

5.2.2.12 uint16_t ADC10_B_getResults (uint32_t *baseAddress*)

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
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	22
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the ADC10B module.

Returns:

A Signed Integer of the contents of the specified memory buffer.

5.2.2.13 bool ADC10_B_init (uint32_t *baseAddress*, uint16_t *sampleHoldSignalSourceSelect*, uint8_t *clockSourceSelect*, uint16_t *clockSourceDivider*)

Initializes the ADC10B 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
CCS 4.2.1	None	108
CCS 4.2.1	Size	60
CCS 4.2.1	Speed	58
IAR 5.51.6	None	98
IAR 5.51.6	Size	80
IAR 5.51.6	Speed	80
MSPGCC 4.8.0	None	150
MSPGCC 4.8.0	Size	74
MSPGCC 4.8.0	Speed	74

Parameters:

baseAddress is the base address of the ADC10B module.

sampleHoldSignalSourceSelect is the signal that will trigger a sample-and-hold for an input signal to be converted. This parameter is device specific and sources should be found in the device's datasheet. Valid values are:

- **ADC10_B_SAMPLEHOLDSOURCE_SC** [Default]
- **ADC10_B_SAMPLEHOLDSOURCE_1**
- **ADC10_B_SAMPLEHOLDSOURCE_2**
- **ADC10_B_SAMPLEHOLDSOURCE_3**

Modified bits are **ADC10SHSx** of **ADC10CTL1** register.

clockSourceSelect selects the clock that will be used by the ADC10B core and the sampling timer if a sampling pulse mode is enabled. Valid values are:

- **ADC10_B_CLOCKSOURCE_ADC10OSC** [Default] - MODOSC 5 MHz oscillator from the clock system
- **ADC10_B_CLOCKSOURCE_ACLK** - The Auxiliary Clock
- **ADC10_B_CLOCKSOURCE_MCLK** - The Master Clock
- **ADC10_B_CLOCKSOURCE_SMCLK** - The Sub-Master Clock

Modified bits are **ADC10SSELx** of **ADC10CTL1** register.

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

- **ADC10_B_CLOCKDIVIDER_1** [Default]
- **ADC10_B_CLOCKDIVIDER_2**
- **ADC10_B_CLOCKDIVIDER_3**
- **ADC10_B_CLOCKDIVIDER_4**
- **ADC10_B_CLOCKDIVIDER_5**
- **ADC10_B_CLOCKDIVIDER_6**
- **ADC10_B_CLOCKDIVIDER_7**
- **ADC10_B_CLOCKDIVIDER_8**
- **ADC10_B_CLOCKDIVIDER_12**
- **ADC10_B_CLOCKDIVIDER_16**
- **ADC10_B_CLOCKDIVIDER_20**
- **ADC10_B_CLOCKDIVIDER_24**
- **ADC10_B_CLOCKDIVIDER_28**
- **ADC10_B_CLOCKDIVIDER_32**
- **ADC10_B_CLOCKDIVIDER_64**
- **ADC10_B_CLOCKDIVIDER_128**
- **ADC10_B_CLOCKDIVIDER_192**
- **ADC10_B_CLOCKDIVIDER_256**
- **ADC10_B_CLOCKDIVIDER_320**
- **ADC10_B_CLOCKDIVIDER_384**
- **ADC10_B_CLOCKDIVIDER_448**
- **ADC10_B_CLOCKDIVIDER_512**

Modified bits are **ADC10DIVx** of **ADC10CTL1** register; bits **ADC10PDIVx** of **ADC10CTL2** register.

Returns:

STATUS_SUCCESS or STATUS_FAILURE of the initialization process.

5.2.2.14 uint8_t ADC10_B_isBusy (uint32_t *baseAddress*)

Returns the busy status of the ADC10B core.

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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	32
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Parameters:

baseAddress is the base address of the ADC10B module.

Returns:

ADC10_B_BUSY or ADC10_B_NOTBUSY dependent if there is a conversion currently taking place. Return one of the following:

- **ADC10_B_NOTBUSY**
- **ADC10_B_BUSY**

5.2.2.15 void ADC10_B_memoryConfigure (uint32_t *baseAddress*, uint8_t *inputSourceSelect*, uint8_t *positiveRefVoltageSourceSelect*, uint8_t *negativeRefVoltageSourceSelect*)

Configures the controls of the selected memory buffer.

Maps an input signal conversion into the memory buffer, as well as the positive and negative reference voltages for each conversion being stored into the memory buffer. If the internal reference is used for the positive reference voltage, the internal REF module has 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
CCS 4.2.1	None	44
CCS 4.2.1	Size	14
CCS 4.2.1	Speed	14
IAR 5.51.6	None	26
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Parameters:

baseAddress is the base address of the ADC10B module.

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

- **ADC10_B_INPUT_A0** [Default]
- **ADC10_B_INPUT_A1**
- **ADC10_B_INPUT_A2**
- **ADC10_B_INPUT_A3**
- **ADC10_B_INPUT_A4**
- **ADC10_B_INPUT_A5**
- **ADC10_B_INPUT_A6**
- **ADC10_B_INPUT_A7**
- **ADC10_B_INPUT_VEREF_P**
- **ADC10_B_INPUT_VEREF_N**
- **ADC10_B_INPUT_TEMPSENSOR**
- **ADC10_B_INPUT_BATTERYMONITOR**
- **ADC10_B_INPUT_A12**
- **ADC10_B_INPUT_A13**
- **ADC10_B_INPUT_A14**
- **ADC10_B_INPUT_A15**

  Modified bits are **ADC10INCHx** of **ADC10MCTL0** register.

positiveRefVoltageSourceSelect is the reference voltage source to set as the upper limit for the conversion that is to be stored in the specified memory buffer. Valid values are:

- **ADC10_B_VREFPOS_AVCC** [Default]
- **ADC10_B_VREFPOS_EXT**
- **ADC10_B_VREFPOS_INT**

  Modified bits are **ADC10SREF** of **ADC10MCTL0** register.

negativeRefVoltageSourceSelect is the reference voltage source to set as the lower limit for the conversion that is to be stored in the specified memory buffer. Valid values are:

- **ADC10_B_VREFNEG_AVSS** [Default]
- **ADC10_B_VREFNEG_EXT**

  Modified bits are **ADC10SREF** of **ADC10MCTL0** register.

Returns:
None

5.2.2.16 void ADC10_B_setDataReadBackFormat (uint32_t *baseAddress*, uint16_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
CCS 4.2.1	None	40
CCS 4.2.1	Size	12
CCS 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	62
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the ADC10B module.

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

- **ADC10_B_UNSIGNED_BINARY** [Default]
- **ADC10_B_SIGNED_2SCOMPLEMENT**
  Modified bits are **ADC10DF** of **ADC10CTL2** register.

Returns:

None

5.2.2.17 void ADC10_B_setReferenceBufferSamplingRate (uint32_t *baseAddress*, uint16_t *samplingRateSelect*)

Use to set the reference buffer's sampling rate.

Sets the reference buffer's sampling rate to the selected sampling rate. The default sampling rate is maximum of 200-ksps, and can be reduced to a maximum of 50-ksps to conserve power.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	40
CCS 4.2.1	Size	12
CCS 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	62
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Parameters:

baseAddress is the base address of the ADC10B module.

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

- **ADC10_B_MAXSAMPLINGRATE_200KSPS** [Default]

- **ADC10_B_MAXSAMPLINGRATE_50KSPS**
  Modified bits are **ADC10SR** of **ADC10CTL2** register.

Description:
Modified bits of **ADC10CTL2** register.

Returns:
None

5.2.2.18 ADC10_B_setResolution

Use to change the resolution of the converted data.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	44
CCS 4.2.1	Size	16
CCS 4.2.1	Speed	16
IAR 5.51.6	None	22
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	68
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Prototype:
```
void
ADC10_B_setResolution(uint32_t baseAddress,
                      uint8_t resolutionSelect)
```

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

Parameters:
baseAddress is the base address of the ADC10B module.
resolutionSelect determines the resolution of the converted data. Valid values are:

- **ADC10_B_RESOLUTION_8BIT**
- **ADC10_B_RESOLUTION_10BIT** [Default]
  Modified bits are **ADC10RES** of **ADC10CTL2** register.

Returns:
None

5.2.2.19 ADC10_B_setSampleHoldSignalInversion

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	14
CCS 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	64
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
ADC10_B_setSampleHoldSignalInversion(uint32_t baseAddress,
                                     uint16_t invertedSignal)
```

Description:

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.

Parameters:

baseAddress is the base address of the ADC10B module.

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

- **ADC10_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.
- **ADC10_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 **ADC10ISSH** of **ADC10CTL1** register.

Returns:

None

5.2.2.20 ADC10_B_setupSamplingTimer

Sets up and enables the Sampling Timer Pulse Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	54
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	20
IAR 5.51.6	None	30
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	28
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Prototype:

```
void
ADC10_B_setupSamplingTimer(uint32_t baseAddress,
                           uint16_t clockCycleHoldCount,
                           uint16_t multipleSamplesEnabled)
```

Description:

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.

Parameters:

baseAddress is the base address of the ADC10B module.

clockCycleHoldCount sets the amount of clock cycles to sample-and- hold for the memory buffer. Valid values are:

- **ADC10_B_CYCLEHOLD_4_CYCLES** [Default]
- **ADC10_B_CYCLEHOLD_8_CYCLES**
- **ADC10_B_CYCLEHOLD_16_CYCLES**
- **ADC10_B_CYCLEHOLD_32_CYCLES**
- **ADC10_B_CYCLEHOLD_64_CYCLES**
- **ADC10_B_CYCLEHOLD_96_CYCLES**
- **ADC10_B_CYCLEHOLD_128_CYCLES**
- **ADC10_B_CYCLEHOLD_192_CYCLES**
- **ADC10_B_CYCLEHOLD_256_CYCLES**
- **ADC10_B_CYCLEHOLD_384_CYCLES**
- **ADC10_B_CYCLEHOLD_512_CYCLES**
- **ADC10_B_CYCLEHOLD_768_CYCLES**
- **ADC10_B_CYCLEHOLD_1024_CYCLES**

  Modified bits are **ADC10SHTx** of **ADC10CTL0** register.

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

- **ADC10_B_MULTIPLESAMPLESDISABLE** - a timer trigger will be needed to start every ADC conversion.
- **ADC10_B_MULTIPLESAMPLESENABLE** - during a sequenced and/or repeated conversion mode, after the first conversion, no sample/hold signal is necessary to start subsequent samples.

  Modified bits are **ADC10MSC** of **ADC10CTL0** register.

Returns:
None

5.2.2.21 ADC10_B_setWindowComp

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	38
CCS 4.2.1	Size	10
CCS 4.2.1	Speed	10
IAR 5.51.6	None	24
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Prototype:

```
void
ADC10_B_setWindowComp(uint32_t baseAddress,
                      uint16_t highThreshold,
                      uint16_t lowThreshold)
```

Description:
Sets the high and low threshold for the window comparator feature. Use the ADC10HIIE, ADC10INIE, ADC10LOIE interrupts to utilize this feature.

Parameters:
baseAddress is the base address of the ADC10B 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.

Modified bits of **ADC10LO** register and bits of **ADC10HI** register.

Returns:
None

5.2.2.22 ADC10_B_startConversion

Enables/Starts an Analog-to-Digital Conversion.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	48
CCS 4.2.1	Size	18
CCS 4.2.1	Speed	18
IAR 5.51.6	None	24
IAR 5.51.6	Size	26
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	80
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Prototype:

```
void
ADC10_B_startConversion(uint32_t baseAddress,
                        uint8_t conversionSequenceModeSelect)
```

Description:

This function enables/starts the conversion process of the ADC. If the sample/hold signal source chosen during initialization was ADC10OSC, 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 ADC10_B_setupSamplingTimer() function. Also note that when a sequence conversion mode is selected, the first input channel is the one mapped to the memory buffer, the next input channel selected for conversion is one less than the input channel just converted (i.e. A1 comes after A2), until A0 is reached, and if in repeating mode, then the next input channel will again be the one mapped to the memory buffer. Note that after this function is called, the ADC10_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.

Parameters:

baseAddress is the base address of the ADC10B module.

conversionSequenceModeSelect determines the ADC operating mode. Valid values are:

- **ADC10_B_SINGLECHANNEL** [Default] - one-time conversion of a single channel into a single memory buffer
- **ADC10_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
- **ADC10_B_REPEATED_SINGLECHANNEL** - repeated conversions of one channel into a single memory buffer
- **ADC10_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 **ADC10CONSEQx** of **ADC10CTL1** register.

Returns:

None

5.3 Programming Example

The following example shows how to initialize and use the ADC10_B API to start a single channel, single conversion.

```
    // Initialize ADC10_B with ADC10_B's built-in oscillator
    ADC10_B_init (ADC10_B_BASE,
                          ADC10_B_SAMPLEHOLDSOURCE_SC,
                          ADC10_B_CLOCKSOURCE_ADC10OSC,
                          ADC10_B_CLOCKDIVIDEBY_1);

    //Switch ON ADC10_B
ADC10_B_enable(ADC10_B_BASE);

    // Setup sampling timer to sample-and-hold for 16 clock cycles
    ADC10_B_setupSamplingTimer (ADC10_B_BASE,
                                                    ADC10_B_CYCLEHOLD_16_CYCLES,
                                                    FALSE);

    // Configure the Input to the Memory Buffer with the specified Reference Voltages
    ADC10_B_memoryConfigure (ADC10_B_BASE,
                                                ADC10_B_INPUT_A0,
                                                ADC10_B_VREFPOS_AVCC, // Vref+ = AVcc
                                                ADC10_B_VREFNEG_AVSS  // Vref- = AVss
                                                );
    while (1)
    {
            // Start a single conversion, no repeating or sequences.
            ADC10_B_startConversion (ADC10_B_BASE,
                                                     ADC10_B_SINGLECHANNEL);

            // Wait for the Interrupt Flag to assert
            while( !(ADC10_B_getInterruptStatus(ADC10_B_BASE,ADC10IFG0)) );

            // Clear the Interrupt Flag and start another conversion
            ADC10_B_clearInterrupt(ADC10_B_BASE,ADC10IFG0);
    }
```

6 Comparator (COMP_D)

Introduction 41
API Functions 41
Programming Example 54

6.1 Introduction

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

The COMP_D 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_D 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_D module also has control over the REF module to generate a reference voltage as an input.

The COMP_D 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_d.c`, with `comp_d.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the CCS 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 withing 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
CCS 4.2.1	None	770
CCS 4.2.1	Size	340
CCS 4.2.1	Speed	336
IAR 5.51.6	None	490
IAR 5.51.6	Size	346
IAR 5.51.6	Speed	374
MSPGCC 4.8.0	None	1150
MSPGCC 4.8.0	Size	418
MSPGCC 4.8.0	Speed	468

6.2 API Functions

Functions

- void COMP_D_clearInterrupt (uint32_t baseAddress, uint16_t interruptFlagMask)
- void COMP_D_disable (uint32_t baseAddress)
- void COMP_D_disableInputBuffer (uint32_t baseAddress, uint8_t inputPort)
- void COMP_D_disableInterrupt (uint32_t baseAddress, uint16_t interruptMask)
- void COMP_D_enable (uint32_t baseAddress)
- void COMP_D_enableInputBuffer (uint32_t baseAddress, uint8_t inputPort)
- void COMP_D_enableInterrupt (uint32_t baseAddress, uint16_t interruptMask)
- uint8_t COMP_D_getInterruptStatus (uint32_t baseAddress, uint16_t interruptFlagMask)
- bool COMP_D_init (uint32_t baseAddress, uint8_t positiveTerminalInput, uint8_t negativeTerminalInput, uint8_t outputFilterEnableAndDelayLevel, uint16_t invertedOutputPolarity)

- void COMP_D_interruptSetEdgeDirection (uint32_t baseAddress, uint16_t edgeDirection)
- void COMP_D_interruptToggleEdgeDirection (uint32_t baseAddress)
- void COMP_D_IOSwap (uint32_t baseAddress)
- uint16_t COMP_D_outputValue (uint32_t baseAddress)
- void COMP_D_setReferenceAccuracy (uint32_t baseAddress, uint16_t referenceAccuracy)
- void COMP_D_setReferenceVoltage (uint32_t baseAddress, uint16_t supplyVoltageReferenceBase, uint16_t lowerLimitSupplyVoltageFractionOf32, uint16_t upperLimitSupplyVoltageFractionOf32)
- void COMP_D_shortInputs (uint32_t baseAddress)
- void COMP_D_unshortInputs (uint32_t baseAddress)

6.2.1 Detailed Description

The COMP_D 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_D.

The COMP_D initialization and output functions are

- COMP_D_init()
- COMP_D_setReferenceVoltage()
- COMP_D_enable()
- COMP_D_disable()
- COMP_D_outputValue()

The COMP_D interrupts are handled by

- COMP_D_enableInterrupt()
- COMP_D_disableInterrupt()
- COMP_D_clearInterrupt()
- COMP_D_getInterruptStatus()
- COMP_D_interruptSetEdgeDirection()
- COMP_D_interruptToggleEdgeDirection()

Auxiliary features of the COMP_D are handled by

- COMP_D_enableShortOfInputs()
- COMP_D_disableShortOfInputs()
- COMP_D_disableInputBuffer()
- COMP_D_enableInputBuffer()
- COMP_D_IOSwap()
- COMP_D_setReferenceAccuracy()

6.2.2 Function Documentation

6.2.2.1 COMP_D_clearInterrupt

Clears Comparator interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	34
CCS 4.2.1	Size	6
CCS 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	50
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
COMP_D_clearInterrupt(uint32_t baseAddress,
                      uint16_t interruptFlagMask)
```

Description:

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

Parameters:

baseAddress is the base address of the COMP_D module.

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

- **COMP_D_INTERRUPT_FLAG** - Output interrupt flag
- **COMP_D_INTERRUPT_FLAG_INVERTED_POLARITY** - Output interrupt flag inverted polarity

6.2.2.2 COMP_D_disable

Turns off the Comparator module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
COMP_D_disable(uint32_t baseAddress)
```

Description:

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

Parameters:

baseAddress is the base address of the COMP_D module.

Returns:

None

6.2.2.3 COMP_D_disableInputBuffer

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	16
CCS 4.2.1	Speed	16
IAR 5.51.6	None	28
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	62
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

Prototype:

```
void
COMP_D_disableInputBuffer(uint32_t baseAddress,
                          uint8_t inputPort)
```

Description:

This function sets the bit to disable the buffer for the specified input port to allow for analog signals from any of the comparator input pins. This bit is automatically set when the input is initialized to be used with the comparator 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.

Parameters:

baseAddress is the base address of the COMP_D module.

inputPort is the port in which the input buffer will be disabled. Valid values are:

- **COMP_D_INPUT0** [Default]
- **COMP_D_INPUT1**
- **COMP_D_INPUT2**
- **COMP_D_INPUT3**
- **COMP_D_INPUT4**
- **COMP_D_INPUT5**
- **COMP_D_INPUT6**
- **COMP_D_INPUT7**
- **COMP_D_INPUT8**
- **COMP_D_INPUT9**
- **COMP_D_INPUT10**
- **COMP_D_INPUT11**
- **COMP_D_INPUT12**
- **COMP_D_INPUT13**
- **COMP_D_INPUT14**
- **COMP_D_INPUT15**
- **COMP_D_VREF**

  Modified bits are **CDPDx** of **CDCTL3** register.

Returns:

None

6.2.2.4 COMP_D_disableInterrupt

Disables selected Comparator interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	34
CCS 4.2.1	Size	6
CCS 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	50
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
COMP_D_disableInterrupt(uint32_t baseAddress,
                        uint16_t interruptMask)
```

Description:

Disables the indicated Comparator interrupt sources. 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 COMP_D module.

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

- **COMP_D_INTERRUPT** - Output interrupt
- **COMP_D_INTERRUPT_INVERTED_POLARITY** - Output interrupt inverted polarity

6.2.2.5 COMP_D_enable

Turns on the Comparator module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
COMP_D_enable(uint32_t baseAddress)
```

Description:

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

Parameters:

baseAddress is the base address of the COMP_D module.

Returns:

None

6.2.2.6 COMP_D_enableInputBuffer

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	16
CCS 4.2.1	Speed	16
IAR 5.51.6	None	28
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

Prototype:

```
void
COMP_D_enableInputBuffer(uint32_t baseAddress,
                         uint8_t inputPort)
```

Description:

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

Parameters:

baseAddress is the base address of the COMP_D module.

inputPort is the port in which the input buffer will be enabled. Valid values are:

- **COMP_D_INPUT0** [Default]
- **COMP_D_INPUT1**
- **COMP_D_INPUT2**
- **COMP_D_INPUT3**
- **COMP_D_INPUT4**
- **COMP_D_INPUT5**
- **COMP_D_INPUT6**
- **COMP_D_INPUT7**
- **COMP_D_INPUT8**
- **COMP_D_INPUT9**
- **COMP_D_INPUT10**
- **COMP_D_INPUT11**
- **COMP_D_INPUT12**
- **COMP_D_INPUT13**
- **COMP_D_INPUT14**
- **COMP_D_INPUT15**
- **COMP_D_VREF**

Modified bits are **CDPDx** of **CDCTL3** register.

Returns:

None

6.2.2.7 COMP_D_enableInterrupt

Enables selected Comparator interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	34
CCS 4.2.1	Size	6
CCS 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	46
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
COMP_D_enableInterrupt(uint32_t baseAddress,
                       uint16_t interruptMask)
```

Description:

Enables the indicated Comparator 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 COMP_D module.

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

- **COMP_D_INTERRUPT** - Output interrupt
- **COMP_D_INTERRUPT_INVERTED_POLARITY** - Output interrupt inverted polarity

6.2.2.8 COMP_D_getInterruptStatus

Gets the current Comparator interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	52
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint8_t
COMP_D_getInterruptStatus(uint32_t baseAddress,
                          uint16_t interruptFlagMask)
```

Description:

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

Parameters:

baseAddress is the base address of the COMP_D module.

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

- **COMP_D_INTERRUPT_FLAG** - Output interrupt flag
- **COMP_D_INTERRUPT_FLAG_INVERTED_POLARITY** - Output interrupt flag inverted polarity

6.2.2.9 COMP_D_init

Initializes the COMP_D Module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	176
CCS 4.2.1	Size	118
CCS 4.2.1	Speed	118
IAR 5.51.6	None	162
IAR 5.51.6	Size	136
IAR 5.51.6	Speed	144
MSPGCC 4.8.0	None	280
MSPGCC 4.8.0	Size	152
MSPGCC 4.8.0	Speed	168

Prototype:

```
bool
COMP_D_init(uint32_t baseAddress,
            uint8_t positiveTerminalInput,
            uint8_t negativeTerminalInput,
            uint8_t outputFilterEnableAndDelayLevel,
            uint16_t invertedOutputPolarity)
```

Description:

Upon successful initialization of the COMP_D module, this function will have reset all necessary register bits and set the given options in the registers. To actually use the COMP_D module, the COMP_D_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.

Parameters:

baseAddress is the base address of the COMP_D module.

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

- **COMP_D_INPUT0** [Default]
- **COMP_D_INPUT1**
- **COMP_D_INPUT2**
- **COMP_D_INPUT3**
- **COMP_D_INPUT4**
- **COMP_D_INPUT5**
- **COMP_D_INPUT6**
- **COMP_D_INPUT7**
- **COMP_D_INPUT8**
- **COMP_D_INPUT9**
- **COMP_D_INPUT10**
- **COMP_D_INPUT11**
- **COMP_D_INPUT12**
- **COMP_D_INPUT13**
- **COMP_D_INPUT14**
- **COMP_D_INPUT15**
- **COMP_D_VREF**
  Modified bits are **CDRSEL** of **CDCTL2** register; bits **CDPDx** of **CDCTL3** register; bits **CDIPEN** of **CDCTL0** register.

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

- **COMP_D_INPUT0** [Default]
- **COMP_D_INPUT1**
- **COMP_D_INPUT2**
- **COMP_D_INPUT3**
- **COMP_D_INPUT4**

- **COMP_D_INPUT5**
- **COMP_D_INPUT6**
- **COMP_D_INPUT7**
- **COMP_D_INPUT8**
- **COMP_D_INPUT9**
- **COMP_D_INPUT10**
- **COMP_D_INPUT11**
- **COMP_D_INPUT12**
- **COMP_D_INPUT13**
- **COMP_D_INPUT14**
- **COMP_D_INPUT15**
- **COMP_D_VREF**
  Modified bits are **CDRSEL** of **CDCTL2** register; bits **CDPDx** of **CDCTL3** register; bits **CDIMEN** of **CDCTL0** 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_D_FILTEROUTPUT_OFF** [Default]
- **COMP_D_FILTEROUTPUT_DLYLVL1**
- **COMP_D_FILTEROUTPUT_DLYLVL2**
- **COMP_D_FILTEROUTPUT_DLYLVL3**
- **COMP_D_FILTEROUTPUT_DLYLVL4**
  Modified bits are **CDFDLY** of **CDCTL1** register.

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

- **COMP_D_NORMALOUTPUTPOLARITY** [Default]
- **COMP_D_INVERTEDOUTPUTPOLARITY**

Returns:
STATUS_SUCCESS or STATUS_FAILURE of the initialization process

6.2.2.10 COMP_D_interruptSetEdgeDirection

Explicitly sets the edge direction that would trigger an interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	52
CCS 4.2.1	Size	20
CCS 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	78
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	20

Prototype:

```
void
COMP_D_interruptSetEdgeDirection(uint32_t baseAddress,
                                 uint16_t edgeDirection)
```

Description:
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.

Parameters:
baseAddress is the base address of the COMP_D 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_D_FALLINGEDGE** [Default] - sets the bit to generate an interrupt when the output of the comparator 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_D_RISINGEDGE** - sets the bit to generate an interrupt when the output of the comparator 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 **CDIES** of **CDCTL1** register.

Returns:
None

6.2.2.11 COMP_D_interruptToggleEdgeDirection

Toggles the edge direction that would trigger an interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	6
CCS 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	30
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:

```
void
COMP_D_interruptToggleEdgeDirection(uint32_t baseAddress)
```

Description:
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.

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

Returns:
None

6.2.2.12 COMP_D_IOSwap

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	8
CCS 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	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
COMP_D_IOSwap(uint32_t baseAddress)
```

Description:

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.

Parameters:

baseAddress is the base address of the COMP_D module.

Returns:

None

6.2.2.13 COMP_D_outputValue

Returns the output value of the COMP_D module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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	22
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint16_t
COMP_D_outputValue(uint32_t baseAddress)
```

Description:

Returns the output value of the COMP_D module.

Parameters:

baseAddress is the base address of the COMP_D module.

Returns:

COMP_D_HIGH or COMP_D_LOW as the output value of the Comparator module. Return one of the following:

- **COMP_D_HIGH**
- **COMP_D_LOW**

  indicates the output should be normal

6.2.2.14 COMP_D_setReferenceAccuracy

Sets the reference accuracy.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	14
CCS 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	64
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
COMP_D_setReferenceAccuracy(uint32_t baseAddress,
                            uint16_t referenceAccuracy)
```

Description:

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

Parameters:

baseAddress is the base address of the COMP_D module.

referenceAccuracy is the reference accuracy setting of the comparator. Clocked is for low power/low accuracy. Valid values are:

- **COMP_D_ACCURACY_STATIC**
- **COMP_D_ACCURACY_CLOCKED**
  Modified bits are **CDREFACC** of **CDCTL2** register.

Returns:

None

6.2.2.15 COMP_D_setReferenceVoltage

Generates a Reference Voltage to the terminal selected during initialization.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	118
CCS 4.2.1	Size	74
CCS 4.2.1	Speed	70
IAR 5.51.6	None	100
IAR 5.51.6	Size	76
IAR 5.51.6	Speed	74
MSPGCC 4.8.0	None	192
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	108

Prototype:

```
void
COMP_D_setReferenceVoltage(uint32_t baseAddress,
                           uint16_t supplyVoltageReferenceBase,
                           uint16_t lowerLimitSupplyVoltageFractionOf32,
                           uint16_t upperLimitSupplyVoltageFractionOf32)
```

Description:

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: Vbase $*$ (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.

Parameters:

baseAddress is the base address of the COMP_D module.

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

- **COMP_D_REFERENCE_AMPLIFIER_DISABLED**
- **COMP_D_VREFBASE1_5V**
- **COMP_D_VREFBASE2_0V**
- **COMP_D_VREFBASE2_5V**

Modified bits are **CDREFL** of **CDCTL2** register.

lowerLimitSupplyVoltageFractionOf32 is the numerator of the equation to generate the reference voltage for the lower limit reference voltage.
Modified bits are **CDREF0** of **CDCTL2** register.

upperLimitSupplyVoltageFractionOf32 is the numerator of the equation to generate the reference voltage for the upper limit reference voltage.
Modified bits are **CDREF1** of **CDCTL2** register.

Returns:

None

6.2.2.16 COMP_D_shortInputs

Shorts the two input pins chosen during initialization.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
COMP_D_shortInputs(uint32_t baseAddress)
```

Description:

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

Parameters:

baseAddress is the base address of the COMP_D module.

Returns:

None

6.2.2.17 COMP_D_unshortInputs

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
COMP_D_unshortInputs(uint32_t baseAddress)
```

Description:

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

Parameters:

baseAddress is the base address of the COMP_D module.

Returns:

None

6.3 Programming Example

The following example shows how to initialize and use the COMP_D 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 D module
/* Base Address of Comparator D,
  Pin CD2 to Positive(+) Terminal,
  Reference Voltage to Negative(-) Terminal,
  Normal Power Mode,
  Output Filter On with minimal delay,
  Non-Inverted Output Polarity
*/
COMP_D_init(COMP_D_BASE,
    COMP_D_INPUT2,
    COMP_D_VREF,
    COMP_D_FILTEROUTPUT_OFF,
    COMP_D_NORMALOUTPUTPOLARITY
    );

// Set the reference voltage that is being supplied to the (-) terminal
/* Base Address of Comparator D,
 Reference Voltage of 2.0 V,
 Upper Limit of 2.0*(32/32) = 2.0V,
 Lower Limit of 2.0*(32/32) = 2.0V
*/
COMP_D_setReferenceVoltage(COMP_D_BASE,
    COMP_D_VREFBASE2_0V,
    32,
    32,
    COMP_D_ACCURACY_STATIC
    );

  //Disable Input Buffer on P1.2/CD2
      /* Base Address of Comparator D,
```

```
Input Buffer port
Selecting the CDx input pin to the comparator
multiplexer with the CDx bits automatically
disables output driver and input buffer for
that pin, regardless of the state of the
associated CDPD.x bit
        */
         COMP_D_disableInputBuffer(COMP_D_BASE,
                 COMP_D_INPUT2);
 // Allow power to Comparator module
 COMP_D_enable(COMP_D_BASE);

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

7 Cyclical Redundancy Check (CRC)

Introduction 56
API Functions 56
Programming Example 61

7.1 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.

T

he following code metrics were performed with the CCS 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 withing 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
CCS 4.2.1	None	178
CCS 4.2.1	Size	40
CCS 4.2.1	Speed	40
IAR 5.51.6	None	50
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	186
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	50

7.2 API Functions

Functions

- uint16_t CRC_getData (uint32_t baseAddress)
- uint16_t CRC_getResult (uint32_t baseAddress)
- uint16_t CRC_getResultBitsReversed (uint32_t baseAddress)
- void CRC_set16BitData (uint32_t baseAddress, uint16_t dataIn)
- void CRC_set8BitData (uint32_t baseAddress, uint8_t dataIn)
- void CRC_setDataByteBitsReversed (uint32_t baseAddress, uint16_t dataIn)
- void CRC_setSeed (uint32_t baseAddress, uint16_t seed)

7.2.1 Detailed Description

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

- CRC_setSeed()
- CRC_setDataByte()

- CRC_setDataWord()
- CRC_setSignatureByteReversed()
- CRC_getSignature()
- CRC_getResult()
- CRC_getResultBitReversed()

7.2.2 Function Documentation

7.2.2.1 CRC_getData

Returns the value currently in the Data register.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	4
CCS 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	18
MSPGCC 4.8.0	Size	4
MSPGCC 4.8.0	Speed	4

Prototype:

```
uint16_t
CRC_getData(uint32_t baseAddress)
```

Description:

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

Parameters:

baseAddress is the base address of the CRC module.

Returns:

The value currently in the data register

7.2.2.2 CRC_getResult

Returns the value pf the Signature Result.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
uint16_t
CRC_getResult(uint32_t baseAddress)
```

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

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

Returns:
The value currently in the data register

7.2.2.3 CRC_getResultBitsReversed

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	22
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint16_t
CRC_getResultBitsReversed(uint32_t baseAddress)
```

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

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

Returns:
The bit-wise reversed format of the Signature Result

7.2.2.4 CRC_set16BitData

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	6
CCS 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	30
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
CRC_set16BitData(uint32_t baseAddress,
                 uint16_t dataIn)
```

Description:

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

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

7.2.2.5 CRC_set8BitData

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	6
CCS 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	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
CRC_set8BitData(uint32_t baseAddress,
                uint8_t dataIn)
```

Description:

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

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

7.2.2.6 CRC_setDataByteBitsReversed

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	6
CCS 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	32
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
void
CRC_setDataByteBitsReversed(uint32_t baseAddress,
                            uint16_t dataIn)
```

Description:

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.

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

7.2.2.7 CRC_setSeed

Sets the seed for the CRC.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	6
CCS 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	32
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
void
CRC_setSeed(uint32_t baseAddress,
            uint16_t seed)
```

Description:

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.

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

7.3 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_setDataWord(CRC_BASE,
                  data[i]);
}

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

8 Clock System (CS)

Introduction 62
API Functions 63
Programming Example 64

8.1 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 up to five clock sources:

- XT1CLK - Low-frequency/high-frequency oscillator that can be used either with low-frequency 32768-Hz watch crystals, standard crystals, resonators, or external clock sources in the 4 MHz to 24 MHz range. When optional XT2 is present, the XT1 high-frequency mode may or may not be available, depending on the device configuration. See the device-specific data sheet for supported functions.
- VLOCLK - Internal very-low-power low-frequency oscillator with 10-kHz typical frequency
- DCOCLK - Internal digitally controlled oscillator (DCO) with three selectable fixed frequencies
- XT2CLK - Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 4 MHz to 24 MHz range. See device-specific data sheet for availability.

Four system clock signals are available from the clock module:

- ACLK - Auxiliary clock. The ACLK is software selectable as XT1CLK, VLOCLK, DCOCLK, and when available, XT2CLK. 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 XT1CLK, VLOCLK, DCOCLK, and when available, XT2CLK. MCLK can be divided by 1, 2, 4, 8, 16, or 32. MCLK is used by the CPU and system.
- SMCLK - Subsystem master clock. SMCLK is software selectable as XT1CLK, VLOCLK, DCOCLK, and when available, XT2CLK. SMCLK is software selectable by individual peripheral modules.
- MODCLK - Module clock. MODCLK is used by various peripheral modules and is sourced by MODOSC.

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 XT1 in LF mode is sourcing any system clock (ACLK, MCLK, or SMCLK), and a fault is detected, the system clock is automatically switched to the VLO for its clock source (VLOCLK). Similarly, if XT1 in HF mode is sourcing any system clock and a fault is detected, the system clock is automatically switched to MODOSC for its clock source (MODCLK).

When XT2 (if available) is sourcing any system clock and a fault is detected, the system clock is automatically switched to MODOSC for its clock source (MODCLK).

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.c`, with `cs.h` containing the API definitions for use by applications.

T

he following code metrics were performed with the CCS 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 withing 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
CCS 4.2.1	None	1270
CCS 4.2.1	Size	944
CCS 4.2.1	Speed	966
IAR 5.51.6	None	1038
IAR 5.51.6	Size	672
IAR 5.51.6	Speed	958
MSPGCC 4.8.0	None	2624
MSPGCC 4.8.0	Size	1038
MSPGCC 4.8.0	Speed	2174

8.2 API Functions

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 for FR57xx devices:

- CS_XT1Start()
- CS_bypassXT1()
- CS_bypassXT1WithTimeout()
- CS_XT1StartWithTimeout()
- CS_XT1Off()
- CS_XT2Start()
- CS_bypassXT2()
- CS_XT2StartWithTimeout()
- CS_bypassXT2WithTimeout()
- CS_XT2Off()

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 XT1 or XT2 is used and the user intends to call CS_getMCLK, CS_getSMCLK or CS_getACLK APIs and XT1Start, XT1ByPass, XT1StartWithTimeout, XT1ByPassWithTimeout. If not any of the previous API are going to be called, it is not necessary to invoke this API.

8.3 Programming Example

The following example shows the configuration of the CS module that sets ACLK=SMCLK=MCLK=DCOCLK

```
//Set DCO Frequency to 8MHz
CS_setDCOFreq(CS_BASE,CS_DCORSEL_0,CS_DCOFSEL_3);

//configure MCLK, SMCLK and ACLK to be source by DCOCLK
CS_clockSignalInit(CS_BASE,CS_ACLK,CS_DCOCLK_SELECT,CS_CLOCK_DIVIDER_1);
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);
```

9 Direct Memory Access (DMA)

Introduction 65
API Functions 65
Programming Example 80

9.1 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 CCS 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 withing 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
CCS 4.2.1	None	534
CCS 4.2.1	Size	296
CCS 4.2.1	Speed	302
IAR 5.51.6	None	380
IAR 5.51.6	Size	286
IAR 5.51.6	Speed	306
MSPGCC 4.8.0	None	882
MSPGCC 4.8.0	Size	332
MSPGCC 4.8.0	Speed	342

9.2 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)
- 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)

9.2.1 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()

9.2.2 Function Documentation

9.2.2.1 DMA_clearInterrupt

Clears the interrupt flag for the selected channel.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
DMA_clearInterrupt(uint8_t channelSelect)
```

Description:

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

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

9.2.2.2 DMA_clearNMIAbort

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	8
CCS 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

Prototype:
```
void
DMA_clearNMIAbort(uint8_t channelSelect)
```

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

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

9.2.2.3 DMA_disableInterrupt

Disables the DMA interrupt for the selected channel.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
DMA_disableInterrupt(uint8_t channelSelect)
```

Description:

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.

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

9.2.2.4 DMA_disableNMIAbort

Disables any NMI from interrupting a DMA transfer.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
DMA_disableNMIAbort(void)
```

Description:

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

Returns:

None

9.2.2.5 DMA_disableRoundRobinPriority

Disables Round Robin prioritization.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
DMA_disableRoundRobinPriority(void)
```

Description:

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).

Returns:

None

9.2.2.6 DMA_disableTransferDuringReadModifyWrite

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
DMA_disableTransferDuringReadModifyWrite(void)
```

Description:

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.

Returns:

None

9.2.2.7 DMA_disableTransfers

Disables transfers from being triggered.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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

Prototype:

```
void
DMA_disableTransfers(uint8_t channelSelect)
```

Description:

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.

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

9.2.2.8 DMA_enableInterrupt

Enables the DMA interrupt for the selected channel.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
DMA_enableInterrupt(uint8_t channelSelect)
```

Description:

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.

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

9.2.2.9 DMA_enableNMIAbort

Enables a NMI to interrupt a DMA transfer.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
DMA_enableNMIAbort(void)
```

Description:

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.

Returns:

None

9.2.2.10 DMA_enableRoundRobinPriority

Enables Round Robin prioritization.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
DMA_enableRoundRobinPriority(void)
```

Description:

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.

Returns:

None

9.2.2.11 DMA_enableTransferDuringReadModifyWrite

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
DMA_enableTransferDuringReadModifyWrite(void)
```

Description:

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

Returns:

None

9.2.2.12 DMA_enableTransfers

Enables transfers to be triggered.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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

Prototype:

```
void
DMA_enableTransfers(uint8_t channelSelect)
```

Description:

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

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

9.2.2.13 DMA_getInterruptStatus

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	12
CCS 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

Prototype:

```
uint16_t
DMA_getInterruptStatus(uint8_t channelSelect)
```

Description:

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

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

9.2.2.14 DMA_init

Initializes the specified DMA channel.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	160
CCS 4.2.1	Size	94
CCS 4.2.1	Speed	100
IAR 5.51.6	None	132
IAR 5.51.6	Size	88
IAR 5.51.6	Speed	86
MSPGCC 4.8.0	None	242
MSPGCC 4.8.0	Size	92
MSPGCC 4.8.0	Speed	102

Prototype:
```
bool
DMA_init(uint8_t channelSelect,
         uint16_t transferModeSelect,
         uint16_t transferSize,
         uint8_t triggerSourceSelect,
         uint8_t transferUnitSelect,
         uint8_t triggerTypeSelect)
```

Description:
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.

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.

9.2.2.15 DMA_NMIAbortStatus

Returns the status of the NMIAbort for the selected channel.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	12
CCS 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

Prototype:
```
uint16_t
DMA_NMIAbortStatus(uint8_t channelSelect)
```

Description:
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.

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

9.2.2.16 DMA_setDstAddress

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	70
CCS 4.2.1	Size	38
CCS 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

Prototype:

```
void
DMA_setDstAddress(uint8_t channelSelect,
                  uint32_t dstAddress,
                  uint16_t directionSelect)
```

Description:

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.

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

9.2.2.17 DMA_setSrcAddress

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	66
CCS 4.2.1	Size	36
CCS 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

Prototype:

```
void
DMA_setSrcAddress(uint8_t channelSelect,
                  uint32_t srcAddress,
                  uint16_t directionSelect)
```

Description:

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.

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 **DMAxSA** of **DMAxSA** 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 **DMAxCTL** register.

Returns:

None

9.2.2.18 DMA_setTransferSize

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
DMA_setTransferSize(uint8_t channelSelect,
                    uint16_t transferSize)
```

Description:

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

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

9.2.2.19 DMA_startTransfer

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	8
CCS 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

Prototype:
```
void
DMA_startTransfer(uint8_t channelSelect)
```

Description:
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.

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

9.3 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);
 }
```

10 EUSCI Universal Asynchronous Receiver/Transmitter (EUSCI_A_UART)

Introduction 81
API Functions 81
Programming Example 94

10.1 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 CCS 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 withing 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
CCS 4.2.1	None	806
CCS 4.2.1	Size	378
CCS 4.2.1	Speed	378
IAR 5.51.6	None	542
IAR 5.51.6	Size	360
IAR 5.51.6	Speed	420
MSPGCC 4.8.0	None	1216
MSPGCC 4.8.0	Size	422
MSPGCC 4.8.0	Speed	548

10.2 API Functions

Functions

- void EUSCI_A_UART_clearInterruptFlag (uint32_t baseAddress, uint8_t mask)

- void EUSCI_A_UART_disable (uint32_t baseAddress)
- void EUSCI_A_UART_disableInterrupt (uint32_t baseAddress, uint8_t mask)
- void EUSCI_A_UART_enable (uint32_t baseAddress)
- void EUSCI_A_UART_enableInterrupt (uint32_t baseAddress, uint8_t mask)
- uint8_t EUSCI_A_UART_getInterruptStatus (uint32_t baseAddress, uint8_t mask)
- uint32_t EUSCI_A_UART_getReceiveBufferAddress (uint32_t baseAddress)
- uint32_t EUSCI_A_UART_getTransmitBufferAddress (uint32_t baseAddress)
- bool EUSCI_A_UART_initAdvance (uint32_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)
- uint8_t EUSCI_A_UART_queryStatusFlags (uint32_t baseAddress, uint8_t mask)
- uint8_t EUSCI_A_UART_receiveData (uint32_t baseAddress)
- void EUSCI_A_UART_resetDormant (uint32_t baseAddress)
- void EUSCI_A_UART_selectDeglitchTime (uint32_t baseAddress, uint32_t deglitchTime)
- void EUSCI_A_UART_setDormant (uint32_t baseAddress)
- void EUSCI_A_UART_transmitAddress (uint32_t baseAddress, uint8_t transmitAddress)
- void EUSCI_A_UART_transmitBreak (uint32_t baseAddress)
- void EUSCI_A_UART_transmitData (uint32_t baseAddress, uint8_t transmitData)

10.2.1 Detailed Description

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

The EUSI_A_UART API is broken into three groups of functions: those that deal with configuration and control of the EUSI_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 EUSI_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 EUSI_UART is handled by the

- EUSCI_A_UART_transmitData()
- EUSCI_A_UART_receiveData()
- EUSCI_A_UART_transmitAddress()
- EUSCI_A_UART_transmitBreak()

Managing the EUSI_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()

10.2.2 Function Documentation

10.2.2.1 EUSCI_A_UART_clearInterruptFlag

Clears UART interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	54
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
EUSCI_A_UART_clearInterruptFlag(uint32_t baseAddress,
                                uint8_t mask)
```

Description:

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.

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

10.2.2.2 EUSCI_A_UART_disable

Disables the UART block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
EUSCI_A_UART_disable(uint32_t baseAddress)
```

Description:
This will disable operation of the UART block.

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

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

Returns:
None

10.2.2.3 EUSCI_A_UART_disableInterrupt

Disables individual UART interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	70
CCS 4.2.1	Size	26
CCS 4.2.1	Speed	26
IAR 5.51.6	None	40
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	98
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Prototype:

```
void
EUSCI_A_UART_disableInterrupt(uint32_t baseAddress,
                              uint8_t mask)
```

Description:
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.

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

10.2.2.4 EUSCI_A_UART_enable

Enables the UART block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
EUSCI_A_UART_enable(uint32_t baseAddress)
```

Description:

This will enable operation of the UART block.

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

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

Returns:

None

10.2.2.5 EUSCI_A_UART_enableInterrupt

Enables individual UART interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	70
CCS 4.2.1	Size	22
CCS 4.2.1	Speed	22
IAR 5.51.6	None	40
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	90
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Prototype:

```
void
EUSCI_A_UART_enableInterrupt(uint32_t baseAddress,
                             uint8_t mask)
```

Description:

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.

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

10.2.2.6 EUSCI_A_UART_getInterruptStatus

Gets the current UART interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint8_t
EUSCI_A_UART_getInterruptStatus(uint32_t baseAddress,
                                uint8_t mask)
```

Description:

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

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

10.2.2.7 EUSCI_A_UART_getReceiveBufferAddress

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	26
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint32_t
EUSCI_A_UART_getReceiveBufferAddress(uint32_t baseAddress)
```

Description:

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.

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Returns:

Address of RX Buffer

10.2.2.8 EUSCI_A_UART_getTransmitBufferAddress

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	26
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint32_t
EUSCI_A_UART_getTransmitBufferAddress(uint32_t baseAddress)
```

Description:

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

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

Returns:

Address of TX Buffer

10.2.2.9 EUSCI_A_UART_initAdvance

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.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	214
CCS 4.2.1	Size	158
CCS 4.2.1	Speed	158
IAR 5.51.6	None	198
IAR 5.51.6	Size	168
IAR 5.51.6	Speed	166
MSPGCC 4.8.0	None	382
MSPGCC 4.8.0	Size	160
MSPGCC 4.8.0	Speed	280

Prototype:

```
bool
EUSCI_A_UART_initAdvance(uint32_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)
```

Description:

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
```

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

10.2.2.10 EUSCI_A_UART_queryStatusFlags

Gets the current UART status flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint8_t
EUSCI_A_UART_queryStatusFlags(uint32_t baseAddress,
                              uint8_t mask)
```

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

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

10.2.2.11 EUSCI_A_UART_receiveData

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	34
CCS 4.2.1	Size	20
CCS 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	58
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Prototype:

```
uint8_t
EUSCI_A_UART_receiveData(uint32_t baseAddress)
```

Description:

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

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.

10.2.2.12 EUSCI_A_UART_resetDormant

Re-enables UART module from dormant mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
EUSCI_A_UART_resetDormant(uint32_t baseAddress)
```

Description:

Not dormant. All received characters set UCRXIFG.

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

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

Returns:

None

10.2.2.13 EUSCI_A_UART_selectDeglitchTime

Sets the deglitch time.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	40
CCS 4.2.1	Size	14
CCS 4.2.1	Speed	14
IAR 5.51.6	None	24
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

Prototype:

```
void
EUSCI_A_UART_selectDeglitchTime(uint32_t baseAddress,
                                uint32_t deglitchTime)
```

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

10.2.2.14 void EUSCI_A_UART_setDormant (uint32_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
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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 EUSCI_A_UART module.

Description:
Modified bits of **UCAxCTL1** register.

Returns:
None

10.2.2.15 EUSCI_A_UART_transmitAddress

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	38
CCS 4.2.1	Size	12
CCS 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	54
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Prototype:
```
void
EUSCI_A_UART_transmitAddress(uint32_t baseAddress,
                             uint8_t transmitAddress)
```

Parameters:
baseAddress is the base address of the EUSCI_A_UART module.
transmitAddress is the next byte to be transmitted

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

Returns:
None

10.2.2.16 EUSCI_A_UART_transmitBreak

Transmit break.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	64
CCS 4.2.1	Size	42
CCS 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	96
MSPGCC 4.8.0	Size	54
MSPGCC 4.8.0	Speed	60

Prototype:

```
void
EUSCI_A_UART_transmitBreak(uint32_t baseAddress)
```

Description:

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.

Parameters:

baseAddress is the base address of the EUSCI_A_UART module.

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

Returns:

None

10.2.2.17 EUSCI_A_UART_transmitData

Transmits a byte from the UART Module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	44
CCS 4.2.1	Size	20
CCS 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	72
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

Prototype:

```
void
EUSCI_A_UART_transmitData(uint32_t baseAddress,
                          uint8_t transmitData)
```

Description:

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

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

10.3 Programming Example

The following example shows how to use the EUSI_UART API to initialize the EUSI_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);
```

11 EUSCI Synchronous Peripheral Interface (EUSCI_A_SPI)

Introduction 95
API Functions 95
Programming Example 107

11.1 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 CCS 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 withing 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
CCS 4.2.1	None	648
CCS 4.2.1	Size	282
CCS 4.2.1	Speed	278
IAR 5.51.6	None	374
IAR 5.51.6	Size	276
IAR 5.51.6	Speed	276
MSPGCC 4.8.0	None	940
MSPGCC 4.8.0	Size	320
MSPGCC 4.8.0	Speed	320

11.2 Functions

Functions

- void EUSCI_A_SPI_changeClockPhasePolarity (uint32_t baseAddress, uint16_t clockPhase, uint16_t clockPolarity)
- void EUSCI_A_SPI_clearInterruptFlag (uint32_t baseAddress, uint8_t mask)
- void EUSCI_A_SPI_disable (uint32_t baseAddress)
- void EUSCI_A_SPI_disableInterrupt (uint32_t baseAddress, uint8_t mask)
- void EUSCI_A_SPI_enable (uint32_t baseAddress)
- void EUSCI_A_SPI_enableInterrupt (uint32_t baseAddress, uint8_t mask)
- uint8_t EUSCI_A_SPI_getInterruptStatus (uint32_t baseAddress, uint8_t mask)
- uint32_t EUSCI_A_SPI_getReceiveBufferAddress (uint32_t baseAddress)
- uint32_t EUSCI_A_SPI_getTransmitBufferAddress (uint32_t baseAddress)
- uint16_t EUSCI_A_SPI_isBusy (uint32_t baseAddress)

- void EUSCI_A_SPI_masterChangeClock (uint32_t baseAddress, uint32_t clockSourceFrequency, uint32_t desiredSpiClock)
- void EUSCI_A_SPI_masterInit (uint32_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 (uint32_t baseAddress)
- void EUSCI_A_SPI_select4PinFunctionality (uint32_t baseAddress, uint8_t select4PinFunctionality)
- void EUSCI_A_SPI_slaveInit (uint32_t baseAddress, uint16_t msbFirst, uint16_t clockPhase, uint16_t clockPolarity, uint16_t spiMode)
- void EUSCI_A_SPI_transmitData (uint32_t baseAddress, uint8_t transmitData)

11.2.1 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()

11.2.2 Function Documentation

11.2.2.1 EUSCI_A_SPI_changeClockPhasePolarity

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
CCS 4.2.1	None	56
CCS 4.2.1	Size	22
CCS 4.2.1	Speed	22
IAR 5.51.6	None	32
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

Prototype:

```
void
EUSCI_A_SPI_changeClockPhasePolarity(uint32_t baseAddress,
                                     uint16_t clockPhase,
                                     uint16_t clockPolarity)
```

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]

Description:

Modified bits are **UCCKPL**, **UCCKPH** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

11.2.2.2 EUSCI_A_SPI_clearInterruptFlag

Clears the selected SPI interrupt status flag.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	54
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
EUSCI_A_SPI_clearInterruptFlag(uint32_t baseAddress,
                               uint8_t mask)
```

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**

Description:

Modified bits of **UCAxIFG** register.

Returns:

None

11.2.2.3 EUSCI_A_SPI_disable

Disables the SPI block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
EUSCI_A_SPI_disable(uint32_t baseAddress)
```

Description:

This will disable operation of the SPI block.

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

11.2.2.4 EUSCI_A_SPI_disableInterrupt

Disables individual SPI interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	54
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
EUSCI_A_SPI_disableInterrupt(uint32_t baseAddress,
                             uint8_t mask)
```

Description:

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.

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

11.2.2.5 EUSCI_A_SPI_enable

Enables the SPI block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
EUSCI_A_SPI_enable(uint32_t baseAddress)
```

Description:

This will enable operation of the SPI block.

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:
None

11.2.2.6 EUSCI_A_SPI_enableInterrupt

Enables individual SPI interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	50
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:
```
void
EUSCI_A_SPI_enableInterrupt(uint32_t baseAddress,
                            uint8_t mask)
```

Description:
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.

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

11.2.2.7 EUSCI_A_SPI_getInterruptStatus

Gets the current SPI interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint8_t
EUSCI_A_SPI_getInterruptStatus(uint32_t baseAddress,
                               uint8_t mask)
```

Description:

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

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

11.2.2.8 EUSCI_A_SPI_getReceiveBufferAddress

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	26
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint32_t
EUSCI_A_SPI_getReceiveBufferAddress(uint32_t baseAddress)
```

Description:

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.

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Returns:

the address of the RX Buffer

11.2.2.9 EUSCI_A_SPI_getTransmitBufferAddress

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	26
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint32_t
EUSCI_A_SPI_getTransmitBufferAddress(uint32_t baseAddress)
```

Description:

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

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

Returns:

the address of the TX Buffer

11.2.2.10 EUSCI_A_SPI_isBusy

Indicates whether or not the SPI bus is busy.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	12
CCS 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	26
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint16_t
EUSCI_A_SPI_isBusy(uint32_t baseAddress)
```

Description:

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

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

11.2.2.11 EUSCI_A_SPI_masterChangeClock

Initializes the SPI Master clock. At the end of this function call, SPI module is left enabled.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	60
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	36
IAR 5.51.6	None	56
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	92
MSPGCC 4.8.0	Size	42
MSPGCC 4.8.0	Speed	42

Prototype:

```
void
EUSCI_A_SPI_masterChangeClock(uint32_t baseAddress,
                              uint32_t clockSourceFrequency,
                              uint32_t desiredSpiClock)
```

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

Description:

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

11.2.2.12 EUSCI_A_SPI_masterInit

Initializes the SPI Master block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	122
CCS 4.2.1	Size	86
CCS 4.2.1	Speed	86
IAR 5.51.6	None	112
IAR 5.51.6	Size	94
IAR 5.51.6	Speed	94
MSPGCC 4.8.0	None	198
MSPGCC 4.8.0	Size	110
MSPGCC 4.8.0	Speed	110

Prototype:

```
void
EUSCI_A_SPI_masterInit(uint32_t baseAddress,
                       uint8_t selectClockSource,
                       uint32_t clockSourceFrequency,
                       uint32_t desiredSpiClock,
                       uint16_t msbFirst,
```

```
                                uint16_t clockPhase,
                                uint16_t clockPolarity,
                                uint16_t spiMode)
```

Description:
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()

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

11.2.2.13 EUSCI_A_SPI_receiveData

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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

Prototype:
```
uint8_t
EUSCI_A_SPI_receiveData(uint32_t baseAddress)
```

Description:
This function reads a byte of data from the SPI receive data Register.

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.

11.2.2.14 EUSCI_A_SPI_select4PinFunctionality

Selects 4Pin Functionality.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	38
CCS 4.2.1	Size	12
CCS 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	58
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:
```
void
EUSCI_A_SPI_select4PinFunctionality(uint32_t baseAddress,
                                    uint8_t select4PinFunctionality)
```

Description:
This function should be invoked only in 4-wire mode. Invoking this function has no effect in 3-wire mode.

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

11.2.2.15 EUSCI_A_SPI_slaveInit

Initializes the SPI Slave block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	62
CCS 4.2.1	Size	38
CCS 4.2.1	Speed	34
IAR 5.51.6	None	42
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	90
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

Prototype:

```
void
EUSCI_A_SPI_slaveInit(uint32_t baseAddress,
                      uint16_t msbFirst,
                      uint16_t clockPhase,
                      uint16_t clockPolarity,
                      uint16_t spiMode)
```

Description:

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()

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

11.2.2.16 EUSCI_A_SPI_transmitData

Transmits a byte from the SPI Module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
EUSCI_A_SPI_transmitData(uint32_t baseAddress,
                         uint8_t transmitData)
```

Description:

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

Parameters:

baseAddress is the base address of the EUSCI_A_SPI module.

transmitData data to be transmitted from the SPI module

Returns:

None

11.3 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
        );
```

12 EUSCI Synchronous Peripheral Interface (EUSCI_B_SPI)

Introduction 108
API Functions 108
Programming Example 120

12.1 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 CCS 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 withing 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
CCS 4.2.1	None	642
CCS 4.2.1	Size	278
CCS 4.2.1	Speed	274
IAR 5.51.6	None	364
IAR 5.51.6	Size	268
IAR 5.51.6	Speed	268
MSPGCC 4.8.0	None	928
MSPGCC 4.8.0	Size	314
MSPGCC 4.8.0	Speed	314

12.2 Functions

Functions

- void EUSCI_B_SPI_changeClockPhasePolarity (uint32_t baseAddress, uint16_t clockPhase, uint16_t clockPolarity)
- void EUSCI_B_SPI_clearInterruptFlag (uint32_t baseAddress, uint8_t mask)
- void EUSCI_B_SPI_disable (uint32_t baseAddress)
- void EUSCI_B_SPI_disableInterrupt (uint32_t baseAddress, uint8_t mask)
- void EUSCI_B_SPI_enable (uint32_t baseAddress)
- void EUSCI_B_SPI_enableInterrupt (uint32_t baseAddress, uint8_t mask)
- uint8_t EUSCI_B_SPI_getInterruptStatus (uint32_t baseAddress, uint8_t mask)
- uint32_t EUSCI_B_SPI_getReceiveBufferAddress (uint32_t baseAddress)
- uint32_t EUSCI_B_SPI_getTransmitBufferAddress (uint32_t baseAddress)
- uint16_t EUSCI_B_SPI_isBusy (uint32_t baseAddress)

- void EUSCI_B_SPI_masterChangeClock (uint32_t baseAddress, uint32_t clockSourceFrequency, uint32_t desiredSpiClock)
- void EUSCI_B_SPI_masterInit (uint32_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 (uint32_t baseAddress)
- void EUSCI_B_SPI_select4PinFunctionality (uint32_t baseAddress, uint8_t select4PinFunctionality)
- void EUSCI_B_SPI_slaveInit (uint32_t baseAddress, uint16_t msbFirst, uint16_t clockPhase, uint16_t clockPolarity, uint16_t spiMode)
- void EUSCI_B_SPI_transmitData (uint32_t baseAddress, uint8_t transmitData)

12.2.1 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()

12.2.2 Function Documentation

12.2.2.1 EUSCI_B_SPI_changeClockPhasePolarity

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
CCS 4.2.1	None	56
CCS 4.2.1	Size	22
CCS 4.2.1	Speed	22
IAR 5.51.6	None	32
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

Prototype:

```
void
EUSCI_B_SPI_changeClockPhasePolarity(uint32_t baseAddress,
                                     uint16_t clockPhase,
                                     uint16_t clockPolarity)
```

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]

Description:

Modified bits are **UCCKPL**, **UCCKPH** and **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

12.2.2.2 EUSCI_B_SPI_clearInterruptFlag

Clears the selected SPI interrupt status flag.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	54
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
EUSCI_B_SPI_clearInterruptFlag(uint32_t baseAddress,
                               uint8_t mask)
```

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**

Description:

Modified bits of **UCAxIFG** register.

Returns:

None

12.2.2.3 EUSCI_B_SPI_disable

Disables the SPI block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
EUSCI_B_SPI_disable(uint32_t baseAddress)
```

Description:

This will disable operation of the SPI block.

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

12.2.2.4 EUSCI_B_SPI_disableInterrupt

Disables individual SPI interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	54
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
EUSCI_B_SPI_disableInterrupt(uint32_t baseAddress,
                             uint8_t mask)
```

Description:

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.

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

12.2.2.5 EUSCI_B_SPI_enable

Enables the SPI block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
EUSCI_B_SPI_enable(uint32_t baseAddress)
```

Description:

This will enable operation of the SPI block.

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:
None

12.2.2.6 EUSCI_B_SPI_enableInterrupt

Enables individual SPI interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	8
CCS 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	50
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:
```
void
EUSCI_B_SPI_enableInterrupt(uint32_t baseAddress,
                            uint8_t mask)
```

Description:
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.

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

12.2.2.7 EUSCI_B_SPI_getInterruptStatus

Gets the current SPI interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint8_t
EUSCI_B_SPI_getInterruptStatus(uint32_t baseAddress,
                               uint8_t mask)
```

Description:

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

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

12.2.2.8 EUSCI_B_SPI_getReceiveBufferAddress

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	26
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint32_t
EUSCI_B_SPI_getReceiveBufferAddress(uint32_t baseAddress)
```

Description:

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.

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Returns:

the address of the RX Buffer

12.2.2.9 EUSCI_B_SPI_getTransmitBufferAddress

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	26
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint32_t
EUSCI_B_SPI_getTransmitBufferAddress(uint32_t baseAddress)
```

Description:

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

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

Returns:

the address of the TX Buffer

12.2.2.10 EUSCI_B_SPI_isBusy

Indicates whether or not the SPI bus is busy.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	12
CCS 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	24
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint16_t
EUSCI_B_SPI_isBusy(uint32_t baseAddress)
```

Description:

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

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

12.2.2.11 EUSCI_B_SPI_masterChangeClock

Initializes the SPI Master clock. At the end of this function call, SPI module is left enabled.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	60
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	36
IAR 5.51.6	None	56
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	92
MSPGCC 4.8.0	Size	42
MSPGCC 4.8.0	Speed	42

Prototype:

```
void
EUSCI_B_SPI_masterChangeClock(uint32_t baseAddress,
                              uint32_t clockSourceFrequency,
                              uint32_t desiredSpiClock)
```

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

Description:

Modified bits are **UCSWRST** of **UCAxCTLW0** register.

Returns:

None

12.2.2.12 EUSCI_B_SPI_masterInit

Initializes the SPI Master block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	116
CCS 4.2.1	Size	82
CCS 4.2.1	Speed	82
IAR 5.51.6	None	104
IAR 5.51.6	Size	88
IAR 5.51.6	Speed	88
MSPGCC 4.8.0	None	188
MSPGCC 4.8.0	Size	104
MSPGCC 4.8.0	Speed	104

Prototype:

```
void
EUSCI_B_SPI_masterInit(uint32_t baseAddress,
                       uint8_t selectClockSource,
                       uint32_t clockSourceFrequency,
                       uint32_t desiredSpiClock,
                       uint16_t msbFirst,
```

```
                                uint16_t clockPhase,
                                uint16_t clockPolarity,
                                uint16_t spiMode)
```

Description:
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()

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

12.2.2.13 EUSCI_B_SPI_receiveData

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

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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

Prototype:
```
uint8_t
EUSCI_B_SPI_receiveData(uint32_t baseAddress)
```

Description:
This function reads a byte of data from the SPI receive data Register.

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.

12.2.2.14 EUSCI_B_SPI_select4PinFunctionality

Selects 4Pin Functionality.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	38
CCS 4.2.1	Size	12
CCS 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	58
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:
```
void
EUSCI_B_SPI_select4PinFunctionality(uint32_t baseAddress,
                                    uint8_t select4PinFunctionality)
```

Description:
This function should be invoked only in 4-wire mode. Invoking this function has no effect in 3-wire mode.

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

12.2.2.15 EUSCI_B_SPI_slaveInit

Initializes the SPI Slave block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	62
CCS 4.2.1	Size	38
CCS 4.2.1	Speed	34
IAR 5.51.6	None	42
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	90
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

Prototype:

```
void
EUSCI_B_SPI_slaveInit(uint32_t baseAddress,
                      uint16_t msbFirst,
                      uint16_t clockPhase,
                      uint16_t clockPolarity,
                      uint16_t spiMode)
```

Description:

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()

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

12.2.2.16 EUSCI_B_SPI_transmitData

Transmits a byte from the SPI Module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
EUSCI_B_SPI_transmitData(uint32_t baseAddress,
                         uint8_t transmitData)
```

Description:

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

Parameters:

baseAddress is the base address of the EUSCI_B_SPI module.

transmitData data to be transmitted from the SPI module

Returns:

None

12.3 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
        );
```

13 EUSCI Inter-Integrated Circuit (EUSCI_B_I2C)

Introduction 121
API Functions 123
Programming Example 148

13.1 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.

13.1.1 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.

13.1.2 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 CCS 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 withing 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
CCS 4.2.1	None	1974
CCS 4.2.1	Size	972
CCS 4.2.1	Speed	970
IAR 5.51.6	None	1430
IAR 5.51.6	Size	936
IAR 5.51.6	Speed	1134
MSPGCC 4.8.0	None	2976
MSPGCC 4.8.0	Size	1300
MSPGCC 4.8.0	Speed	1358

13.2 API Functions

Functions

- void EUSCI_B_I2C_clearInterruptFlag (uint32_t baseAddress, uint16_t mask)
- void EUSCI_B_I2C_disable (uint32_t baseAddress)
- void EUSCI_B_I2C_disableInterrupt (uint32_t baseAddress, uint16_t mask)
- void EUSCI_B_I2C_disableMultiMasterMode (uint32_t baseAddress)
- void EUSCI_B_I2C_enable (uint32_t baseAddress)
- void EUSCI_B_I2C_enableInterrupt (uint32_t baseAddress, uint16_t mask)
- void EUSCI_B_I2C_enableMultiMasterMode (uint32_t baseAddress)
- uint16_t EUSCI_B_I2C_getInterruptStatus (uint32_t baseAddress, uint16_t mask)
- uint8_t EUSCI_B_I2C_getMode (uint32_t baseAddress)
- uint32_t EUSCI_B_I2C_getReceiveBufferAddress (uint32_t baseAddress)
- uint32_t EUSCI_B_I2C_getTransmitBufferAddress (uint32_t baseAddress)
- uint16_t EUSCI_B_I2C_isBusBusy (uint32_t baseAddress)
- void EUSCI_B_I2C_masterInit (uint32_t baseAddress, uint8_t selectClockSource, uint32_t i2cClk, uint32_t dataRate, uint8_t byteCounterThreshold, uint8_t autoSTOPGeneration)
- uint16_t EUSCI_B_I2C_masterIsStartSent (uint32_t baseAddress)
- uint16_t EUSCI_B_I2C_masterIsStopSent (uint32_t baseAddress)
- uint8_t EUSCI_B_I2C_masterMultiByteReceiveFinish (uint32_t baseAddress)
- bool EUSCI_B_I2C_masterMultiByteReceiveFinishWithTimeout (uint32_t baseAddress, uint8_t *txData, uint32_t timeout)
- uint8_t EUSCI_B_I2C_masterMultiByteReceiveNext (uint32_t baseAddress)
- void EUSCI_B_I2C_masterMultiByteReceiveStop (uint32_t baseAddress)
- void EUSCI_B_I2C_masterMultiByteSendFinish (uint32_t baseAddress, uint8_t txData)
- bool EUSCI_B_I2C_masterMultiByteSendFinishWithTimeout (uint32_t baseAddress, uint8_t txData, uint32_t timeout)
- void EUSCI_B_I2C_masterMultiByteSendNext (uint32_t baseAddress, uint8_t txData)
- bool EUSCI_B_I2C_masterMultiByteSendNextWithTimeout (uint32_t baseAddress, uint8_t txData, uint32_t timeout)
- void EUSCI_B_I2C_masterMultiByteSendStart (uint32_t baseAddress, uint8_t txData)
- bool EUSCI_B_I2C_masterMultiByteSendStartWithTimeout (uint32_t baseAddress, uint8_t txData, uint32_t timeout)
- void EUSCI_B_I2C_masterMultiByteSendStop (uint32_t baseAddress)
- bool EUSCI_B_I2C_masterMultiByteSendStopWithTimeout (uint32_t baseAddress, uint32_t timeout)
- uint8_t EUSCI_B_I2C_masterReceiveSingleByte (uint32_t baseAddress)
- void EUSCI_B_I2C_masterReceiveStart (uint32_t baseAddress)
- void EUSCI_B_I2C_masterSendSingleByte (uint32_t baseAddress, uint8_t txData)
- bool EUSCI_B_I2C_masterSendSingleByteWithTimeout (uint32_t baseAddress, uint8_t txData, uint32_t timeout)
- void EUSCI_B_I2C_masterSendStart (uint32_t baseAddress)
- uint8_t EUSCI_B_I2C_masterSingleReceive (uint32_t baseAddress)
- void EUSCI_B_I2C_setMode (uint32_t baseAddress, uint8_t mode)
- void EUSCI_B_I2C_setSlaveAddress (uint32_t baseAddress, uint8_t slaveAddress)
- uint8_t EUSCI_B_I2C_slaveDataGet (uint32_t baseAddress)
- void EUSCI_B_I2C_slaveDataPut (uint32_t baseAddress, uint8_t transmitData)
- void EUSCI_B_I2C_slaveInit (uint32_t baseAddress, uint8_t slaveAddress, uint8_t slaveAddressOffset, uint32_t slaveOwnAddressEnable)

13.2.1 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

13.2.2 Function Documentation

13.2.2.1 EUSCI_B_I2C_clearInterruptFlag

Clears I2C interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	34
CCS 4.2.1	Size	6
CCS 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	50
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
EUSCI_B_I2C_clearInterruptFlag(uint32_t baseAddress,
                               uint16_t mask)
```

Description:

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.

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

13.2.2.2 EUSCI_B_I2C_disable

Disables the I2C block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
EUSCI_B_I2C_disable(uint32_t baseAddress)
```

Description:
This will disable operation of the I2C block.

Parameters:
baseAddress is the base address of the USCI I2C module.

Modified bits are **UCSWRST** of **UCBxCTLW0** register.

Returns:
None

13.2.2.3 EUSCI_B_I2C_disableInterrupt

Disables individual I2C interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	34
CCS 4.2.1	Size	6
CCS 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	50
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
EUSCI_B_I2C_disableInterrupt(uint32_t baseAddress,
                             uint16_t mask)
```

Description:
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.

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

13.2.2.4 EUSCI_B_I2C_disableMultiMasterMode

Disables Multi Master Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	12
CCS 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	40
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:
```
void
EUSCI_B_I2C_disableMultiMasterMode(uint32_t baseAddress)
```

Description:
At the end of this function, the I2C module is still disabled till EUSCI_B_I2C_enable is invoked

Parameters:
baseAddress is the base address of the I2C module.

Modified bits are **UCSWRST** and **UCMM** of **UCBxCTLW0** register.

Returns:
None

13.2.2.5 EUSCI_B_I2C_enable

Enables the I2C block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
EUSCI_B_I2C_enable(uint32_t baseAddress)
```

Description:

This will enable operation of the I2C block.

Parameters:

baseAddress is the base address of the USCI I2C module.

Modified bits are **UCSWRST** of **UCBxCTLW0** register.

Returns:

None

13.2.2.6 EUSCI_B_I2C_enableInterrupt

Enables individual I2C interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	34
CCS 4.2.1	Size	6
CCS 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	46
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
EUSCI_B_I2C_enableInterrupt(uint32_t baseAddress,
                            uint16_t mask)
```

Description:

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.

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

13.2.2.7 EUSCI_B_I2C_enableMultiMasterMode

Enables Multi Master Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	12
CCS 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	40
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:
```
void
EUSCI_B_I2C_enableMultiMasterMode(uint32_t baseAddress)
```

Description:
At the end of this function, the I2C module is still disabled till EUSCI_B_I2C_enable is invoked

Parameters:
baseAddress is the base address of the I2C module.

Modified bits are **UCSWRST** and **UCMM** of **UCBxCTLW0** register.

Returns:
None

13.2.2.8 EUSCI_B_I2C_getInterruptStatus

Gets the current I2C interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	8
CCS 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	34
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint16_t
EUSCI_B_I2C_getInterruptStatus(uint32_t baseAddress,
                               uint16_t mask)
```

Description:

This returns the interrupt status for the I2C module based on which flag is passed.

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

13.2.2.9 EUSCI_B_I2C_getMode

Gets the mode of the I2C device.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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	32
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint8_t
EUSCI_B_I2C_getMode(uint32_t baseAddress)
```

Description:

Current I2C transmit/receive mode.

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

13.2.2.10 EUSCI_B_I2C_getReceiveBufferAddress

Returns the address of the RX Buffer of the I2C for the DMA module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	26
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint32_t
EUSCI_B_I2C_getReceiveBufferAddress(uint32_t baseAddress)
```

Description:

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.

Parameters:
baseAddress is the base address of the I2C module.

Returns:
The address of the I2C RX Buffer

13.2.2.11 EUSCI_B_I2C_getTransmitBufferAddress

Returns the address of the TX Buffer of the I2C for the DMA module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	26
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:
```
uint32_t
EUSCI_B_I2C_getTransmitBufferAddress(uint32_t baseAddress)
```

Description:
Returns the address of the I2C TX Buffer. This can be used in conjunction with the DMA to obtain transmitted data directly from memory.

Parameters:
baseAddress is the base address of the I2C module.

Returns:
The address of the I2C TX Buffer

13.2.2.12 EUSCI_B_I2C_isBusBusy

Indicates whether or not the I2C bus is busy.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	12
CCS 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	24
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:
```
uint16_t
EUSCI_B_I2C_isBusBusy(uint32_t baseAddress)
```

Description:
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.

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

13.2.2.13 EUSCI_B_I2C_masterInit

Initializes the I2C Master block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	122
CCS 4.2.1	Size	78
CCS 4.2.1	Speed	78
IAR 5.51.6	None	118
IAR 5.51.6	Size	84
IAR 5.51.6	Speed	84
MSPGCC 4.8.0	None	184
MSPGCC 4.8.0	Size	100
MSPGCC 4.8.0	Speed	100

Prototype:
```
void
EUSCI_B_I2C_masterInit(uint32_t baseAddress,
                       uint8_t selectClockSource,
                       uint32_t i2cClk,
                       uint32_t dataRate,
                       uint8_t byteCounterThreshold,
                       uint8_t autoSTOPGeneration)
```

Description:
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.

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

13.2.2.14 EUSCI_B_I2C_masterIsStartSent

Indicates whether Start got sent.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	8
CCS 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	20
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
uint16_t
EUSCI_B_I2C_masterIsStartSent(uint32_t baseAddress)
```

Description:

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.

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

13.2.2.15 EUSCI_B_I2C_masterIsStopSent

Indicates whether STOP got sent.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	8
CCS 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	20
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
uint16_t
EUSCI_B_I2C_masterIsStopSent(uint32_t baseAddress)
```

Description:

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.

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

13.2.2.16 EUSCI_B_I2C_masterMultiByteReceiveFinish

Finishes multi-byte reception at the Master end.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	46
CCS 4.2.1	Size	26
CCS 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	70
MSPGCC 4.8.0	Size	34
MSPGCC 4.8.0	Speed	40

Prototype:
```
uint8_t
EUSCI_B_I2C_masterMultiByteReceiveFinish(uint32_t baseAddress)
```

Description:
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.

Parameters:
baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTP** of **UCBxCTLW0** register.

Returns:
Received byte at Master end.

13.2.2.17 EUSCI_B_I2C_masterMultiByteReceiveFinishWithTimeout

Finishes multi-byte reception at the Master end with timeout.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	142
CCS 4.2.1	Size	80
CCS 4.2.1	Speed	80
IAR 5.51.6	None	106
IAR 5.51.6	Size	76
IAR 5.51.6	Speed	98
MSPGCC 4.8.0	None	202
MSPGCC 4.8.0	Size	126
MSPGCC 4.8.0	Speed	144

Prototype:

```
bool
EUSCI_B_I2C_masterMultiByteReceiveFinishWithTimeout(uint32_t baseAddress,
                                                    uint8_t *txData,
                                                    uint32_t timeout)
```

Description:

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.

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

13.2.2.18 EUSCI_B_I2C_masterMultiByteReceiveNext

Starts multi-byte reception at the Master end one byte at a time.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint8_t
EUSCI_B_I2C_masterMultiByteReceiveNext(uint32_t baseAddress)
```

Description:

This function is used by the Master module to receive each byte of a multi- byte reception. This function reads currently received byte.

Parameters:

baseAddress is the base address of the I2C Master module.

Returns:

Received byte at Master end.

13.2.2.19 EUSCI_B_I2C_masterMultiByteReceiveStop

Sends the STOP at the end of a multi-byte reception at the Master end.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
EUSCI_B_I2C_masterMultiByteReceiveStop(uint32_t baseAddress)
```

Description:

This function is used by the Master module to initiate STOP

Parameters:

baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTP** of **UCBxCTLW0** register.

Returns:

None

13.2.2.20 EUSCI_B_I2C_masterMultiByteSendFinish

Finishes multi-byte transmission from Master to Slave.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	58
CCS 4.2.1	Size	30
CCS 4.2.1	Speed	30
IAR 5.51.6	None	56
IAR 5.51.6	Size	52
IAR 5.51.6	Speed	50
MSPGCC 4.8.0	None	106
MSPGCC 4.8.0	Size	44
MSPGCC 4.8.0	Speed	44

Prototype:

```
void
EUSCI_B_I2C_masterMultiByteSendFinish(uint32_t baseAddress,
                                      uint8_t txData)
```

Description:

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.

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

13.2.2.21 EUSCI_B_I2C_masterMultiByteSendFinishWithTimeout

Finishes multi-byte transmission from Master to Slave with timeout.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	146
CCS 4.2.1	Size	88
CCS 4.2.1	Speed	88
IAR 5.51.6	None	126
IAR 5.51.6	Size	104
IAR 5.51.6	Speed	120
MSPGCC 4.8.0	None	216
MSPGCC 4.8.0	Size	124
MSPGCC 4.8.0	Speed	132

Prototype:

```
bool
EUSCI_B_I2C_masterMultiByteSendFinishWithTimeout(uint32_t baseAddress,
                                                 uint8_t txData,
                                                 uint32_t timeout)
```

Description:

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.

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.

13.2.2.22 EUSCI_B_I2C_masterMultiByteSendNext

Continues multi-byte transmission from Master to Slave.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	44
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	20
IAR 5.51.6	None	38
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

Prototype:

```
void
EUSCI_B_I2C_masterMultiByteSendNext(uint32_t baseAddress,
                                    uint8_t txData)
```

Description:
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.

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

13.2.2.23 EUSCI_B_I2C_masterMultiByteSendNextWithTimeout

Continues multi-byte transmission from Master to Slave with timeout.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	88
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	50
IAR 5.51.6	None	80
IAR 5.51.6	Size	50
IAR 5.51.6	Speed	76
MSPGCC 4.8.0	None	132
MSPGCC 4.8.0	Size	92
MSPGCC 4.8.0	Speed	104

Prototype:
```
bool
EUSCI_B_I2C_masterMultiByteSendNextWithTimeout(uint32_t baseAddress,
                                               uint8_t txData,
                                               uint32_t timeout)
```

Description:
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.

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.

13.2.2.24 EUSCI_B_I2C_masterMultiByteSendStart

Starts multi-byte transmission from Master to Slave.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	76
CCS 4.2.1	Size	38
CCS 4.2.1	Speed	38
IAR 5.51.6	None	66
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	52
MSPGCC 4.8.0	None	138
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	50

Prototype:

```
void
EUSCI_B_I2C_masterMultiByteSendStart(uint32_t baseAddress,
                                     uint8_t txData)
```

Description:
This function is used by the master module to start a multi byte transaction.

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

13.2.2.25 EUSCI_B_I2C_masterMultiByteSendStartWithTimeout

Starts multi-byte transmission from Master to Slave with timeout.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	124
CCS 4.2.1	Size	72
CCS 4.2.1	Speed	72
IAR 5.51.6	None	104
IAR 5.51.6	Size	56
IAR 5.51.6	Speed	88
MSPGCC 4.8.0	None	198
MSPGCC 4.8.0	Size	110
MSPGCC 4.8.0	Speed	122

Prototype:

```
bool
EUSCI_B_I2C_masterMultiByteSendStartWithTimeout(uint32_t baseAddress,
                                                uint8_t txData,
                                                uint32_t timeout)
```

Description:
This function is used by the master module to start a multi byte transaction.

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.

13.2.2.26 EUSCI_B_I2C_masterMultiByteSendStop

Send STOP byte at the end of a multi-byte transmission from Master to Slave.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	34
CCS 4.2.1	Size	18
CCS 4.2.1	Speed	18
IAR 5.51.6	None	32
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Prototype:
```
void
EUSCI_B_I2C_masterMultiByteSendStop(uint32_t baseAddress)
```

Description:
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.

Parameters:
baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTP** of **UCBxCTLW0** register.

Returns:
None

13.2.2.27 EUSCI_B_I2C_masterMultiByteSendStopWithTimeout

Send STOP byte at the end of a multi-byte transmission from Master to Slave with timeout.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	76
CCS 4.2.1	Size	44
CCS 4.2.1	Speed	44
IAR 5.51.6	None	66
IAR 5.51.6	Size	62
IAR 5.51.6	Speed	62
MSPGCC 4.8.0	None	110
MSPGCC 4.8.0	Size	56
MSPGCC 4.8.0	Speed	52

Prototype:

```
bool
EUSCI_B_I2C_masterMultiByteSendStopWithTimeout(uint32_t baseAddress,
                                               uint32_t timeout)
```

Description:

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.

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.

13.2.2.28 EUSCI_B_I2C_masterReceiveSingleByte

Does single byte reception from Slave.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	44
CCS 4.2.1	Size	26
CCS 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	72
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

Prototype:

```
uint8_t
EUSCI_B_I2C_masterReceiveSingleByte(uint32_t baseAddress)
```

Description:

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

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.

13.2.2.29 EUSCI_B_I2C_masterReceiveStart

Starts reception at the Master end.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	12
CCS 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	40
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:

```
void
EUSCI_B_I2C_masterReceiveStart(uint32_t baseAddress)
```

Description:
This function is used by the Master module initiate reception of a single byte. This function sends a start.

Parameters:
baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTT** of **UCBxCTLW0** register.

Returns:
None

13.2.2.30 EUSCI_B_I2C_masterSendSingleByte

Does single byte transmission from Master to Slave.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	100
CCS 4.2.1	Size	52
CCS 4.2.1	Speed	52
IAR 5.51.6	None	94
IAR 5.51.6	Size	52
IAR 5.51.6	Speed	68
MSPGCC 4.8.0	None	196
MSPGCC 4.8.0	Size	70
MSPGCC 4.8.0	Speed	70

Prototype:

```
void
EUSCI_B_I2C_masterSendSingleByte(uint32_t baseAddress,
                                 uint8_t txData)
```

Description:
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.

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

13.2.2.31 EUSCI_B_I2C_masterSendSingleByteWithTimeout

Does single byte transmission from Master to Slave with timeout.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	192
CCS 4.2.1	Size	110
CCS 4.2.1	Speed	110
IAR 5.51.6	None	160
IAR 5.51.6	Size	116
IAR 5.51.6	Speed	130
MSPGCC 4.8.0	None	310
MSPGCC 4.8.0	Size	148
MSPGCC 4.8.0	Speed	154

Prototype:

```
bool
EUSCI_B_I2C_masterSendSingleByteWithTimeout(uint32_t baseAddress,
                                            uint8_t txData,
                                            uint32_t timeout)
```

Description:

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.

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.

13.2.2.32 EUSCI_B_I2C_masterSendStart

This function is used by the Master module to initiate START.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
EUSCI_B_I2C_masterSendStart(uint32_t baseAddress)
```

Description:
This function is used by the Master module to initiate START

Parameters:
baseAddress is the base address of the I2C Master module.

Modified bits are **UCTXSTT** of **UCBxCTLW0** register.

Returns:
None

13.2.2.33 EUSCI_B_I2C_masterSingleReceive

receives a byte that has been sent to the I2C Master Module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	34
CCS 4.2.1	Size	20
CCS 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	58
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Prototype:
```
uint8_t
EUSCI_B_I2C_masterSingleReceive(uint32_t baseAddress)
```

Description:
This function reads a byte of data from the I2C receive data Register.

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.

13.2.2.34 EUSCI_B_I2C_setMode

Sets the mode of the I2C device.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	40
CCS 4.2.1	Size	14
CCS 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	60
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Prototype:

```
void
EUSCI_B_I2C_setMode(uint32_t baseAddress,
                    uint8_t mode)
```

Description:

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.

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

13.2.2.35 EUSCI_B_I2C_setSlaveAddress

Sets the address that the I2C Master will place on the bus.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	38
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
EUSCI_B_I2C_setSlaveAddress(uint32_t baseAddress,
                            uint8_t slaveAddress)
```

Description:

This function will set the address that the I2C Master will place on the bus when initiating a transaction.

Parameters:

baseAddress is the base address of the USCI I2C module.

slaveAddress 7-bit slave address

Modified bits of **UCBxI2CSA** register.

Returns:

None

13.2.2.36 EUSCI_B_I2C_slaveDataGet

Receives a byte that has been sent to the I2C Module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint8_t
EUSCI_B_I2C_slaveDataGet(uint32_t baseAddress)
```

Description:

This function reads a byte of data from the I2C receive data Register.

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.

13.2.2.37 EUSCI_B_I2C_slaveDataPut

Transmits a byte from the I2C Module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	8
CCS 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	38
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
EUSCI_B_I2C_slaveDataPut(uint32_t baseAddress,
                         uint8_t transmitData)
```

Description:

This function will place the supplied data into I2C transmit data register to start transmission.

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

13.2.2.38 EUSCI_B_I2C_slaveInit

Initializes the I2C Slave block.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	66
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	34
IAR 5.51.6	None	52
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	40
MSPGCC 4.8.0	Speed	40

Prototype:
```
void
EUSCI_B_I2C_slaveInit(uint32_t baseAddress,
                      uint8_t slaveAddress,
                      uint8_t slaveAddressOffset,
                      uint32_t slaveOwnAddressEnable)
```

Description:
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.

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

13.3 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);
```

14 FRAM Controller

Introduction 150
API Functions 150
Programming Example 156

14.1 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 CCS 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 withing 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
CCS 4.2.1	None	384
CCS 4.2.1	Size	142
CCS 4.2.1	Speed	144
IAR 5.51.6	None	162
IAR 5.51.6	Size	118
IAR 5.51.6	Speed	350
MSPGCC 4.8.0	None	524
MSPGCC 4.8.0	Size	204
MSPGCC 4.8.0	Speed	306

14.2 API Functions

Functions

- void FRAM_configureWaitStateControl (uint8_t manualWaitState, uint8_t accessTime, uint8_t prechargeTime)
- 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)

14.2.1 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 give status of FRAM.

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 status is given by

- FRAM_status()

14.2.2 Function Documentation

14.2.2.1 FRAM_configureWaitStateControl

Configures the wait state control of the FRAM module Configures the wait state control of the FRAM module. If using the FRAM_AUTO_MODE the values for accessTime and prechargeTime do not matter.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	20
IAR 5.51.6	None	18
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	16
MSPGCC 4.8.0	None	74
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Prototype:

```
void
FRAM_configureWaitStateControl(uint8_t manualWaitState,
                               uint8_t accessTime,
                               uint8_t prechargeTime)
```

Parameters:

manualWaitState chooses if the wait state control is manual or automatic Valid values are:

- **FRAM_AUTO_MODE**
- **FRAM_MANUAL_MODE**

accessTime 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**
- **FRAM_PRECHARGE_TIME_CYCLES_0**
- **FRAM_PRECHARGE_TIME_CYCLES_1**
- **FRAM_PRECHARGE_TIME_CYCLES_2**
- **FRAM_PRECHARGE_TIME_CYCLES_3**
- **FRAM_PRECHARGE_TIME_CYCLES_4**
- **FRAM_PRECHARGE_TIME_CYCLES_5**
- **FRAM_PRECHARGE_TIME_CYCLES_6**
- **FRAM_PRECHARGE_TIME_CYCLES_7**

prechargeTime

14.2.2.2 void FRAM_disableInterrupt (uint16_t *interruptMask*)

Disables selected FRAM interrupt sources.

If header file contains legacy definitions: FRAM_PUC_ON_DOUBLE_BIT_ERROR - Enable PUC reset if FRAM detects double bit error is detected. FRAM_DOUBLE_BIT_ERROR_INTERRUPT - Interrupts when a double bit error is detected. FRAM_SINGLE_BIT_ERROR_INTERRUPT - Interrupts when a single bit error is detected.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	20
IAR 5.51.6	None	26
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	62
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

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_VIOLATION_INTERRUPT** - Interrupts when an access violation occurs.
- **FRAM_ACCESS_TIME_ERROR_INTERRUPT** - Interrupts when an access time error occurs.

Returns:

None

14.2.2.3 void FRAM_enableInterrupt (uint8_t *interruptMask*)

Enables selected FRAM interrupt sources.

If header file contains legacy definitions: FRAM_PUC_ON_DOUBLE_BIT_ERROR - Enable PUC reset if FRAM detects double bit error is detected. FRAM_DOUBLE_BIT_ERROR_INTERRUPT - Interrupts when a double bit error is detected. FRAM_SINGLE_BIT_ERROR_INTERRUPT - Interrupts when a single bit error is detected.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	38
CCS 4.2.1	Size	22
CCS 4.2.1	Speed	22
IAR 5.51.6	None	28
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	62
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

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_VIOLATION_INTERRUPT** - Interrupts when an access violation occurs.
- **FRAM_ACCESS_TIME_ERROR_INTERRUPT** - Interrupts when an access time error occurs.

Returns:

None

14.2.2.4 uint8_t FRAM_getInterruptStatus (uint16_t *interruptFlagMask*)

Returns the status of the selected FRAM interrupt flags.

If header file contains legacy definitions: FRAM_SINGLE_BIT_ERROR_FLAG - Interrupt flag is set if a correctable bit error has been detected and corrected in the FRAM memory error detection logic. FRAM_DOUBLE_BIT_ERROR_FLAG .- Interrupt flag is set if an uncorrectable bit error has been detected in the FRAM memory error detection logic.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	16
CCS 4.2.1	Size	8
CCS 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.
- **FRAM_ACCESS_VIOLATION_FLAG** - Interrupt flag is set if an access violation is triggered.

Returns:

The current interrupt flag status for the corresponding mask. Return 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.
- **FRAM_ACCESS_VIOLATION_FLAG** Interrupt flag is set if an access violation is triggered.

indicating the status of the masked flags

14.2.2.5 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
CCS 4.2.1	None	60
CCS 4.2.1	Size	20
CCS 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

14.2.2.6 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
CCS 4.2.1	None	62
CCS 4.2.1	Size	16
CCS 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

14.2.2.7 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
CCS 4.2.1	None	68
CCS 4.2.1	Size	20
CCS 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

14.2.2.8 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
CCS 4.2.1	None	62
CCS 4.2.1	Size	16
CCS 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

14.3 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);
```

15 GPIO

Introduction 157
API Functions 158
Programming Example 175

15.1 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 CCS 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 withing 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
CCS 4.2.1	None	4066
CCS 4.2.1	Size	2032
CCS 4.2.1	Speed	2080
IAR 5.51.6	None	3020
IAR 5.51.6	Size	2138
IAR 5.51.6	Speed	3368
MSPGCC 4.8.0	None	7636
MSPGCC 4.8.0	Size	2836
MSPGCC 4.8.0	Speed	2612

15.2 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)

15.2.1 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_disbleInterrupt()
- GPIO_clearInterruptFlag()
- GPIO_getInterruptStatus()
- GPIO_interruptEdgeSelect()

The GPIO pin state is accessed with

- GPIO_setOutputHighOnPin()
- GPIO_setOutputLowOnPin()
- GPIO_toggleOutputOnPin()
- GPIO_getInputPinValue()

15.2.2 Function Documentation

15.2.2.1 GPIO_clearInterruptFlag

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
CCS 4.2.1	None	178
CCS 4.2.1	Size	102
CCS 4.2.1	Speed	102
IAR 5.51.6	None	134
IAR 5.51.6	Size	118
IAR 5.51.6	Speed	160
MSPGCC 4.8.0	None	284
MSPGCC 4.8.0	Size	144
MSPGCC 4.8.0	Speed	120

Prototype:

```
void
GPIO_clearInterruptFlag(uint8_t selectedPort,
                        uint16_t selectedPins)
```

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**

Description:

Modified bits of **PxIFG** register.

Returns:

None

15.2.2.2 GPIO_disableInterrupt

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
CCS 4.2.1	None	178
CCS 4.2.1	Size	102
CCS 4.2.1	Speed	102
IAR 5.51.6	None	134
IAR 5.51.6	Size	118
IAR 5.51.6	Speed	158
MSPGCC 4.8.0	None	284
MSPGCC 4.8.0	Size	144
MSPGCC 4.8.0	Speed	120

Prototype:

```
void
GPIO_disableInterrupt(uint8_t selectedPort,
                      uint16_t selectedPins)
```

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**

Description:

Modified bits of **PxIE** register.

Returns:

None

15.2.2.3 GPIO_enableInterrupt

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
CCS 4.2.1	None	178
CCS 4.2.1	Size	102
CCS 4.2.1	Speed	102
IAR 5.51.6	None	134
IAR 5.51.6	Size	116
IAR 5.51.6	Speed	164
MSPGCC 4.8.0	None	264
MSPGCC 4.8.0	Size	144
MSPGCC 4.8.0	Speed	120

Prototype:

```
void
GPIO_enableInterrupt(uint8_t selectedPort,
                     uint16_t selectedPins)
```

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**

Description:

Modified bits of **PxIE** register.

Returns:

None

15.2.2.4 GPIO_getInputPinValue

This function gets the input value on the selected pin.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	188
CCS 4.2.1	Size	102
CCS 4.2.1	Speed	104
IAR 5.51.6	None	142
IAR 5.51.6	Size	114
IAR 5.51.6	Speed	174
MSPGCC 4.8.0	None	240
MSPGCC 4.8.0	Size	158
MSPGCC 4.8.0	Speed	198

Prototype:

```
uint8_t
GPIO_getInputPinValue(uint8_t selectedPort,
                      uint16_t selectedPins)
```

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

15.2.2.5 uint16_t GPIO_getInterruptStatus (uint8_t *selectedPort*, uint16_t *selectedPins*)

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
CCS 4.2.1	None	184
CCS 4.2.1	Size	100
CCS 4.2.1	Speed	104
IAR 5.51.6	None	152
IAR 5.51.6	Size	118
IAR 5.51.6	Speed	172
MSPGCC 4.8.0	None	246
MSPGCC 4.8.0	Size	150
MSPGCC 4.8.0	Speed	178

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]

15.2.2.6 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.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	252
CCS 4.2.1	Size	132
CCS 4.2.1	Speed	132
IAR 5.51.6	None	184
IAR 5.51.6	Size	158
IAR 5.51.6	Speed	192
MSPGCC 4.8.0	None	422
MSPGCC 4.8.0	Size	186
MSPGCC 4.8.0	Speed	160

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**

Description:
Modified bits of **PxIES** register.

Returns:
None

15.2.2.7 GPIO_setAsInputPin

This function configures the selected Pin as input pin.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	300
CCS 4.2.1	Size	144
CCS 4.2.1	Speed	144
IAR 5.51.6	None	220
IAR 5.51.6	Size	164
IAR 5.51.6	Speed	250
MSPGCC 4.8.0	None	648
MSPGCC 4.8.0	Size	192
MSPGCC 4.8.0	Speed	160

Prototype:
```
void
GPIO_setAsInputPin(uint8_t selectedPort,
                   uint16_t selectedPins)
```

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**

Description:
Modified bits of **PxDIR** register, bits of **PxREN** register and bits of **PxSEL** register.

Returns:
None

15.2.2.8 GPIO_setAsInputPinWithPullDownresistor

This function sets the selected Pin in input Mode with Pull Down resistor.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	338
CCS 4.2.1	Size	162
CCS 4.2.1	Speed	162
IAR 5.51.6	None	246
IAR 5.51.6	Size	120
IAR 5.51.6	Speed	292
MSPGCC 4.8.0	None	744
MSPGCC 4.8.0	Size	212
MSPGCC 4.8.0	Speed	186

Prototype:
```
void
GPIO_setAsInputPinWithPullDownresistor(uint8_t selectedPort,
                                       uint16_t selectedPins)
```

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**

Description:
Modified bits of **PxDIR** register, bits of **PxOUT** register and bits of **PxREN** register.

Returns:
None

15.2.2.9 GPIO_setAsInputPinWithPullUpresistor

This function sets the selected Pin in input Mode with Pull Up resistor.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	338
CCS 4.2.1	Size	162
CCS 4.2.1	Speed	162
IAR 5.51.6	None	246
IAR 5.51.6	Size	120
IAR 5.51.6	Speed	292
MSPGCC 4.8.0	None	724
MSPGCC 4.8.0	Size	212
MSPGCC 4.8.0	Speed	186

Prototype:

```
void
GPIO_setAsInputPinWithPullUpresistor(uint8_t selectedPort,
                                     uint16_t selectedPins)
```

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**

Description:
Modified bits of **PxDIR** register, bits of **PxOUT** register and bits of **PxREN** register.

Returns:
None

15.2.2.10 GPIO_setAsOutputPin

This function configures the selected Pin as output pin.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	258
CCS 4.2.1	Size	138
CCS 4.2.1	Speed	138
IAR 5.51.6	None	190
IAR 5.51.6	Size	170
IAR 5.51.6	Speed	236
MSPGCC 4.8.0	None	504
MSPGCC 4.8.0	Size	188
MSPGCC 4.8.0	Speed	162

Prototype:

```
void
GPIO_setAsOutputPin(uint8_t selectedPort,
                    uint16_t selectedPins)
```

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**

Description:

Modified bits of **PxDIR** register and bits of **PxSEL** register.

Returns:

None

15.2.2.11 GPIO_setAsPeripheralModuleFunctionInputPin

This function configures the peripheral module function in the input direction for the selected pin for either primary, secondary or ternary module function modes.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	576
CCS 4.2.1	Size	264
CCS 4.2.1	Speed	270
IAR 5.51.6	None	424
IAR 5.51.6	Size	234
IAR 5.51.6	Speed	440
MSPGCC 4.8.0	None	1274
MSPGCC 4.8.0	Size	358
MSPGCC 4.8.0	Speed	330

Prototype:

```
void
GPIO_setAsPeripheralModuleFunctionInputPin(uint8_t selectedPort,
                                           uint16_t selectedPins,
                                           uint8_t mode)
```

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**

Description:
Modified bits of **PxDIR** register and bits of **PxSEL** register.

Returns:
None

15.2.2.12 GPIO_setAsPeripheralModuleFunctionOutputPin

This function configures the peripheral module function in the output direction for the selected pin for either primary, secondary or ternary module function modes.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	576
CCS 4.2.1	Size	216
CCS 4.2.1	Speed	252
IAR 5.51.6	None	424
IAR 5.51.6	Size	244
IAR 5.51.6	Speed	370
MSPGCC 4.8.0	None	1214
MSPGCC 4.8.0	Size	304
MSPGCC 4.8.0	Speed	312

Prototype:
```
void
GPIO_setAsPeripheralModuleFunctionOutputPin(uint8_t selectedPort,
                                            uint16_t selectedPins,
                                            uint8_t mode)
```

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**

Description:

Modified bits of **PxDIR** register and bits of **PxSEL** register.

Returns:

None

15.2.2.13 GPIO_setOutputHighOnPin

This function sets output HIGH on the selected Pin.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	174
CCS 4.2.1	Size	102
CCS 4.2.1	Speed	102
IAR 5.51.6	None	130
IAR 5.51.6	Size	114
IAR 5.51.6	Speed	154
MSPGCC 4.8.0	None	256
MSPGCC 4.8.0	Size	144
MSPGCC 4.8.0	Speed	120

Prototype:

```
void
GPIO_setOutputHighOnPin(uint8_t selectedPort,
                        uint16_t selectedPins)
```

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**

Description:
Modified bits of **PxOUT** register.

Returns:
None

15.2.2.14 GPIO_setOutputLowOnPin

This function sets output LOW on the selected Pin.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	174
CCS 4.2.1	Size	102
CCS 4.2.1	Speed	102
IAR 5.51.6	None	130
IAR 5.51.6	Size	116
IAR 5.51.6	Speed	160
MSPGCC 4.8.0	None	276
MSPGCC 4.8.0	Size	144
MSPGCC 4.8.0	Speed	120

Prototype:

```
void
GPIO_setOutputLowOnPin(uint8_t selectedPort,
                       uint16_t selectedPins)
```

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**

Description:
Modified bits of **PxOUT** register.

Returns:
None

15.2.2.15 GPIO_toggleOutputOnPin

This function toggles the output on the selected Pin.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	174
CCS 4.2.1	Size	102
CCS 4.2.1	Speed	102
IAR 5.51.6	None	130
IAR 5.51.6	Size	114
IAR 5.51.6	Speed	154
MSPGCC 4.8.0	None	256
MSPGCC 4.8.0	Size	156
MSPGCC 4.8.0	Speed	140

Prototype:

```
void
GPIO_toggleOutputOnPin(uint8_t selectedPort,
                       uint16_t selectedPins)
```

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**

Description:

Modified bits of **PxOUT** register.

Returns:

None

15.3 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
        );
}
```

16 Memory Protection Unit (MPU)

Introduction 177
API Functions 177
Programming Example 183

16.1 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 CCS 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 withing 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
CCS 4.2.1	None	932
CCS 4.2.1	Size	530
CCS 4.2.1	Speed	556
IAR 5.51.6	None	748
IAR 5.51.6	Size	440
IAR 5.51.6	Speed	496
MSPGCC 4.8.0	None	1566
MSPGCC 4.8.0	Size	534
MSPGCC 4.8.0	Speed	616

16.2 API Functions

Functions

- uint16_t MPU_clearAllInterruptFlags (uint32_t baseAddress)
- uint16_t MPU_clearInterruptFlag (uint32_t baseAddress, uint16_t memAccFlag)
- void MPU_createThreeSegments (uint32_t baseAddress, uint16_t seg1boundary, uint16_t seg2boundary, uint8_t seg1accmask, uint8_t seg2accmask, uint8_t seg3accmask)
- void MPU_createTwoSegments (uint32_t baseAddress, uint16_t seg1boundary, uint8_t seg1accmask, uint8_t seg2accmask)
- void MPU_disablePUCOnViolation (uint32_t baseAddress, uint16_t segment)
- void MPU_enablePUCOnViolation (uint32_t baseAddress, uint16_t segment)
- uint16_t MPU_getInterruptStatus (uint32_t baseAddress, uint16_t memAccFlag)
- void MPU_start (uint32_t baseAddress)

16.2.1 Detailed Description

The MPU API is broken into three group of functions: those that handle initialization, those that deal with memory segmentation and access rights definition, and those that handle interrupts.

The MPU initialization function is

- MPU_start()

The MPU memory segmentation and access right definition functions are

- MPU_createTwoSegments()
- MPU_createThreeSegments()

The MPU interrupt handler functions

- MPU_enablePUCOnViolation()
- MPU_getInterruptStatus()
- MPU_clearInterruptFlag()
- MPU_clearAllInterruptFlags()

16.2.2 Function Documentation

16.2.2.1 MPU_clearAllInterruptFlags

Clears all Memory Segment Access Violation Interrupt Flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	48
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	34
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

Prototype:

```
uint16_t
MPU_clearAllInterruptFlags(uint32_t baseAddress)
```

Description:

Modified registers are MPUCTL1

Parameters:

baseAddress is the base address of the MPU module.

Returns:

The current interrupt status as the mask of the set flags. Return 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

indicating the status of the interrupt flags.

16.2.2.2 MPU_clearInterruptFlag

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
CCS 4.2.1	None	56
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	30
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

Prototype:

```
uint16_t
MPU_clearInterruptFlag(uint32_t baseAddress,
                       uint16_t memAccFlag)
```

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

Returns:

The current interrupt status as the mask of the set flags. Return 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

indicating the status of the masked flags.

16.2.2.3 void MPU_createThreeSegments (uint32_t *baseAddress*, uint16_t *seg1boundary*, uint16_t *seg2boundary*, uint8_t *seg1accmask*, uint8_t *seg2accmask*, uint8_t *seg3accmask*)

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. For MPU device consult Table 6-1 in Section 6.2.2 of the User's Guide to find the options available for MPUSBx.

Modified registers are MPUCTL0 , MPUSEG , MPUSAM

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	392
CCS 4.2.1	Size	256
CCS 4.2.1	Speed	272
IAR 5.51.6	None	360
IAR 5.51.6	Size	222
IAR 5.51.6	Speed	224
MSPGCC 4.8.0	None	728
MSPGCC 4.8.0	Size	246
MSPGCC 4.8.0	Speed	292

Parameters:

baseAddress is the base address of the MPU module.

seg1boundary Valid values can be found in the Family User's Guide Valid values are:

- **MPU_Range** of 0x4000 - 0x13C0

seg2boundary Valid values can be found in the Family User's Guide Valid values are:

- **MPU_Range** of 0x4000 - 0x13C0

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

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

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

Returns:

None

16.2.2.4 void MPU_createTwoSegments (uint32_t *baseAddress*, uint16_t *seg1boundary*, uint8_t *seg1accmask*, uint8_t *seg2accmask*)

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. For MPU device consult Table 6-1 in Section 6.2.2 of the User's Guide to find the options available for MPUSBx.

Modified registers are MPUCTL0 , MPUSEG , MPUSAM

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	282
CCS 4.2.1	Size	170
CCS 4.2.1	Speed	180
IAR 5.51.6	None	248
IAR 5.51.6	Size	156
IAR 5.51.6	Speed	198
MSPGCC 4.8.0	None	510
MSPGCC 4.8.0	Size	174
MSPGCC 4.8.0	Speed	210

Parameters:

baseAddress is the base address of the MPU module.

seg1boundary Valid values can be found in the Family User's Guide Valid values are:

- **MPU_Range** of 0x4000 - 0x13C0

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

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

Returns:

None

16.2.2.5 void MPU_disablePUCOnViolation (uint32_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.

Modified registers are MPUCTL0 , MPUSAM

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	48
CCS 4.2.1	Size	16
CCS 4.2.1	Speed	16
IAR 5.51.6	None	26
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

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

Returns:

None

16.2.2.6 void MPU_enablePUCOnViolation (uint32_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.

Modified registers are MPUCTL0 , MPUSAM

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	48
CCS 4.2.1	Size	16
CCS 4.2.1	Speed	16
IAR 5.51.6	None	26
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

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

Returns:

None

16.2.2.7 uint16_t MPU_getInterruptStatus (uint32_t *baseAddress*, uint16_t *memAccFlag*)

Returns the memory segment violation flag status requested by the user.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	8
CCS 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	32
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 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

Returns:

The current interrupt status as the mask of the set flags. Return 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

indicating the status of the masked flags.

16.2.2.8 void MPU_start (uint32_t *baseAddress*)

The following function enables the MPU module in the device.

Modified register is MPUCTL0

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	12
CCS 4.2.1	Speed	12
IAR 5.51.6	None	16
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
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 MPU module.

Returns:
None

16.3 Programming Example

The following example shows some MPU operations using the APIs

```
//Define memory segment boundaries and set access right for each memory segment
MPU_createThreeSegments(MPU_BASE,0x04,0x08,
                                                    MPU_READ|MPU_WRITE|MPU_EXEC,
                                                    MPU_READ,
                                                    MPU_READ|MPU_WRITE|MPU_EXEC);

// 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);
```

17 32-Bit Hardware Multiplier (MPY32)

Introduction 184
API Functions 184
Programming Example 196

17.1 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 CCS 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 withing 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
CCS 4.2.1	None	390
CCS 4.2.1	Size	212
CCS 4.2.1	Speed	212
IAR 5.51.6	None	262
IAR 5.51.6	Size	188
IAR 5.51.6	Speed	212
MSPGCC 4.8.0	None	780
MSPGCC 4.8.0	Size	338
MSPGCC 4.8.0	Speed	338

17.2 API Functions

Functions

- uint16_t MPY32_getCarryBitValue (void)
- uint16_t MPY32_getResult16Bit (void)
- uint32_t MPY32_getResult24Bit (void)
- uint32_t MPY32_getResult32Bit (void)
- uint64 MPY32_getResult64Bit (void)
- uint8_t MPY32_getResult8Bit (void)
- uint16_t MPY32_getSumExtension (void)
- void MPY32_resetFractionMode (void)
- void MPY32_resetSaturationMode (void)
- void MPY32_setFractionMode (void)
- 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_setSaturationMode (void)
- void MPY32_setWriteDelay (uint16_t writeDelaySelect)

17.2.1 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_setSaturationMode()
- MPY32_resetSaturationMode()
- MPY32_setFractionMode()
- MPY32_resetFractionMode()

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_getResult8Bit()
- MPY32_getResult16Bit()
- MPY32_getResult24Bit()
- MPY32_getResult32Bit()
- MPY32_getResult64Bit()
- MPY32_getSumExtension()
- MPY32_getCarryBitValue()

17.2.2 Function Documentation

17.2.2.1 MPY32_getCarryBitValue

Returns the Carry Bitof the last multiplication operation.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	8
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
uint16_t
MPY32_getCarryBitValue(void)
```

Description:

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.

Returns:

The value of the MPY32 module Carry Bit 0x0 or 0x1.

17.2.2.2 MPY32_getResult16Bit

Returns an 16-bit result of the last multiplication operation.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
uint16_t
MPY32_getResult16Bit(void)
```

Description:

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.

Returns:

The 16-bit result of the last multiplication operation.

17.2.2.3 MPY32_getResult24Bit

Returns an 24-bit result of the last multiplication operation.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	14
CCS 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

Prototype:

```
uint32_t
MPY32_getResult24Bit(void)
```

Description:

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.

Returns:

The 24-bit result of the last multiplication operation.

17.2.2.4 MPY32_getResult32Bit

Returns an 32-bit result of the last multiplication operation.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	14
CCS 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

Prototype:

```
uint32_t
MPY32_getResult32Bit(void)
```

Description:

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.

Returns:

The 32-bit result of the last multiplication operation.

17.2.2.5 MPY32_getResult64Bit

Returns an 64-bit result of the last multiplication operation.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	42
CCS 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

Prototype:

```
uint64
MPY32_getResult64Bit(void)
```

Description:

This function returns all 64 bits of the result registers. The way this is passed is with 4 integers contained within a uint16 struct.

Returns:

The 64-bit result separated into 4 uint16_ts in a uint16 struct

17.2.2.6 MPY32_getResult8Bit

Returns an 8-bit result of the last multiplication operation.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	14
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
uint8_t
MPY32_getResult8Bit(void)
```

Description:

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.

Returns:

The 8-bit result of the last multiplication operation.

17.2.2.7 MPY32_getSumExtension

Returns the Sum Extension of the last multiplication operation.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
uint16_t
MPY32_getSumExtension(void)
```

Description:

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.

Returns:

The value of the MPY32 module Sum Extension.

17.2.2.8 MPY32_resetFractionMode

Disables Fraction Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
MPY32_resetFractionMode(void)
```

Description:

This function disables fraction mode.

Returns:

None

17.2.2.9 MPY32_resetSaturationMode

Disables Saturation Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
MPY32_resetSaturationMode(void)
```

Description:
This function disables saturation mode, which allows the raw result of the MPY result registers to be returned.

Returns:
None

17.2.2.10 MPY32_setFractionMode

Enables Fraction Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
MPY32_setFractionMode(void)
```

Description:
This function enables fraction mode.

Returns:
None

17.2.2.11 MPY32_setOperandOne16Bit

Sets an 16-bit value into operand 1.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
MPY32_setOperandOne16Bit(uint8_t multiplicationType,
                         uint16_t operand)
```

Description:
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.

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

17.2.2.12 MPY32_setOperandOne24Bit

Sets an 24-bit value into operand 1.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	46
CCS 4.2.1	Size	16
CCS 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	48
MSPGCC 4.8.0	Speed	48

Prototype:
```
void
MPY32_setOperandOne24Bit(uint8_t multiplicationType,
                         uint32_t operand)
```

Description:
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.

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

17.2.2.13 MPY32_setOperandOne32Bit

Sets an 32-bit value into operand 1.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	44
CCS 4.2.1	Size	16
CCS 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	44
MSPGCC 4.8.0	Speed	44

Prototype:

```
void
MPY32_setOperandOne32Bit(uint8_t multiplicationType,
                         uint32_t operand)
```

Description:

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.

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

17.2.2.14 MPY32_setOperandOne8Bit

Sets an 8-bit value into operand 1.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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	18
MSPGCC 4.8.0	Speed	18

Prototype:

```
void
MPY32_setOperandOne8Bit(uint8_t multiplicationType,
                        uint8_t operand)
```

Description:

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.

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

17.2.2.15 MPY32_setOperandTwo16Bit

Sets an 16-bit value into operand 2, which starts the multiplication.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	14
CCS 4.2.1	Size	6
CCS 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

Prototype:
```
void
MPY32_setOperandTwo16Bit(uint16_t operand)
```

Description:
This function sets the second operand of the multiplication operation and starts the operation.

Parameters:
operand is the 16-bit value to load into the 2nd operand.

Returns:
None

17.2.2.16 MPY32_setOperandTwo24Bit

Sets an 24-bit value into operand 2, which starts the multiplication.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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	22
MSPGCC 4.8.0	Speed	22

Prototype:

```
void
MPY32_setOperandTwo24Bit(uint32_t operand)
```

Description:
This function sets the second operand of the multiplication operation and starts the operation.

Parameters:
operand is the 24-bit value to load into the 2nd operand.

Returns:
None

17.2.2.17 MPY32_setOperandTwo32Bit

Sets an 32-bit value into operand 2, which starts the multiplication.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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

Prototype:

```
void
MPY32_setOperandTwo32Bit(uint32_t operand)
```

Description:
This function sets the second operand of the multiplication operation and starts the operation.

Parameters:
operand is the 32-bit value to load into the 2nd operand.

Returns:
None

17.2.2.18 MPY32_setOperandTwo8Bit

Sets an 8-bit value into operand 2, which starts the multiplication.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	14
CCS 4.2.1	Size	6
CCS 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	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
MPY32_setOperandTwo8Bit(uint8_t operand)
```

Description:

This function sets the second operand of the multiplication operation and starts the operation.

Parameters:

operand is the 8-bit value to load into the 2nd operand.

Returns:

None

17.2.2.19 MPY32_setSaturationMode

Enables Saturation Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
MPY32_setSaturationMode(void)
```

Description:

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.

Returns:

None

17.2.2.20 MPY32_setWriteDelay

Sets the write delay setting for the MPY32 module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	12
CCS 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

Prototype:

```
void
MPY32_setWriteDelay(uint16_t writeDelaySelect)
```

Description:

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 unpredicatble results if registers are changed before the results are ready.

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 **MPYDLYWRTEN** of **MPY32CTL0** register.

Returns:

None

17.3 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_BASE,
                         MPY32_MULTIPLY_UNSIGNED,
                         0x1234);
// Set Operand 2 to begin the multiplication operation
MPY32_setOperandTwo16Bit(MPY32_BASE,
                         0x5678);

__bis_SR_register(LPM4_bits);             // Enter LPM4
__no_operation();                         // BREAKPOINT HERE to verify the
                                           // correct result in the registers
```

18 Power Management Module (PMM)

Introduction 197
API Functions 197
Programming Example 204

18.1 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 CCS 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 withing 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
CCS 4.2.1	None	190
CCS 4.2.1	Size	174
CCS 4.2.1	Speed	178
IAR 5.51.6	None	174
IAR 5.51.6	Size	118
IAR 5.51.6	Speed	136
MSPGCC 4.8.0	None	482
MSPGCC 4.8.0	Size	210
MSPGCC 4.8.0	Speed	210

18.2 API Functions

Functions

- void PMM_clearInterrupt (uint16_t mask)
- void PMM_disableSVSH (void)
- void PMM_disableSVSL (void)
- void PMM_enableSVSH (void)
- void PMM_enableSVSL (void)
- uint16_t PMM_getInterruptStatus (uint16_t mask)
- void PMM_lockLPM5 (void)
- void PMM_regOff (void)
- void PMM_regOn (void)
- void PMM_trigBOR (void)
- void PMM_trigPOR (void)
- void PMM_unlockLPM5 (void)

18.2.1 Detailed Description

PMM_enableSVSH() / **PMM_disableSVSH()** If disabled on FR57xx, 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. If enabled, SVSH is always enabled. Note: this API has different functionality depending on the part.

PMM_enableSVSL() / **PMM_disableSVSL()** If disabled, Low-side SVS (SVSL) is disabled in low power modes. SVSL is always enabled in active mode and LPM0. If enabled, SVSL is enabled in LPM0/1/2. SVSL is always enabled in AM and always disabled in LPM3/4 and LPM3.5/4.5.

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.

18.2.2 Function Documentation

18.2.2.1 PMM_clearInterrupt

Clear all interrupt flags for the PMM.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	16
CCS 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

Prototype:

```
void
PMM_clearInterrupt(uint16_t mask)
```

Description:

Modified registers are PMMCTL0 , PMMIFG .

Parameters:

mask is the mask for specifying the required flag Mask value is the logical OR of any of the following:

- **PMM_PMMBORIFG**
- **PMM_PMMRSTIFG**
- **PMM_PMMPORIFG**
- **PMM_SVSLIFG**
- **PMM_SVSHIFG**
- **PMM_PMMLPM5IFG**
- **PMM_ALL**

Returns:

None

18.2.2.2 PMM_disableSVSH

Disables the high-side SVS circuitry.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	18
CCS 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	46
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Prototype:

```
void
PMM_disableSVSH(void)
```

Description:

Modified registers are PMMCTL0 .

Returns:

None

18.2.2.3 PMM_disableSVSL

Disables the low-side SVS circuitry.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	18
CCS 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	46
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Prototype:

```
void
PMM_disableSVSL(void)
```

Description:

Modified registers are PMMCTL0 .

Returns:

None

18.2.2.4 PMM_enableSVSH

Enables the high-side SVS circuitry.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	18
CCS 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	46
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Prototype:

```
void
PMM_enableSVSH(void)
```

Description:

Modified registers are PMMCTL0 .

Returns:

None

18.2.2.5 PMM_enableSVSL

Enables the low-side SVS circuitry.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	18
CCS 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	46
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Prototype:

```
void
PMM_enableSVSL(void)
```

Description:

Modified registers are PMMCTL0 , SVSMLCTL .

Returns:

None

18.2.2.6 PMM_getInterruptStatus

Returns interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	14
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
uint16_t
PMM_getInterruptStatus(uint16_t mask)
```

Parameters:

mask is the mask for specifying the required flag Mask value is the logical OR of any of the following:

- **PMM_PMMBORIFG**
- **PMM_PMMRSTIFG**
- **PMM_PMMPORIFG**
- **PMM_SVSLIFG**
- **PMM_SVSHIFG**
- **PMM_PMMLPM5IFG**
- **PMM_ALL**

Returns:

Logical OR of any of the following:

- **PMM_PMMBORIFG**
- **PMM_PMMRSTIFG**
- **PMM_PMMPORIFG**
- **PMM_SVSLIFG**
- **PMM_SVSHIFG**
- **PMM_PMMLPM5IFG**
- **PMM_ALL**

indicating the status of the selected interrupt flags

18.2.2.7 void PMM_lockLPM5 (void)

Lock I/O pin and other LPMx.5 relevant (e.g. RTC) configurations upon entry/exit to/from LPMx.5. Once power is applied to the device, this bit, once set, can only be cleared by the user or via another power cycle. LPMx.5 configuration remains locked. Pin state is held during LPMx.5 entry and exit.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:
None

18.2.2.8 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 registers are PMMCTL0 .

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	18
CCS 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	46
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Returns:
None

18.2.2.9 void PMM_regOn (void)

Makes the low-dropout voltage regulator (LDO) remain ON when going into LPM 3/4. [Default].

Modified registers are PMMCTL0 .

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	18
CCS 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	46
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Returns:
None

18.2.2.10 void PMM_trigBOR (void)

Calling this function will trigger a software Brown Out Rest (BOR).

Modified registers are PMMCTL0 .

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	16
CCS 4.2.1	Size	16
CCS 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	44
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Returns:
None

18.2.2.11 void PMM_trigPOR (void)

Calling this function will trigger a software Power On Reset (POR).

Modified registers are PMMCTL0 .

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	16
CCS 4.2.1	Size	16
CCS 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	44
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Returns:
None

18.2.2.12 void PMM_unlockLPM5 (void)

LPMx.5 configuration is not locked and defaults to its reset condition.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Returns:
None

18.3 Programming Example

The following example shows some pmm operations using the APIs

```
   //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_PMMPORIFG,
       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_PMMPORIFG,
       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 f
       {
               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 trigge
    }

        if (PMM_getInterruptStatus(PMM_BASE, PMM_PMMBORIFG))    // Was this reset triggered by the BOR fla
        {
                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 t
                                                                    //  in the PM5CTL0 register upon wake up
    }
//-------------------
    while (1)
    {
        __no_operation();             // Don't sleep
    }
```

19 Internal Reference (REF)

Introduction 206
API Functions 206
Programming Example 213

19.1 Introduction

The Internal Reference (REF) API provides a set of functions for using the MSP430Ware REF 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 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. These include but are not limited to the ADC12_B and COMP_B modules, dependent upon the particular 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 1.5 V, 2.0 V, and 2.5 V. In addition, when enabled, a buffered bandgap voltage is also available.

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

T

he following code metrics were performed with the CCS 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 withing 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
CCS 4.2.1	None	222
CCS 4.2.1	Size	72
CCS 4.2.1	Speed	72
IAR 5.51.6	None	88
IAR 5.51.6	Size	54
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	342
MSPGCC 4.8.0	Size	68
MSPGCC 4.8.0	Speed	68

19.2 API Functions

Functions

- void REF_disableReferenceVoltage (uint32_t baseAddress)
- void REF_disableTempSensor (uint32_t baseAddress)
- void REF_enableReferenceVoltage (uint32_t baseAddress)
- void REF_enableTempSensor (uint32_t baseAddress)
- uint16_t REF_getBandgapMode (uint32_t baseAddress)
- bool REF_isBandgapActive (uint32_t baseAddress)
- bool REF_isRefGenActive (uint32_t baseAddress)
- uint16_t REF_isRefGenBusy (uint32_t baseAddress)
- void REF_setReferenceVoltage (uint32_t baseAddress, uint8_t referenceVoltageSelect)

19.2.1 Detailed Description

The REF 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 module.

The reference voltage of the REF module is handled by

- REF_setReferenceVoltage()
- REF_enableReferenceVoltage()
- REF_disableReferenceVoltage()

The internal temperature sensor is handled by

- REF_disableTempSensor()
- REF_enableTempSensor()

The status of the REF module is handled by

- REF_getBandgapMode()
- REF_isBandgapActive()
- REF_isRefGenBusy()
- REF_isRefGen()

19.2.2 Function Documentation

19.2.2.1 REF_disableReferenceVoltage

Disables the reference voltage.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	36
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
REF_disableReferenceVoltage(uint32_t baseAddress)
```

Description:

This function is used to disable the generated reference voltage. Please note, if the

Parameters:

baseAddress is the base address of the REF module.

Modified bits are **REFON** of **REFCTL0** register.

Returns:

None

19.2.2.2 REF_disableTempSensor

Disables the internal temperature sensor to save power consumption.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	34
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
REF_disableTempSensor(uint32_t baseAddress)
```

Description:

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_isRefGenBusy() returns REF_BUSY, this function will have no effect.

Parameters:

baseAddress is the base address of the REF module.

Modified bits are **REFTCOFF** of **REFCTL0** register.

Returns:

None

19.2.2.3 REF_enableReferenceVoltage

Enables the reference voltage to be used by peripherals.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	34
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
REF_enableReferenceVoltage(uint32_t baseAddress)
```

Description:
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_isRefGenBusy() returns REF_BUSY, this function will have no effect.

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

Modified bits are **REFON** of **REFCTL0** register.

Returns:
None

19.2.2.4 REF_enableTempSensor

Enables the internal temperature sensor.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	36
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:
```
void
REF_enableTempSensor(uint32_t baseAddress)
```

Description:
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_isRefGenBusy() returns REF_BUSY, this function will have no effect.

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

Modified bits are **REFTCOFF** of **REFCTL0** register.

Returns:
None

19.2.2.5 REF_getBandgapMode

Returns the bandgap mode of the REF module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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	22
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint16_t
REF_getBandgapMode(uint32_t baseAddress)
```

Description:

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.

Parameters:

baseAddress is the base address of the REF module.

Returns:

One of the following:

- **REF_STATICMODE** if the bandgap is operating in static mode
- **REF_SAMPLEMODE** if the bandgap is operating in sample mode

bandgap mode of the REF module

19.2.2.6 REF_isBandgapActive

Returns the active status of the bandgap in the REF module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	8
CCS 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	42
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
bool
REF_isBandgapActive(uint32_t baseAddress)
```

Description:

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.

Parameters:

baseAddress is the base address of the REF module.

Returns:
One of the following:

- **REF_ACTIVE** if active
- **REF_INACTIVE** if not active

indicating the bandgap active status of the module

19.2.2.7 REF_isRefGenActive

Returns the active status of the reference generator in the REF module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	8
CCS 4.2.1	Speed	8
IAR 5.51.6	None	14
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	42
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
bool
REF_isRefGenActive(uint32_t baseAddress)
```

Description:
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.

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

Returns:
One of the following:

- **REF_ACTIVE** if active
- **REF_INACTIVE** if not active

indicating the reference generator active status of the module

19.2.2.8 REF_isRefGenBusy

Returns the busy status of the reference generator in the REF module.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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	22
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
uint16_t
REF_isRefGenBusy(uint32_t baseAddress)
```

Description:

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.

Parameters:

baseAddress is the base address of the REF module.

Returns:

One of the following:

- **REF_NOTBUSY** if the reference generator is not being used
- **REF_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

19.2.2.9 REF_setReferenceVoltage

Sets the reference voltage for the voltage generator.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	38
CCS 4.2.1	Size	12
CCS 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	74
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:

```
void
REF_setReferenceVoltage(uint32_t baseAddress,
                        uint8_t referenceVoltageSelect)
```

Description:

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_isRefGenBusy() returns REF_BUSY, this function will have no effect.

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_VREF1_5V** [Default]
- **REF_VREF2_0V**
- **REF_VREF2_5V**

  Modified bits are **REFVSEL** of **REFCTL0** register.

Returns:

None

19.3 Programming Example

The following example shows how to initialize and use the REF API with the ADC12 module to use as a positive reference to the analog signal input.

```
// By default, REFMSTR=1 => REFCTL is used to configure the internal reference

// If ref generator busy, WAIT
while(REF_refGenBusyStatus(REF_BASE));
// Select internal ref = 2.5V
REF_setReferenceVoltage(REF_BASE,
                        REF_VREF2_5V);
// Internal Reference ON
REF_enableReferenceVoltage(REF_BASE);

__delay_cycles(75);                      // Delay (~75us) for Ref to settle
```

20 Real-Time Clock (RTC_B)

Introduction 214
API Functions 214
Programming Example 227

20.1 Introduction

The Real Time Clock (RTC_B) API provides a set of functions for using the MSP430Ware RTC 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_A module is used, then Counter mode may also be initialized, as well as prescale counters.

The RTC module provides the ability to keep track of the current time and date in calendar mode, or can be setup as a 32-bit counter (RTC_B_A Only).

The RTC 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.

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 CCS 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 withing 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
CCS 4.2.1	None	1150
CCS 4.2.1	Size	534
CCS 4.2.1	Speed	532
IAR 5.51.6	None	814
IAR 5.51.6	Size	648
IAR 5.51.6	Speed	648
MSPGCC 4.8.0	None	1960
MSPGCC 4.8.0	Size	682
MSPGCC 4.8.0	Speed	688

20.2 API Functions

Functions

- void RTC_B_calendarInit (uint32_t baseAddress, Calendar CalendarTime, uint16_t formatSelect)
- void RTC_B_clearInterrupt (uint32_t baseAddress, uint8_t interruptFlagMask)
- uint16_t RTC_B_convertBCDToBinary (uint32_t baseAddress, uint16_t valueToConvert)
- uint16_t RTC_B_convertBinaryToBCD (uint32_t baseAddress, uint16_t valueToConvert)
- void RTC_B_definePrescaleEvent (uint32_t baseAddress, uint8_t prescaleSelect, uint8_t prescaleEventDivider)
- void RTC_B_disableInterrupt (uint32_t baseAddress, uint8_t interruptMask)
- void RTC_B_enableInterrupt (uint32_t baseAddress, uint8_t interruptMask)
- Calendar RTC_B_getCalendarTime (uint32_t baseAddress)
- uint8_t RTC_B_getInterruptStatus (uint32_t baseAddress, uint8_t interruptFlagMask)
- uint8_t RTC_B_getPrescaleValue (uint32_t baseAddress, uint8_t prescaleSelect)
- void RTC_B_holdClock (uint32_t baseAddress)
- void RTC_B_initCalendar (uint32_t baseAddress, Calendar *CalendarTime, uint16_t formatSelect)

- void RTC_B_setCalendarAlarm (uint32_t baseAddress, uint8_t minutesAlarm, uint8_t hoursAlarm, uint8_t dayOfWeekAlarm, uint8_t dayOfMonthAlarm)
- void RTC_B_setCalendarEvent (uint32_t baseAddress, uint16_t eventSelect)
- void RTC_B_setCalibrationData (uint32_t baseAddress, uint8_t offsetDirection, uint8_t offsetValue)
- void RTC_B_setCalibrationFrequency (uint32_t baseAddress, uint16_t frequencySelect)
- void RTC_B_setPrescaleCounterValue (uint32_t baseAddress, uint8_t prescaleSelect, uint8_t prescaleCounterValue)
- void RTC_B_startClock (uint32_t baseAddress)

20.2.1 Detailed Description

The RTC API is broken into 4 groups of functions: clock settings, calender mode, counter mode, and interrupt condition setup and enable functions.

The RTC clock settings are handled by

- RTC_B_startClock()
- RTC_B_holdClock()
- RTC_B_setCalibrationFrequency()
- RTC_B_setCalibrationData()

The RTC Calender Mode is initialized and setup by

- RTC_B_calendarInit()
- RTC_B_getPrescaleValue()

The RTC interrupts are handled by

- RTC_B_definePrescaleEvent()
- RTC_B_enableInterrupt()
- RTC_B_disableInterrupt()
- RTC_B_getInterruptStatus()
- RTC_B_clearInterrupt()

The RTC conversions are handled by

- RTC_B_convertBCDToBinary()
- RTC_B_convertBinaryToBCD()

20.2.2 Function Documentation

20.2.2.1 RTC_B_calendarInit

Deprecated - Initializes the settings to operate the RTC in calendar mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	128
CCS 4.2.1	Size	84
CCS 4.2.1	Speed	84
IAR 5.51.6	None	114
IAR 5.51.6	Size	102
IAR 5.51.6	Speed	102
MSPGCC 4.8.0	None	218
MSPGCC 4.8.0	Size	152
MSPGCC 4.8.0	Speed	152

Prototype:

```
void
RTC_B_calendarInit(uint32_t baseAddress,
                   Calendar CalendarTime,
                   uint16_t formatSelect)
```

Description:
This function initializes the Calendar mode of the RTC module.

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 **RTCBCD** of **RTCCTL1** register.

Returns:
None

20.2.2.2 RTC_B_clearInterrupt

Clears selected RTC interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	78
CCS 4.2.1	Size	34
CCS 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	166
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	36

Prototype:

```
void
RTC_B_clearInterrupt(uint32_t baseAddress,
                     uint8_t interruptFlagMask)
```

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

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

20.2.2.3 RTC_B_convertBCDToBinary

Returns the given BCD value in Binary Format.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	12
CCS 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	44
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:
```
uint16_t
RTC_B_convertBCDToBinary(uint32_t baseAddress,
                         uint16_t valueToConvert)
```

Description:
This function converts BCD values to Binary format.

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 valueToConvert parameter.

20.2.2.4 RTC_B_convertBinaryToBCD

Returns the given Binary value in BCD Format.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	12
CCS 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	44
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:
```
uint16_t
RTC_B_convertBinaryToBCD(uint32_t baseAddress,
                         uint16_t valueToConvert)
```

Description:
This function converts Binary values to BCD format.

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.

20.2.2.5 RTC_B_definePrescaleEvent

Sets up an interrupt condition for the selected Prescaler.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	54
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	18
IAR 5.51.6	None	32
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	108
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

Prototype:
```
void
RTC_B_definePrescaleEvent(uint32_t baseAddress,
                          uint8_t prescaleSelect,
                          uint8_t prescaleEventDivider)
```

Description:
This function sets the condition for an interrupt to assert based on the individual prescalers.

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

20.2.2.6 RTC_B_disableInterrupt

Disables selected RTC interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	74
CCS 4.2.1	Size	36
CCS 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	170
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	36

Prototype:

```
void
RTC_B_disableInterrupt(uint32_t baseAddress,
                       uint8_t interruptMask)
```

Description:

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_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

20.2.2.7 RTC_B_enableInterrupt

Enables selected RTC interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	74
CCS 4.2.1	Size	32
CCS 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	158
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	36

Prototype:

```
void
RTC_B_enableInterrupt(uint32_t baseAddress,
                      uint8_t interruptMask)
```

Description:

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_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

20.2.2.8 RTC_B_getCalendarTime

Returns the Calendar Time stored in the Calendar registers of the RTC.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	86
CCS 4.2.1	Size	68
CCS 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	176
MSPGCC 4.8.0	Size	58
MSPGCC 4.8.0	Speed	58

Prototype:

```
Calendar
RTC_B_getCalendarTime(uint32_t baseAddress)
```

Description:

This function returns the current Calendar time in the form of a Calendar structure.

Parameters:

baseAddress is the base address of the RTC_B module.

Returns:

A Calendar structure containing the current time.

20.2.2.9 RTC_B_getInterruptStatus

Returns the status of the selected interrupts flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	98
CCS 4.2.1	Size	40
CCS 4.2.1	Speed	40
IAR 5.51.6	None	60
IAR 5.51.6	Size	48
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	154
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	54

Prototype:

```
uint8_t
RTC_B_getInterruptStatus(uint32_t baseAddress,
                         uint8_t interruptFlagMask)
```

Description:

This function returns the status of the interrupt flag for the selected channel.

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

20.2.2.10 RTC_B_getPrescaleValue

Returns the selected prescaler value.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	52
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	36
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	28
MSPGCC 4.8.0	None	74
MSPGCC 4.8.0	Size	34
MSPGCC 4.8.0	Speed	32

Prototype:

```
uint8_t
RTC_B_getPrescaleValue(uint32_t baseAddress,
                       uint8_t prescaleSelect)
```

Description:

This function returns the value of the selected prescale counter register. The counter should be held before reading. If in counter mode, the individual prescaler can be held, while in Calendar mode the whole RTC must be held.

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

20.2.2.11 RTC_B_holdClock

Holds the RTC.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	8
CCS 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	40
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
void
RTC_B_holdClock(uint32_t baseAddress)
```

Description:

This function sets the RTC main hold bit to disable RTC functionality.

Parameters:

baseAddress is the base address of the RTC_B module.

Returns:

None

20.2.2.12 RTC_B_initCalendar

Initializes the settings to operate the RTC in calendar mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	128
CCS 4.2.1	Size	58
CCS 4.2.1	Speed	58
IAR 5.51.6	None	112
IAR 5.51.6	Size	104
IAR 5.51.6	Speed	104
MSPGCC 4.8.0	None	216
MSPGCC 4.8.0	Size	94
MSPGCC 4.8.0	Speed	94

Prototype:

```
void
RTC_B_initCalendar(uint32_t baseAddress,
                   Calendar *CalendarTime,
                   uint16_t formatSelect)
```

Description:

This function initializes the Calendar mode of the RTC module.

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

20.2.2.13 RTC_B_setCalendarAlarm

Sets and Enables the desired Calendar Alarm settings.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	82
CCS 4.2.1	Size	44
CCS 4.2.1	Speed	44
IAR 5.51.6	None	78
IAR 5.51.6	Size	64
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	120
MSPGCC 4.8.0	Size	56
MSPGCC 4.8.0	Speed	56

Prototype:

```
void
RTC_B_setCalendarAlarm(uint32_t baseAddress,
                       uint8_t minutesAlarm,
                       uint8_t hoursAlarm,
                       uint8_t dayOfWeekAlarm,
                       uint8_t dayOfMonthAlarm)
```

Description:

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.

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

20.2.2.14 RTC_B_setCalendarEvent

Sets a single specified Calendar interrupt condition.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	38
CCS 4.2.1	Size	12
CCS 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	56
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:

```
void
RTC_B_setCalendarEvent(uint32_t baseAddress,
                       uint16_t eventSelect)
```

Description:

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_B module.

eventSelect is the condition selected. Valid values are:

- **RTC_B_CALENDAREVENT_MINUTECHANGE** - assert interrupt on every minute
- **RTC_B_CALENDAREVENT_HOURCHANGE** - 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 **RTCTEV** of **RTCCTL** register.

Returns:
None

20.2.2.15 RTC_B_setCalibrationData

Sets the specified calibration for the RTC.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	40
CCS 4.2.1	Size	8
CCS 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	46
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
RTC_B_setCalibrationData(uint32_t baseAddress,
                         uint8_t offsetDirection,
                         uint8_t offsetValue)
```

Description:
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.

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

20.2.2.16 RTC_B_setCalibrationFrequency

Allows and Sets the frequency output to RTCCLK pin for calibration measurement.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	14
CCS 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	64
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
RTC_B_setCalibrationFrequency(uint32_t baseAddress,
                              uint16_t frequencySelect)
```

Description:

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_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

20.2.2.17 RTC_B_setPrescaleCounterValue

Sets the selected prescaler value.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	56
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	20
IAR 5.51.6	None	36
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	34
MSPGCC 4.8.0	Speed	38

Prototype:

```
void
RTC_B_setPrescaleCounterValue(uint32_t baseAddress,
                              uint8_t prescaleSelect,
                              uint8_t prescaleCounterValue)
```

Description:

This function sets the prescale counter value. Before setting the prescale counter, it should be held.

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

20.2.2.18 RTC_B_startClock

Starts the RTC.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	8
CCS 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	40
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:
```
void
RTC_B_startClock(uint32_t baseAddress)
```

Description:
This function clears the RTC main hold bit to allow the RTC to function.

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

Returns:
None

20.3 Programming Example

The following example shows how to initialize and use the RTC API to setup Calender Mode with the current time and various interrupts.

```
  //Initialize Calendar Mode of RTC
    /*
 Base Address of the RTC_B_A
 Pass in current time, initialized above
 Use BCD as Calendar Register Format
     */
    RTC_B_calendarInit(RTC_B_BASE,
        currentTime,
        RTC_B_FORMAT_BCD);

    //Setup Calendar Alarm for 5:00pm on the 5th day of the week.
```

```
//Note: Does not specify day of the week.
RTC_B_setCalendarAlarm(RTC_B_BASE,
    0x00,
    0x17,
    RTC_B_ALARMCONDITION_OFF,
    0x05);

//Specify an interrupt to assert every minute
RTC_B_setCalendarEvent(RTC_B_BASE,
    RTC_B_CALENDAREVENT_MINUTECHANGE);

//Enable interrupt for RTC Ready Status, which asserts when the RTC
//Calendar registers are ready to read.
//Also, enable interrupts for the Calendar alarm and Calendar event.
RTC_B_enableInterrupt(RTC_B_BASE,
    RTCRDYIE + RTCTEVIE + RTCAIE);

//Start RTC Clock
RTC_B_startClock(RTC_B_BASE);

//Enter LPM3 mode with interrupts enabled
__bis_SR_register(LPM3_bits + GIE);
__no_operation();
```

21 SFR Module

Introduction 229
API Functions 229
Programming Example 234

21.1 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 ~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 CCS 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 withing 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
CCS 4.2.1	None	114
CCS 4.2.1	Size	58
CCS 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	360
MSPGCC 4.8.0	Size	68
MSPGCC 4.8.0	Speed	68

21.2 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)

21.2.1 Detailed Description

The SFR API is broken into 2 groups: the SFR interrupts and the SFR ~RST/NMI pin control

The SFR interrupts are handled by

- SFR_enableInterrupt()
- SFR_disableInterrupt()

- SFR_getInterruptStatus()
- SFR_clearInterrupt()

The SFR ~RST/NMI pin is controlled by

- SFR_setResetPinPullResistor()
- SFR_setNMIEdge()
- SFR_setResetNMIPinFunction()

21.2.2 Function Documentation

21.2.2.1 SFR_clearInterrupt

Clears the selected SFR interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	14
CCS 4.2.1	Size	6
CCS 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	50
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
SFR_clearInterrupt(uint8_t interruptFlagMask)
```

Description:

This function clears the status of the selected SFR interrupt flags.

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

21.2.2.2 SFR_disableInterrupt

Disables selected SFR interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	14
CCS 4.2.1	Size	6
CCS 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	50
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
SFR_disableInterrupt(uint8_t interruptMask)
```

Description:

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.

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

21.2.2.3 SFR_enableInterrupt

Enables selected SFR interrupt sources.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	14
CCS 4.2.1	Size	6
CCS 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	36
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
SFR_enableInterrupt(uint8_t interruptMask)
```

Description:

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.

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

21.2.2.4 SFR_getInterruptStatus

Returns the status of the selected SFR interrupt flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	16
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:
```
uint8_t
SFR_getInterruptStatus(uint8_t interruptFlagMask)
```

Description:
This function returns the status of the selected SFR interrupt flags in a bit mask format matching that passed into the interruptFlagMask parameter.

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

21.2.2.5 SFR_setNMIEdge

Sets the edge direction that will assert an NMI from a signal on the ~RST/NMI pin if NMI function is active.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	10
CCS 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	66
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
SFR_setNMIEdge(uint16_t edgeDirection)
```

Description:

This function sets the edge direction that will assert an NMI from a signal on the ~RST/NMI pin if the NMI function is active. To activate the NMI function of the ~RST/NMI use the SFR_setResetNMIPinFunction() passing SFR_RESETPINFUNC_NMI into the resetPinFunction parameter.

Parameters:

edgeDirection is the direction that the signal on the ~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

21.2.2.6 SFR_setResetNMIPinFunction

Sets the function of the ~RST/NMI pin.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	10
CCS 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	60
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
SFR_setResetNMIPinFunction(uint8_t resetPinFunction)
```

Description:
This function sets the functionality of the ~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().

Parameters:
resetPinFunction is the function that the ~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

21.2.2.7 SFR_setResetPinPullResistor

Sets the pull-up/down resistor on the ~RST/NMI pin.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	12
CCS 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	68
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Prototype:
```
void
SFR_setResetPinPullResistor(uint16_t pullResistorSetup)
```

Description:
This function sets the pull-up/down resistors on the ~RST/NMI pin to the settings from the pullResistorSetup parameter.

Parameters:
pullResistorSetup is the selection of how the pull-up/down resistor on the ~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

21.3 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));
```

22 SYS Module

Introduction 236
API Functions 236
Programming Example 247

22.1 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 CCS 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 withing 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
CCS 4.2.1	None	268
CCS 4.2.1	Size	160
CCS 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	670
MSPGCC 4.8.0	Size	202
MSPGCC 4.8.0	Speed	202

22.2 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)

22.2.1 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()

22.2.2 Function Documentation

22.2.2.1 SYS_clearJTAGMailboxFlagStatus

Clears the status of the selected JTAG Mailbox flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	14
CCS 4.2.1	Size	6
CCS 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	50
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
SYS_clearJTAGMailboxFlagStatus(uint8_t mailboxFlagMask)
```

Description:
This function clears the selected JTAG Mailbox flags.

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

22.2.2.2 SYS_disableBSLMemory

Disables BSL memory.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	8
CCS 4.2.1	Size	8
CCS 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

Prototype:
```
void
SYS_disableBSLMemory(void)
```

Description:
This function disables BSL memory, which makes BSL memory act like vacant memory.

Returns:
None

22.2.2.3 SYS_disableBSLProtect

Disables BSL memory protection.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	8
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
SYS_disableBSLProtect(void)
```

Description:
This function disables protection on the BSL memory.

Returns:
None

22.2.2.4 SYS_disableRAMBasedInterruptVectors

Disables RAM-based Interrupt Vectors.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
SYS_disableRAMBasedInterruptVectors(void)
```

Description:
This function disables the interrupt vectors from being generated at the top of the RAM.

Returns:
None

22.2.2.5 SYS_enableBSLMemory

Enables BSL memory.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	8
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
void
SYS_enableBSLMemory(void)
```

Description:
This function enables BSL memory, which allows BSL memory to be addressed

Returns:
None

22.2.2.6 SYS_enableBSLProtect

Enables BSL memory protection.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	8
CCS 4.2.1	Size	8
CCS 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

Prototype:
```
void
SYS_enableBSLProtect(void)
```

Description:
This function enables protection on the BSL memory, which prevents any reading, programming, or erasing of the BSL memory.

Returns:
None

22.2.2.7 SYS_enableDedicatedJTAGPins

Sets the JTAG pins to be exclusively for JTAG until a BOR occurs.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	8
CCS 4.2.1	Size	8
CCS 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

Prototype:
```
void
SYS_enableDedicatedJTAGPins(void)
```

Description:
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.

Returns:
None

22.2.2.8 SYS_enablePMMAccessProtect

Enables PMM Access Protection.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:
```
void
SYS_enablePMMAccessProtect(void)
```

Description:
This function enables the PMM Access Protection, which will lock any changes on the PMM control registers until a BOR occurs.

Returns:
None

22.2.2.9 SYS_enableRAMBasedInterruptVectors

Enables RAM-based Interrupt Vectors.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	6
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:
```
void
SYS_enableRAMBasedInterruptVectors(void)
```

Description:
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.

Returns:
None

22.2.2.10 SYS_getBSLEntryIndication

Returns the indication of a BSL entry sequence from the Spy-Bi-Wire.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	16
CCS 4.2.1	Size	14
CCS 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	30
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint8_t
SYS_getBSLEntryIndication(void)
```

Description:

This function returns the indication of a BSL entry sequence from the Spy- Bi-Wire.

Returns:

One of the following:

- **SYS_BSLENTRY_INDICATED**
- **SYS_BSLENTRY_NOTINDICATED**

indicating if a BSL entry sequence was detected

22.2.2.11 SYS_getJTAGInboxMessage16Bit

Returns the contents of the selected JTAG Inbox in a 16 bit format.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	16
CCS 4.2.1	Size	8
CCS 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	12
MSPGCC 4.8.0	Speed	12

Prototype:

```
uint16_t
SYS_getJTAGInboxMessage16Bit(uint8_t inboxSelect)
```

Description:

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.

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.

22.2.2.12 SYS_getJTAGInboxMessage32Bit

Returns the contents of JTAG Inboxes in a 32 bit format.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	12
CCS 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

Prototype:
```
uint32_t
SYS_getJTAGInboxMessage32Bit(void)
```

Description:
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.

Returns:
The contents of both JTAG messages in a 32 bit format.

22.2.2.13 SYS_getJTAGMailboxFlagStatus

Returns the status of the selected JTAG Mailbox flags.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	16
CCS 4.2.1	Size	8
CCS 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	30
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:
```
uint8_t
SYS_getJTAGMailboxFlagStatus(uint8_t mailboxFlagMask)
```

Description:
This function will return the status of the selected JTAG Mailbox flags in bit mask format matching that passed into the mailboxFlagMask parameter.

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.

22.2.2.14 SYS_JTAGMailboxInit

Initializes JTAG Mailbox with selected properties.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	14
CCS 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	82
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

Prototype:
```
void
SYS_JTAGMailboxInit(uint8_t mailboxSizeSelect,
                    uint8_t autoClearInboxFlagSelect)
```

Description:
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.

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

22.2.2.15 SYS_setBSLSize

Sets the size of the BSL in Flash.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	12
CCS 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	62
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

Prototype:
```
void
SYS_setBSLSize(uint8_t BSLSizeSelect)
```

Description:
This function sets the size of the BSL in Flash memory.

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

22.2.2.16 SYS_setJTAGOutgoingMessage16Bit

Sets a 16 bit outgoing message in to the selected JTAG Outbox.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	8
CCS 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	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
SYS_setJTAGOutgoingMessage16Bit(uint8_t outboxSelect,
                                uint16_t outgoingMessage)
```

Description:

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.

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

22.2.2.17 SYS_setJTAGOutgoingMessage32Bit

Sets a 32 bit message in to both JTAG Outboxes.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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

Prototype:

```
void
SYS_setJTAGOutgoingMessage32Bit(uint32_t outgoingMessage)
```

Description:

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.

Parameters:

outgoingMessage is the message to send to the JTAG.
Modified bits are **MSGHI** and **MSGLO** of **SYSJMBOx** register.

Returns:

None

22.2.2.18 SYS_setRAMAssignedToBSL

Sets RAM assignment to BSL area.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	18
CCS 4.2.1	Size	10
CCS 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	60
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
SYS_setRAMAssignedToBSL(uint8_t BSLRAMAssignment)
```

Description:

This function allows RAM to be assigned to BSL, based on the selection of the BSLRAMAssignment parameter.

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

22.3 Programming Example

The following example shows how to initialize and use the SYS API

```
SYS_enableBSLProtect(SYS_BASE);
```

23 TIMER_A

Introduction 248
API Functions 249
Programming Example 273

23.1 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 CCS 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 withing 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
CCS 4.2.1	None	1356
CCS 4.2.1	Size	718
CCS 4.2.1	Speed	714
IAR 5.51.6	None	1008
IAR 5.51.6	Size	418
IAR 5.51.6	Speed	480
MSPGCC 4.8.0	None	1982
MSPGCC 4.8.0	Size	748
MSPGCC 4.8.0	Speed	3856

23.2 API Functions

Functions

- void TIMER_A_clear (uint32_t baseAddress)
- void TIMER_A_clearCaptureCompareInterruptFlag (uint32_t baseAddress, uint16_t captureCompareRegister)
- void TIMER_A_clearTimerInterruptFlag (uint32_t baseAddress)
- void TIMER_A_configureContinuousMode (uint32_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TAIE, uint16_t timerClear)
- void TIMER_A_configureUpDownMode (uint32_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 (uint32_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 (uint32_t baseAddress, uint16_t captureCompareRegister)
- void TIMER_A_disableInterrupt (uint32_t baseAddress)
- void TIMER_A_enableCaptureCompareInterrupt (uint32_t baseAddress, uint16_t captureCompareRegister)
- void TIMER_A_enableInterrupt (uint32_t baseAddress)
- void TIMER_A_generatePWM (uint32_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 (uint32_t baseAddress, uint16_t captureCompareRegister)
- uint32_t TIMER_A_getCaptureCompareInterruptStatus (uint32_t baseAddress, uint16_t captureCompareRegister, uint16_t mask)
- uint16_t TIMER_A_getCounterValue (uint32_t baseAddress)
- uint32_t TIMER_A_getInterruptStatus (uint32_t baseAddress)
- uint8_t TIMER_A_getOutputForOutputModeOutBitValue (uint32_t baseAddress, uint16_t captureCompareRegister)
- uint8_t TIMER_A_getSynchronizedCaptureCompareInput (uint32_t baseAddress, uint16_t captureCompareRegister, uint16_t synchronized)
- void TIMER_A_initCapture (uint32_t baseAddress, uint16_t captureRegister, uint16_t captureMode, uint16_t captureInputSelect, uint16_t synchronizeCaptureSource, uint16_t captureInterruptEnable, uint16_t captureOutputMode)
- void TIMER_A_initCompare (uint32_t baseAddress, uint16_t compareRegister, uint16_t compareInterruptEnable, uint16_t compareOutputMode, uint16_t compareValue)
- void TIMER_A_setCompareValue (uint32_t baseAddress, uint16_t compareRegister, uint16_t compareValue)
- void TIMER_A_setOutputForOutputModeOutBitValue (uint32_t baseAddress, uint16_t captureCompareRegister, uint8_t outputModeOutBitValue)
- void TIMER_A_startContinousMode (uint32_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TAIE, uint16_t timerClear)
- void TIMER_A_startContinuousMode (uint32_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TAIE, uint16_t timerClear)
- void TIMER_A_startCounter (uint32_t baseAddress, uint16_t timerMode)
- void TIMER_A_startUpDownMode (uint32_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 (uint32_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 (uint32_t baseAddress)

23.2.1 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()

23.2.2 Function Documentation

23.2.2.1 TIMER_A_clear

Reset/Clear the timer clock divider, count direction, count.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
TIMER_A_clear(uint32_t baseAddress)
```

Parameters:

baseAddress is the base address of the TIMER_A module.

Description:

Modified bits of **TAxCTL** register.

Returns:

None

23.2.2.2 TIMER_A_clearCaptureCompareInterruptFlag

Clears the capture-compare interrupt flag.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	8
CCS 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	42
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
void
TIMER_A_clearCaptureCompareInterruptFlag(uint32_t baseAddress,
                                         uint16_t captureCompareRegister)
```

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**

Description:
Modified bits are **CCIFG** of **TAxCCTLn** register.

Returns:
None

23.2.2.3 TIMER_A_clearTimerInterruptFlag

Clears the Timer TAIFG interrupt flag.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:
```
void
TIMER_A_clearTimerInterruptFlag(uint32_t baseAddress)
```

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

Description:
Modified bits are **TAIFG** of **TAxCTL** register.

Returns:
None

23.2.2.4 TIMER_A_configureContinuousMode

Configures TIMER_A in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	56
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	36
IAR 5.51.6	None	48
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	82
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	436

Prototype:
```
void
TIMER_A_configureContinuousMode(uint32_t baseAddress,
                                uint16_t clockSource,
```

```
                                        uint16_t clockSourceDivider,
                                        uint16_t timerInterruptEnable_TAIE,
                                        uint16_t timerClear)
```

Description:
This API does not start the timer. Timer needs to be started when required using the TIMER_A_startCounter API.

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_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **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

23.2.2.5 TIMER_A_configureUpDownMode

Configures TIMER_A in up down mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	94
CCS 4.2.1	Size	68
CCS 4.2.1	Speed	68
IAR 5.51.6	None	94
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	152
MSPGCC 4.8.0	Size	74
MSPGCC 4.8.0	Speed	384

Prototype:

```
void
TIMER_A_configureUpDownMode(uint32_t baseAddress,
                            uint16_t clockSource,
                            uint16_t clockSourceDivider,
                            uint16_t timerPeriod,
                            uint16_t timerInterruptEnable_TAIE,
                            uint16_t captureCompareInterruptEnable_CCR0_CCIE,
                            uint16_t timerClear)
```

Description:

This API does not start the timer. Timer needs to be started when required using the TIMER_A_startCounter API.

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_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **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 captureComapre 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

23.2.2.6 TIMER_A_configureUpMode

Configures TIMER_A in up mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	94
CCS 4.2.1	Size	68
CCS 4.2.1	Speed	68
IAR 5.51.6	None	94
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	152
MSPGCC 4.8.0	Size	74
MSPGCC 4.8.0	Speed	384

Prototype:

```
void
TIMER_A_configureUpMode(uint32_t baseAddress,
                        uint16_t clockSource,
                        uint16_t clockSourceDivider,
                        uint16_t timerPeriod,
                        uint16_t timerInterruptEnable_TAIE,
                        uint16_t captureCompareInterruptEnable_CCR0_CCIE,
                        uint16_t timerClear)
```

Description:
This API does not start the timer. Timer needs to be started when required using the TIMER_A_startCounter API.

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_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**

- **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 captureComapre 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

23.2.2.7 TIMER_A_disableCaptureCompareInterrupt

Disable capture compare interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	10
CCS 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	44
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
TIMER_A_disableCaptureCompareInterrupt(uint32_t baseAddress,
                                       uint16_t captureCompareRegister)
```

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**

Description:
Modified bits of **TAxCCTLn** register.

Returns:
None

23.2.2.8 TIMER_A_disableInterrupt

Disable timer interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:
```
void
TIMER_A_disableInterrupt(uint32_t baseAddress)
```

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

Description:
Modified bits of **TAxCTL** register.

Returns:
None

23.2.2.9 TIMER_A_enableCaptureCompareInterrupt

Enable capture compare interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	10
CCS 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	44
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
TIMER_A_enableCaptureCompareInterrupt(uint32_t baseAddress,
                                      uint16_t captureCompareRegister)
```

Description:
Does not clear interrupt flags

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

23.2.2.10 TIMER_A_enableInterrupt

Enable timer interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
TIMER_A_enableInterrupt(uint32_t baseAddress)
```

Description:
Does not clear interrupt flags

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

Modified bits of **TAxCTL** register.

Returns:
None

23.2.2.11 TIMER_A_generatePWM

Generate a PWM with timer running in up mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	94
CCS 4.2.1	Size	64
CCS 4.2.1	Speed	64
IAR 5.51.6	None	96
IAR 5.51.6	Size	74
IAR 5.51.6	Speed	80
MSPGCC 4.8.0	None	158
MSPGCC 4.8.0	Size	68
MSPGCC 4.8.0	Speed	660

Prototype:

```
void
TIMER_A_generatePWM(uint32_t baseAddress,
                    uint16_t clockSource,
                    uint16_t clockSourceDivider,
                    uint16_t timerPeriod,
                    uint16_t compareRegister,
                    uint16_t compareOutputMode,
                    uint16_t dutyCycle)
```

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_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **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

Description:
Modified bits of **TAxCTL** register, bits of **TAxCCTL0** register, bits of **TAxCCR0** register and bits of **TAxCCTLn** register.

Returns:
None

23.2.2.12 TIMER_A_getCaptureCompareCount

Get current capturecompare count.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	8
CCS 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	34
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint16_t
TIMER_A_getCaptureCompareCount(uint32_t baseAddress,
                               uint16_t captureCompareRegister)
```

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

23.2.2.13 uint32_t TIMER_A_getCaptureCompareInterruptStatus (uint32_t *baseAddress*, uint16_t *captureCompareRegister*, uint16_t *mask*)

Return capture compare interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	10
CCS 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

23.2.2.14 uint16_t TIMER_A_getCounterValue (uint32_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
CCS 4.2.1	None	98
CCS 4.2.1	Size	38
CCS 4.2.1	Speed	38
IAR 5.51.6	None	60
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	110
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

23.2.2.15 uint32_t TIMER_A_getInterruptStatus (uint32_t *baseAddress*)

Get timer interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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	38
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

23.2.2.16 uint8_t TIMER_A_getOutputForOutputModeOutBitValue (uint32_t *baseAddress*, uint16_t *captureCompareRegister*)

Get output bit for output mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	16
CCS 4.2.1	Speed	16
IAR 5.51.6	None	26
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	16
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 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**

23.2.2.17 uint8_t TIMER_A_getSynchronizedCaptureCompareInput (uint32_t *baseAddress*, uint16_t *captureCompareRegister*, uint16_t *synchronized*)

Get synchronized capturecompare input.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	14
CCS 4.2.1	Speed	14
IAR 5.51.6	None	26
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**

23.2.2.18 void TIMER_A_initCapture (uint32_t *baseAddress*, uint16_t *captureRegister*, uint16_t *captureMode*, uint16_t *captureInputSelect*, uint16_t *synchronizeCaptureSource*, uint16_t *captureInterruptEnable*, uint16_t *captureOutputMode*)

Initializes Capture Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	78
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	48
IAR 5.51.6	None	62
IAR 5.51.6	Size	48
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	116
MSPGCC 4.8.0	Size	38
MSPGCC 4.8.0	Speed	38

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_CCIxA**
- **TIMER_A_CAPTURE_INPUTSELECT_CCIxB**
- **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 captureComapre 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**

Description:

Modified bits of **TAxCCTLn** register.

Returns:

None

23.2.2.19 TIMER_A_initCompare

Initializes Compare Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	78
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	34
IAR 5.51.6	None	60
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	122
MSPGCC 4.8.0	Size	34
MSPGCC 4.8.0	Speed	34

Prototype:

```
void
TIMER_A_initCompare(uint32_t baseAddress,
                    uint16_t compareRegister,
                    uint16_t compareInterruptEnable,
                    uint16_t compareOutputMode,
                    uint16_t compareValue)
```

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 captureComapre 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

Description:

Modified bits of **TAxCCRn** register and bits of **TAxCCTLn** register.

Returns:

None

23.2.2.20 TIMER_A_setCompareValue

Sets the value of the capture-compare register.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	8
CCS 4.2.1	Speed	8
IAR 5.51.6	None	18
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:

```
void
TIMER_A_setCompareValue(uint32_t baseAddress,
                        uint16_t compareRegister,
                        uint16_t compareValue)
```

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

Description:

Modified bits of **TAxCCRn** register.

Returns:

None

23.2.2.21 TIMER_A_setOutputForOutputModeOutBitValue

Set output bit for output mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	14
CCS 4.2.1	Speed	14
IAR 5.51.6	None	24
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

Prototype:

```
void
TIMER_A_setOutputForOutputModeOutBitValue(uint32_t baseAddress,
                                          uint16_t captureCompareRegister,
                                          uint8_t outputModeOutBitValue)
```

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**

Description:

Modified bits of **TAxCCTLn** register.

Returns:

None

23.2.2.22 TIMER_A_startContinousMode

DEPRECATED - Spelling Error Fixed. Starts timer in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	58
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	40
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	464

Prototype:

```
void
TIMER_A_startContinousMode(uint32_t baseAddress,
                           uint16_t clockSource,
                           uint16_t clockSourceDivider,
                           uint16_t timerInterruptEnable_TAIE,
                           uint16_t timerClear)
```

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_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **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]

Description:

Modified bits of **TAxCTL** register.

Returns:

None

23.2.2.23 TIMER_A_startContinuousMode

Starts timer in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	60
CCS 4.2.1	Size	40
CCS 4.2.1	Speed	40
IAR 5.51.6	None	52
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	86
MSPGCC 4.8.0	Size	40
MSPGCC 4.8.0	Speed	464

Prototype:

```
void
TIMER_A_startContinuousMode(uint32_t baseAddress,
                            uint16_t clockSource,
                            uint16_t clockSourceDivider,
                            uint16_t timerInterruptEnable_TAIE,
                            uint16_t timerClear)
```

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_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **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]

Description:

Modified bits of **TAxCTL** register.

Returns:

None

23.2.2.24 TIMER_A_startCounter

Starts TIMER_A counter.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	6
CCS 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	38
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
TIMER_A_startCounter(uint32_t baseAddress,
                     uint16_t timerMode)
```

Description:

This function assumes that the timer has been previously configured using TIMER_A_configureContinuousMode, TIMER_A_configureUpMode or TIMER_A_configureUpDownMode.

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

23.2.2.25 TIMER_A_startUpDownMode

DEPRECATED - Replaced by TIMER_A_configureUpMode and TIMER_A_startCounter API. Starts timer in up down mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	98
CCS 4.2.1	Size	72
CCS 4.2.1	Speed	72
IAR 5.51.6	None	98
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	44
MSPGCC 4.8.0	None	156
MSPGCC 4.8.0	Size	78
MSPGCC 4.8.0	Speed	392

Prototype:

```
void
TIMER_A_startUpDownMode(uint32_t baseAddress,
                        uint16_t clockSource,
                        uint16_t clockSourceDivider,
                        uint16_t timerPeriod,
                        uint16_t timerInterruptEnable_TAIE,
                        uint16_t captureCompareInterruptEnable_CCR0_CCIE,
                        uint16_t timerClear)
```

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_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **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 captureComapre 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]

Description:

Modified bits of **TAxCTL** register, bits of **TAxCCTL0** register and bits of **TAxCCR0** register.

Returns:

None

23.2.2.26 TIMER_A_startUpMode

DEPRECATED - Replaced by TIMER_A_configureUpMode and TIMER_A_startCounter API. Starts timer in up mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	98
CCS 4.2.1	Size	72
CCS 4.2.1	Speed	72
IAR 5.51.6	None	98
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	44
MSPGCC 4.8.0	None	156
MSPGCC 4.8.0	Size	78
MSPGCC 4.8.0	Speed	392

Prototype:

```
void
TIMER_A_startUpMode(uint32_t baseAddress,
                    uint16_t clockSource,
                    uint16_t clockSourceDivider,
                    uint16_t timerPeriod,
                    uint16_t timerInterruptEnable_TAIE,
                    uint16_t captureCompareInterruptEnable_CCR0_CCIE,
                    uint16_t timerClear)
```

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_4**
- **TIMER_A_CLOCKSOURCE_DIVIDER_8**
- **TIMER_A_CLOCKSOURCE_DIVIDER_3**
- **TIMER_A_CLOCKSOURCE_DIVIDER_5**
- **TIMER_A_CLOCKSOURCE_DIVIDER_6**
- **TIMER_A_CLOCKSOURCE_DIVIDER_7**
- **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 captureComapre 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]

Description:

Modified bits of **TAxCTL** register, bits of **TAxCCTL0** register and bits of **TAxCCR0** register.

Returns:

None

23.2.2.27 TIMER_A_stop

Stops the timer.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	12
CCS 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	38
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
TIMER_A_stop(uint32_t baseAddress)
```

Parameters:

baseAddress is the base address of the TIMER_A module.

Description:

Modified bits of **TAxCTL** register.

Returns:

None

23.3 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();
    }
```

24 TIMER_B

Introduction 275
API Functions 276
Programming Example 303

24.1 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 CCS 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 withing 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
CCS 4.2.1	None	1458
CCS 4.2.1	Size	756
CCS 4.2.1	Speed	752
IAR 5.51.6	None	1052
IAR 5.51.6	Size	442
IAR 5.51.6	Speed	498
MSPGCC 4.8.0	None	2150
MSPGCC 4.8.0	Size	786
MSPGCC 4.8.0	Speed	3894

24.2 API Functions

Functions

- void TIMER_B_clear (uint32_t baseAddress)
- void TIMER_B_clearCaptureCompareInterruptFlag (uint32_t baseAddress, uint16_t captureCompareRegister)
- void TIMER_B_clearTimerInterruptFlag (uint32_t baseAddress)
- void TIMER_B_configureContinuousMode (uint32_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TBIE, uint16_t timerClear)
- void TIMER_B_configureUpDownMode (uint32_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 (uint32_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 (uint32_t baseAddress, uint16_t captureCompareRegister)
- void TIMER_B_disableInterrupt (uint32_t baseAddress)
- void TIMER_B_enableCaptureCompareInterrupt (uint32_t baseAddress, uint16_t captureCompareRegister)
- void TIMER_B_enableInterrupt (uint32_t baseAddress)
- void TIMER_B_generatePWM (uint32_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 (uint32_t baseAddress, uint16_t captureCompareRegister)
- uint32_t TIMER_B_getCaptureCompareInterruptStatus (uint32_t baseAddress, uint16_t captureCompareRegister, uint16_t mask)
- uint16_t TIMER_B_getCounterValue (uint32_t baseAddress)
- uint32_t TIMER_B_getInterruptStatus (uint32_t baseAddress)
- uint8_t TIMER_B_getOutputForOutputModeOutBitValue (uint32_t baseAddress, uint16_t captureCompareRegister)
- uint8_t TIMER_B_getSynchronizedCaptureCompareInput (uint32_t baseAddress, uint16_t captureCompareRegister, uint16_t synchronized)
- void TIMER_B_initCapture (uint32_t baseAddress, uint16_t captureRegister, uint16_t captureMode, uint16_t captureInputSelect, uint16_t synchronizeCaptureSource, uint16_t captureInterruptEnable, uint16_t captureOutputMode)
- void TIMER_B_initCompare (uint32_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_selectCounterLength (uint16_t baseAddress, uint16_t counterLength)
- void TIMER_B_selectLatchingGroup (uint16_t baseAddress, uint16_t groupLatch)
- void TIMER_B_setCompareValue (uint32_t baseAddress, uint16_t compareRegister, uint16_t compareValue)
- void TIMER_B_setOutputForOutputModeOutBitValue (uint32_t baseAddress, uint16_t captureCompareRegister, uint8_t outputModeOutBitValue)
- void TIMER_B_startContinousMode (uint32_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TBIE, uint16_t timerClear)
- void TIMER_B_startContinuousMode (uint32_t baseAddress, uint16_t clockSource, uint16_t clockSourceDivider, uint16_t timerInterruptEnable_TBIE, uint16_t timerClear)

- void TIMER_B_startCounter (uint32_t baseAddress, uint16_t timerMode)
- void TIMER_B_startUpDownMode (uint32_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 (uint32_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 (uint32_t baseAddress)

24.2.1 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()

24.2.2 Function Documentation

24.2.2.1 TIMER_B_clear

Reset/Clear the TIMER_B clock divider, count direction, count.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
TIMER_B_clear(uint32_t baseAddress)
```

Parameters:

baseAddress is the base address of the TIMER_B module.

Description:

Modified bits of **TBxCTL** register.

Returns:

None

24.2.2.2 TIMER_B_clearCaptureCompareInterruptFlag

Clears the capture-compare interrupt flag.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	8
CCS 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	42
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

Prototype:

```
void
TIMER_B_clearCaptureCompareInterruptFlag(uint32_t baseAddress,
                                         uint16_t captureCompareRegister)
```

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**

Description:
Modified bits are **CCIFG** of **TBxCCTLn** register.

Returns:
None

24.2.2.3 TIMER_B_clearTimerInterruptFlag

Clears the TIMER_B TBIFG interrupt flag.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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	26
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:
```
void
TIMER_B_clearTimerInterruptFlag(uint32_t baseAddress)
```

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

Description:
Modified bits are **TBIFG** of **TBxCTL** register.

Returns:
None

24.2.2.4 TIMER_B_configureContinuousMode

Configures TIMER_B in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	56
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	36
IAR 5.51.6	None	48
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	82
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	436

Prototype:

```
void
TIMER_B_configureContinuousMode(uint32_t baseAddress,
                                uint16_t clockSource,
                                uint16_t clockSourceDivider,
                                uint16_t timerInterruptEnable_TBIE,
                                uint16_t timerClear)
```

Description:

This API does not start the timer. Timer needs to be started when required using the TIMER_B_startCounter API.

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_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **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

24.2.2.5 TIMER_B_configureUpDownMode

Configures TIMER_B in up down mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	94
CCS 4.2.1	Size	68
CCS 4.2.1	Speed	68
IAR 5.51.6	None	94
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	152
MSPGCC 4.8.0	Size	74
MSPGCC 4.8.0	Speed	384

Prototype:

```
void
TIMER_B_configureUpDownMode(uint32_t baseAddress,
                            uint16_t clockSource,
                            uint16_t clockSourceDivider,
                            uint16_t timerPeriod,
                            uint16_t timerInterruptEnable_TBIE,
                            uint16_t captureCompareInterruptEnable_CCR0_CCIE,
                            uint16_t timerClear)
```

Description:
This API does not start the timer. Timer needs to be started when required using the TIMER_B_startCounter API.

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_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **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

24.2.2.6 TIMER_B_configureUpMode

Configures TIMER_B in up mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	94
CCS 4.2.1	Size	68
CCS 4.2.1	Speed	68
IAR 5.51.6	None	94
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	152
MSPGCC 4.8.0	Size	74
MSPGCC 4.8.0	Speed	384

Prototype:

```
void
TIMER_B_configureUpMode(uint32_t baseAddress,
                        uint16_t clockSource,
                        uint16_t clockSourceDivider,
                        uint16_t timerPeriod,
                        uint16_t timerInterruptEnable_TBIE,
                        uint16_t captureCompareInterruptEnable_CCR0_CCIE,
                        uint16_t timerClear)
```

Description:
This API does not start the timer. Timer needs to be started when required using the TIMER_B_startCounter API.

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_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **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

24.2.2.7 TIMER_B_disableCaptureCompareInterrupt

Disable capture compare interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	10
CCS 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	44
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
TIMER_B_disableCaptureCompareInterrupt(uint32_t baseAddress,
                                       uint16_t captureCompareRegister)
```

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**

Description:

Modified bits of **TBxCCTLn** register.

Returns:

None

24.2.2.8 TIMER_B_disableInterrupt

Disable TIMER_B interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
TIMER_B_disableInterrupt(uint32_t baseAddress)
```

Parameters:

baseAddress is the base address of the TIMER_B module.

Description:

Modified bits of **TBxCTL** register.

Returns:

None

24.2.2.9 TIMER_B_enableCaptureCompareInterrupt

Enable capture compare interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	10
CCS 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	44
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
void
TIMER_B_enableCaptureCompareInterrupt(uint32_t baseAddress,
                                      uint16_t captureCompareRegister)
```

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**

Description:

Modified bits of **TBxCCTLn** register.

Returns:

None

24.2.2.10 TIMER_B_enableInterrupt

Enable TIMER_B interrupt.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	20
CCS 4.2.1	Size	6
CCS 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

Prototype:

```
void
TIMER_B_enableInterrupt(uint32_t baseAddress)
```

Description:

Enables TIMER_B interrupt. Does not clear interrupt flags.

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

Modified bits of **TBxCTL** register.

Returns:
None

24.2.2.11 TIMER_B_generatePWM

Generate a PWM with TIMER_B running in up mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	94
CCS 4.2.1	Size	64
CCS 4.2.1	Speed	64
IAR 5.51.6	None	96
IAR 5.51.6	Size	80
IAR 5.51.6	Speed	80
MSPGCC 4.8.0	None	158
MSPGCC 4.8.0	Size	68
MSPGCC 4.8.0	Speed	660

Prototype:
```
void
TIMER_B_generatePWM(uint32_t baseAddress,
                    uint16_t clockSource,
                    uint16_t clockSourceDivider,
                    uint16_t timerPeriod,
                    uint16_t compareRegister,
                    uint16_t compareOutputMode,
                    uint16_t dutyCycle)
```

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_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **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

Description:
Modified bits of **TBxCCTLn** register, bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:
None

24.2.2.12 TIMER_B_getCaptureCompareCount

Get current capturecompare count.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	8
CCS 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	34
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

Prototype:

```
uint16_t
TIMER_B_getCaptureCompareCount(uint32_t baseAddress,
                               uint16_t captureCompareRegister)
```

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

24.2.2.13 uint32_t TIMER_B_getCaptureCompareInterruptStatus (uint32_t *baseAddress*, uint16_t *captureCompareRegister*, uint16_t *mask*)

Return capture compare interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	10
CCS 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

24.2.2.14 uint16_t TIMER_B_getCounterValue (uint32_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
CCS 4.2.1	None	98
CCS 4.2.1	Size	38
CCS 4.2.1	Speed	38
IAR 5.51.6	None	60
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	110
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

24.2.2.15 uint32_t TIMER_B_getInterruptStatus (uint32_t *baseAddress*)

Get TIMER_B interrupt status.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	22
CCS 4.2.1	Size	10
CCS 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	38
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

24.2.2.16 uint8_t TIMER_B_getOutputForOutputModeOutBitValue (uint32_t *baseAddress*, uint16_t *captureCompareRegister*)

Get output bit for output mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	16
CCS 4.2.1	Speed	16
IAR 5.51.6	None	26
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	16
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 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**

24.2.2.17 uint8_t TIMER_B_getSynchronizedCaptureCompareInput (uint32_t *baseAddress*, uint16_t *captureCompareRegister*, uint16_t *synchronized*)

Get synchronized capturecompare input.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	14
CCS 4.2.1	Speed	14
IAR 5.51.6	None	26
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**

24.2.2.18 void TIMER_B_initCapture (uint32_t *baseAddress*, uint16_t *captureRegister*, uint16_t *captureMode*, uint16_t *captureInputSelect*, uint16_t *synchronizeCaptureSource*, uint16_t *captureInterruptEnable*, uint16_t *captureOutputMode*)

Initializes Capture Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	78
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	48
IAR 5.51.6	None	62
IAR 5.51.6	Size	48
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	116
MSPGCC 4.8.0	Size	38
MSPGCC 4.8.0	Speed	38

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_CCIxA** [Default]
- **TIMER_B_CAPTURE_INPUTSELECT_CCIxB**
- **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**

Description:
Modified bits of **TBxCCTLn** register.

Returns:
None

24.2.2.19 TIMER_B_initCompare

Initializes Compare Mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	78
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	34
IAR 5.51.6	None	60
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	122
MSPGCC 4.8.0	Size	34
MSPGCC 4.8.0	Speed	34

Prototype:

```
void
TIMER_B_initCompare(uint32_t baseAddress,
                    uint16_t compareRegister,
                    uint16_t compareInterruptEnable,
                    uint16_t compareOutputMode,
                    uint16_t compareValue)
```

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

Description:
Modified bits of **TBxCCTLn** register and bits of **TBxCCRn** register.

Returns:
None

24.2.2.20 TIMER_B_initCompareLatchLoadEvent

Selects Compare Latch Load Event.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	14
CCS 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

Prototype:

```
void
TIMER_B_initCompareLatchLoadEvent(uint16_t baseAddress,
                                  uint16_t compareRegister,
                                  uint16_t compareLatchLoadEvent)
```

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**

Description:
Modified bits are **CLLD** of **TBxCCTLn** register.

Returns:
None

24.2.2.21 TIMER_B_selectCounterLength

Selects TIMER_B counter length.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	12
CCS 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

Prototype:

```
void
TIMER_B_selectCounterLength(uint16_t baseAddress,
                            uint16_t counterLength)
```

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**

Description:
Modified bits are **CNTL** of **TBxCTL** register.

Returns:
None

24.2.2.22 TIMER_B_selectLatchingGroup

Selects TIMER_B Latching Group.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	12
CCS 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

Prototype:

```
void
TIMER_B_selectLatchingGroup(uint16_t baseAddress,
                            uint16_t groupLatch)
```

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**

Description:

Modified bits are **TBCLGRP** of **TBxCTL** register.

Returns:

None

24.2.2.23 TIMER_B_setCompareValue

Sets the value of the capture-compare register.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	8
CCS 4.2.1	Speed	8
IAR 5.51.6	None	18
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	38
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

Prototype:

```
void
TIMER_B_setCompareValue(uint32_t baseAddress,
                        uint16_t compareRegister,
                        uint16_t compareValue)
```

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

Description:
Modified bits of **TBxCCRn** register.

Returns:
None

24.2.2.24 TIMER_B_setOutputForOutputModeOutBitValue

Set output bit for output mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	14
CCS 4.2.1	Speed	14
IAR 5.51.6	None	24
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

Prototype:
```
void
TIMER_B_setOutputForOutputModeOutBitValue(uint32_t baseAddress,
                                          uint16_t captureCompareRegister,
                                          uint8_t outputModeOutBitValue)
```

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**

Description:
Modified bits of **TBxCCTLn** register.

Returns:
None

24.2.2.25 TIMER_B_startContinousMode

DEPRECATED - Spelling Error Fixed. Starts TIMER_B in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	58
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	40
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	464

Prototype:

```
void
TIMER_B_startContinousMode(uint32_t baseAddress,
                           uint16_t clockSource,
                           uint16_t clockSourceDivider,
                           uint16_t timerInterruptEnable_TBIE,
                           uint16_t timerClear)
```

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_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **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]

Description:
Modified bits of **TBxCTL** register.

Returns:
None

24.2.2.26 TIMER_B_startContinuousMode

DEPRECATED - Replaced by TIMER_B_configureContinuousMode and TIMER_B_startCounter API. Starts TIMER_B in continuous mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	60
CCS 4.2.1	Size	40
CCS 4.2.1	Speed	40
IAR 5.51.6	None	52
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	86
MSPGCC 4.8.0	Size	40
MSPGCC 4.8.0	Speed	464

Prototype:
```
void
TIMER_B_startContinuousMode(uint32_t baseAddress,
                            uint16_t clockSource,
                            uint16_t clockSourceDivider,
                            uint16_t timerInterruptEnable_TBIE,
                            uint16_t timerClear)
```

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_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **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]

Description:
Modified bits of **TBxCTL** register.

Returns:
None

24.2.2.27 TIMER_B_startCounter

Starts TIMER_B counter.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	30
CCS 4.2.1	Size	6
CCS 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	38
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

Prototype:

```
void
TIMER_B_startCounter(uint32_t baseAddress,
                     uint16_t timerMode)
```

Description:
This function assumes that the timer has been previously configured using TIMER_B_configureContinuousMode, TIMER_B_configureUpMode or TIMER_B_configureUpDownMode.

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

24.2.2.28 TIMER_B_startUpDownMode

DEPRECATED - Replaced by TIMER_B_configureUpDownMode and TIMER_B_startCounter API. Starts TIMER_B in up down mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	98
CCS 4.2.1	Size	72
CCS 4.2.1	Speed	72
IAR 5.51.6	None	98
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	44
MSPGCC 4.8.0	None	156
MSPGCC 4.8.0	Size	78
MSPGCC 4.8.0	Speed	392

Prototype:

```
void
TIMER_B_startUpDownMode(uint32_t baseAddress,
                        uint16_t clockSource,
                        uint16_t clockSourceDivider,
                        uint16_t timerPeriod,
                        uint16_t timerInterruptEnable_TBIE,
                        uint16_t captureCompareInterruptEnable_CCR0_CCIE,
                        uint16_t timerClear)
```

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_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**
- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **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]

Description:

Modified bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:

None

24.2.2.29 TIMER_B_startUpMode

DEPRECATED - Replaced by TIMER_B_configureUpMode and TIMER_B_startCounter API. Starts TIMER_B in up mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	98
CCS 4.2.1	Size	72
CCS 4.2.1	Speed	72
IAR 5.51.6	None	98
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	44
MSPGCC 4.8.0	None	156
MSPGCC 4.8.0	Size	78
MSPGCC 4.8.0	Speed	392

Prototype:

```
void
TIMER_B_startUpMode(uint32_t baseAddress,
                    uint16_t clockSource,
                    uint16_t clockSourceDivider,
                    uint16_t timerPeriod,
                    uint16_t timerInterruptEnable_TBIE,
                    uint16_t captureCompareInterruptEnable_CCR0_CCIE,
                    uint16_t timerClear)
```

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_4**
- **TIMER_B_CLOCKSOURCE_DIVIDER_8**
- **TIMER_B_CLOCKSOURCE_DIVIDER_3**
- **TIMER_B_CLOCKSOURCE_DIVIDER_5**

- **TIMER_B_CLOCKSOURCE_DIVIDER_6**
- **TIMER_B_CLOCKSOURCE_DIVIDER_7**
- **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]

Description:

Modified bits of **TBxCTL** register, bits of **TBxCCTL0** register and bits of **TBxCCR0** register.

Returns:

None

24.2.2.30 TIMER_B_stop

Stops the TIMER_B.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	28
CCS 4.2.1	Size	12
CCS 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	38
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

Prototype:

```
void
TIMER_B_stop(uint32_t baseAddress)
```

Parameters:

baseAddress is the base address of the TIMER_B module.

Description:
Modified bits of **TBxCTL** register.

Returns:
None

24.3 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
          );
}
```

25 Tag Length Value

Introduction 304
API Functions 304
Programming Example 309

25.1 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 CCS 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 withing 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
CCS 4.2.1	None	602
CCS 4.2.1	Size	364
CCS 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

25.2 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)

25.2.1 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 is the specified interrupt vector is not defined.

25.2.2 Function Documentation

25.2.2.1 TLV_getDeviceType

Retrieves the unique device ID from the TLV structure.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	16
CCS 4.2.1	Size	8
CCS 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

Prototype:

```
uint16_t
TLV_getDeviceType(void)
```

Returns:

The device ID is returned as type uint16_t.

25.2.2.2 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
CCS 4.2.1	None	126
CCS 4.2.1	Size	72
CCS 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_LDTAG**
- **TLV_TAG_PDTAG**
- **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

25.2.2.3 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
CCS 4.2.1	None	160
CCS 4.2.1	Size	98
CCS 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 is the specified interrupt vector is not defined.

25.2.2.4 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
CCS 4.2.1	None	114
CCS 4.2.1	Size	70
CCS 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.

25.2.2.5 uint16_t TLV_getPeripheral (uint8_t *tag*, uint8_t *instance*)

Gets peripheral information from the TLV.

he 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
CCS 4.2.1	None	186
CCS 4.2.1	Size	116
CCS 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.

25.3 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
                        );
```

26 WatchDog Timer (WDT_A)

Introduction 310
API Functions 310
Programming Example 314

26.1 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 CCS 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 withing 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
CCS 4.2.1	None	222
CCS 4.2.1	Size	90
CCS 4.2.1	Speed	90
IAR 5.51.6	None	152
IAR 5.51.6	Size	54
IAR 5.51.6	Speed	54
MSPGCC 4.8.0	None	276
MSPGCC 4.8.0	Size	118
MSPGCC 4.8.0	Speed	118

26.2 API Functions

Functions

- void WDT_A_hold (uint32_t baseAddress)
- void WDT_A_intervalTimerInit (uint32_t baseAddress, uint8_t clockSelect, uint8_t clockDivider)
- void WDT_A_resetTimer (uint32_t baseAddress)
- void WDT_A_start (uint32_t baseAddress)
- void WDT_A_watchdogTimerInit (uint32_t baseAddress, uint8_t clockSelect, uint8_t clockDivider)

26.2.1 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()

26.2.2 Function Documentation

26.2.2.1 WDT_A_hold

Holds the Watchdog Timer.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	44
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	20
IAR 5.51.6	None	34
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Prototype:

```
void
WDT_A_hold(uint32_t baseAddress)
```

Description:
This function stops the watchdog timer from running, that way no interrupt or PUC is asserted.

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

Returns:
None

26.2.2.2 WDT_A_intervalTimerInit

Sets the clock source for the Watchdog Timer in timer interval mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	46
CCS 4.2.1	Size	16
CCS 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	52
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Prototype:

```
void
WDT_A_intervalTimerInit(uint32_t baseAddress,
                        uint8_t clockSelect,
                        uint8_t clockDivider)
```

Description:
This function sets the watchdog timer as timer interval mode, which will assert an interrupt without causing a PUC.

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 **WDTSSEL** 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

26.2.2.3 WDT_A_resetTimer

Resets the timer counter of the Watchdog Timer.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	42
CCS 4.2.1	Size	18
CCS 4.2.1	Speed	18
IAR 5.51.6	None	32
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

Prototype:
```
void
WDT_A_resetTimer(uint32_t baseAddress)
```

Description:
This function resets the watchdog timer to 0x0000h.

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

Returns:
None

26.2.2.4 WDT_A_start

Starts the Watchdog Timer.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	44
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	20
IAR 5.51.6	None	34
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Prototype:

```
void
WDT_A_start(uint32_t baseAddress)
```

Description:

This function starts the watchdog timer functionality to start counting again.

Parameters:

baseAddress is the base address of the WDT_A module.

Returns:

None

26.2.2.5 WDT_A_watchdogTimerInit

Sets the clock source for the Watchdog Timer in watchdog mode.

Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	46
CCS 4.2.1	Size	16
CCS 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	52
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

Prototype:

```
void
WDT_A_watchdogTimerInit(uint32_t baseAddress,
                        uint8_t clockSelect,
                        uint8_t clockDivider)
```

Description:

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.

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 **WDTSSEL** 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

26.3 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		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2014, Texas Instruments Incorporated