




# **MSP430® Peripheral Driver Library for F5xx and F6xx Devices**

## **USER'S GUIDE**

---

# Copyright

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

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

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



## Revision Information

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

# Table of Contents

<b>Copyright</b>	<b>1</b>
<b>Revision Information</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
<b>2 Navigating to driverlib through CCS Resource Explorer</b>	<b>5</b>
<b>3 How to create a new user project that uses Driverlib</b>	<b>15</b>
<b>4 How to include driverlib into your existing project</b>	<b>17</b>
<b>5 10-Bit Analog-to-Digital Converter (ADC10_A)</b>	<b>19</b>
5.1 Introduction	19
5.2 API Functions	20
5.3 Programming Example	41
<b>6 12-Bit Analog-to-Digital Converter (ADC12_A)</b>	<b>43</b>
6.1 Introduction	43
6.2 API Functions	43
6.3 Programming Example	62
<b>7 Advanced Encryption Standard (AES)</b>	<b>64</b>
7.1 Introduction	64
7.2 API Functions	64
7.3 Programming Example	75
<b>8 Battery Backup System</b>	<b>76</b>
8.1 Introduction	76
8.2 API Functions	76
8.3 Programming Example	80
<b>9 Comparator (COMP_B)</b>	<b>81</b>
9.1 Introduction	81
9.2 API Functions	82
9.3 Programming Example	96
<b>10 Cyclical Redundancy Check (CRC)</b>	<b>97</b>
10.1 Introduction	97
10.2 API Functions	97
10.3 Programming Example	102
<b>11 12-bit Digital-to-Analog Converter (DAC12_A)</b>	<b>103</b>
11.1 Introduction	103
11.2 API Functions	103
11.3 Programming Example	113
<b>12 Direct Memory Access (DMA)</b>	<b>114</b>
12.1 Introduction	114
12.2 API Functions	114
12.3 Programming Example	129
<b>13 EUSCI Universal Asynchronous Receiver/Transmitter (EUSCI_A_UART)</b>	<b>130</b>
13.1 Introduction	130
13.2 API Functions	130
13.3 Programming Example	139
<b>14 EUSCI Synchronous Peripheral Interface (EUSCI_A_SPI)</b>	<b>140</b>
14.1 Introduction	140

---

14.2	Functions	140
14.3	Programming Example	148
<b>15</b>	<b>EUSCI Synchronous Peripheral Interface (EUSCI_B_SPI)</b>	<b>149</b>
15.1	Introduction	149
15.2	Functions	149
15.3	Programming Example	157
<b>16</b>	<b>EUSCI Inter-Integrated Circuit (EUSCI_B_I2C)</b>	<b>158</b>
16.1	Introduction	158
16.2	API Functions	159
16.3	Programming Example	176
<b>17</b>	<b>Flash Memory Controller</b>	<b>178</b>
17.1	Introduction	178
17.2	API Functions	178
17.3	Programming Example	184
<b>18</b>	<b>GPIO</b>	<b>185</b>
18.1	Introduction	185
18.2	API Functions	186
18.3	Programming Example	204
<b>19</b>	<b>LDO-PWR</b>	<b>206</b>
19.1	Introduction	206
19.2	API Functions	206
19.3	Programming Example	214
<b>20</b>	<b>32-Bit Hardware Multiplier (MPY32)</b>	<b>217</b>
20.1	Introduction	217
20.2	API Functions	217
20.3	Programming Example	229
<b>21</b>	<b>Port Mapping Controller</b>	<b>230</b>
21.1	Introduction	230
21.2	API Functions	230
21.3	Programming Example	231
<b>22</b>	<b>Power Management Module (PMM)</b>	<b>233</b>
22.1	Introduction	233
22.2	API Functions	234
22.3	Programming Example	245
<b>23</b>	<b>RAM Controller</b>	<b>246</b>
23.1	Introduction	246
23.2	API Functions	246
23.3	Programming Example	248
<b>24</b>	<b>Internal Reference (REF)</b>	<b>250</b>
24.1	Introduction	250
24.2	API Functions	250
24.3	Programming Example	258
<b>25</b>	<b>Real-Time Clock (RTC)</b>	<b>260</b>
25.1	Introduction	260
25.2	API Functions	260
25.3	Programming Example	276
<b>26</b>	<b>Real-Time Clock (RTC_B)</b>	<b>278</b>
26.1	Introduction	278

26.2	API Functions	278
26.3	Programming Example	287
<b>27</b>	<b>Real-Time Clock (RTC_C)</b>	<b>288</b>
27.1	Introduction	288
27.2	API Functions	288
27.3	Programming Example	306
<b>28</b>	<b>24-Bit Sigma Delta Converter (SD24_B)</b>	<b>308</b>
28.1	Introduction	308
28.2	API Functions	308
28.3	Programming Example	326
<b>29</b>	<b>SFR Module</b>	<b>327</b>
29.1	Introduction	327
29.2	API Functions	327
29.3	Programming Example	332
<b>30</b>	<b>SYS Module</b>	<b>334</b>
30.1	Introduction	334
30.2	API Functions	334
30.3	Programming Example	345
<b>31</b>	<b>TEC</b>	<b>346</b>
31.1	Introduction	346
31.2	API Functions	346
31.3	Programming Example	353
<b>32</b>	<b>TIMER_A</b>	<b>355</b>
32.1	Introduction	355
32.2	API Functions	356
32.3	Programming Example	380
<b>33</b>	<b>TIMER_B</b>	<b>382</b>
33.1	Introduction	382
33.2	API Functions	383
33.3	Programming Example	410
<b>34</b>	<b>TIMER_D</b>	<b>411</b>
34.1	Introduction	411
34.2	API Functions	412
34.3	Programming Example	436
<b>35</b>	<b>Tag Length Value</b>	<b>438</b>
35.1	Introduction	438
35.2	API Functions	438
35.3	Programming Example	443
<b>36</b>	<b>Unified Clock System (UCS)</b>	<b>444</b>
36.1	Introduction	444
36.2	API Functions	445
36.3	Programming Example	462
<b>37</b>	<b>USCI Universal Asynchronous Receiver/Transmitter (USCI_A_UART)</b>	<b>463</b>
37.1	Introduction	463
37.2	API Functions	464
37.3	Programming Example	475
<b>38</b>	<b>USCI Synchronous Peripheral Interface (USCI_A_SPI)</b>	<b>477</b>
38.1	Introduction	477

---

38.2	API Functions	477
38.3	Programming Example	488
<b>39</b>	<b>USCI Synchronous Peripheral Interface (USCI_B_SPI)</b>	<b>490</b>
39.1	Introduction	490
39.2	API Functions	490
39.3	Programming Example	501
<b>40</b>	<b>USCI Inter-Integrated Circuit (USCI_B_I2C)</b>	<b>503</b>
40.1	Introduction	503
40.2	API Functions	505
40.3	Programming Example	529
<b>41</b>	<b>WatchDog Timer (WDT_A)</b>	<b>530</b>
41.1	Introduction	530
41.2	API Functions	530
41.3	Programming Example	534
	<b>IMPORTANT NOTICE</b>	<b>535</b>

# 1 Introduction

The Texas Instruments® MSP430® Peripheral Driver Library is a set of drivers for accessing the peripherals found on the MSP430 5xx/6xx family of microcontrollers. While they are not drivers in the pure operating system sense (that is, they do not have a common interface and do not connect into a global device driver infrastructure), they do provide a mechanism that makes it easy to use the device's peripherals.

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

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

Some consequences of these design goals are:

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

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

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

The following tool chains are supported:

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

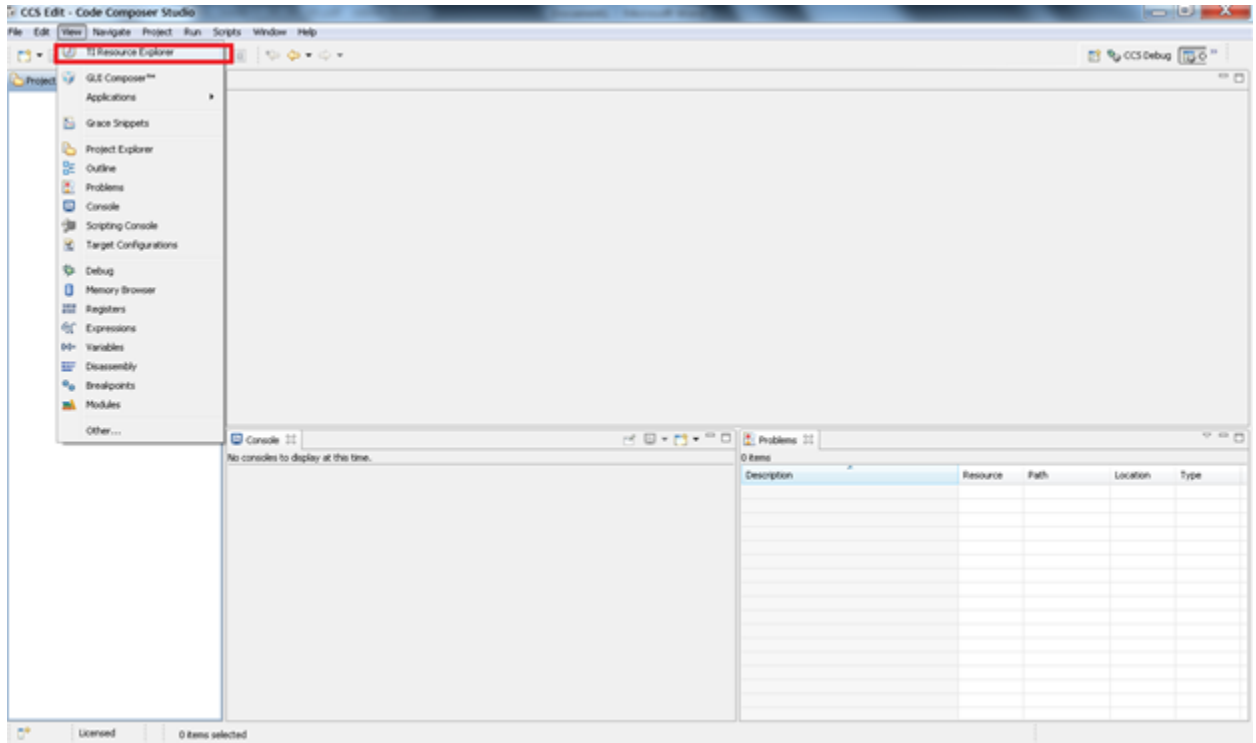
Using assert statements to debug

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

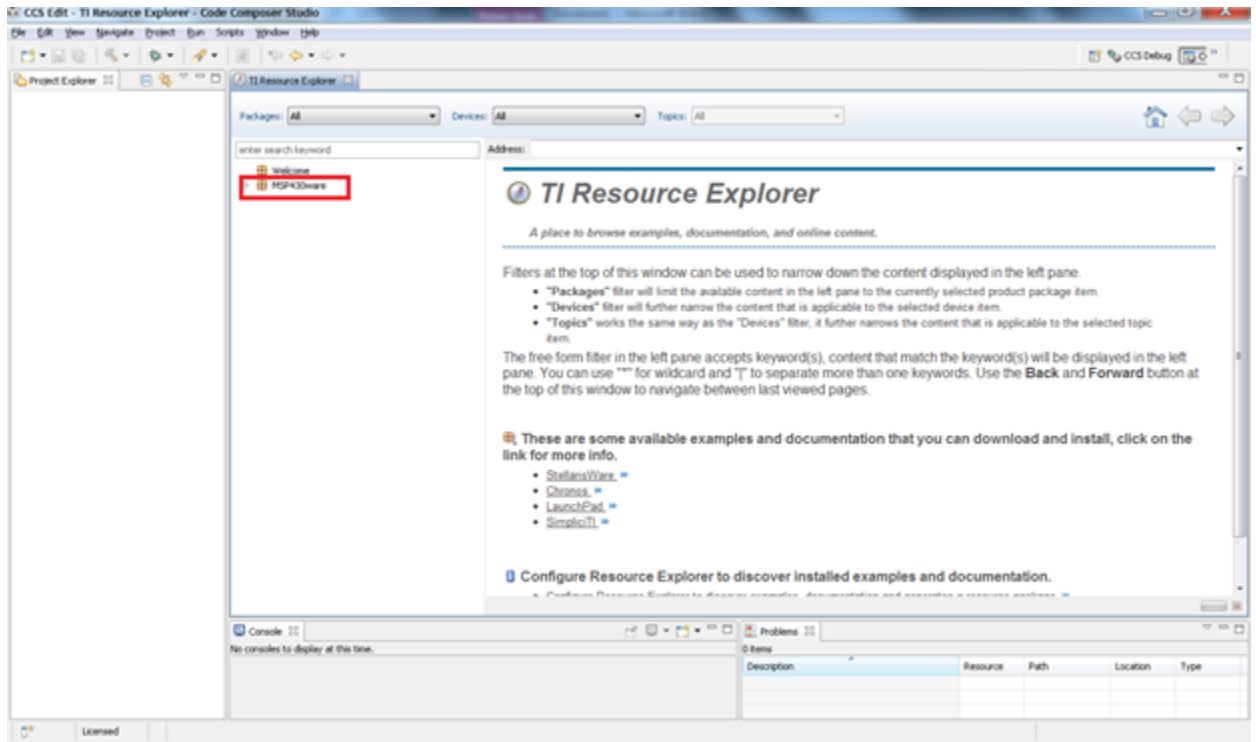


## 2 Navigating to driverlib through CCS Resource Explorer

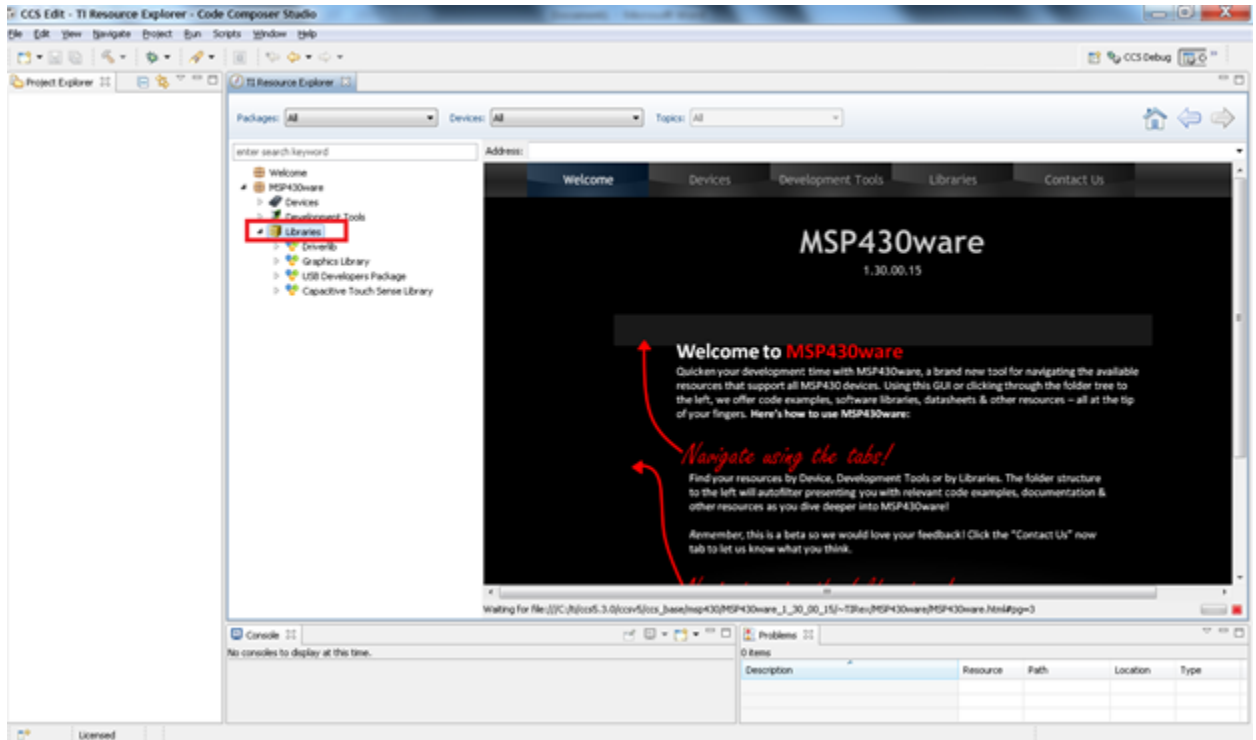
In CCS, click View->TI Resource Explorer

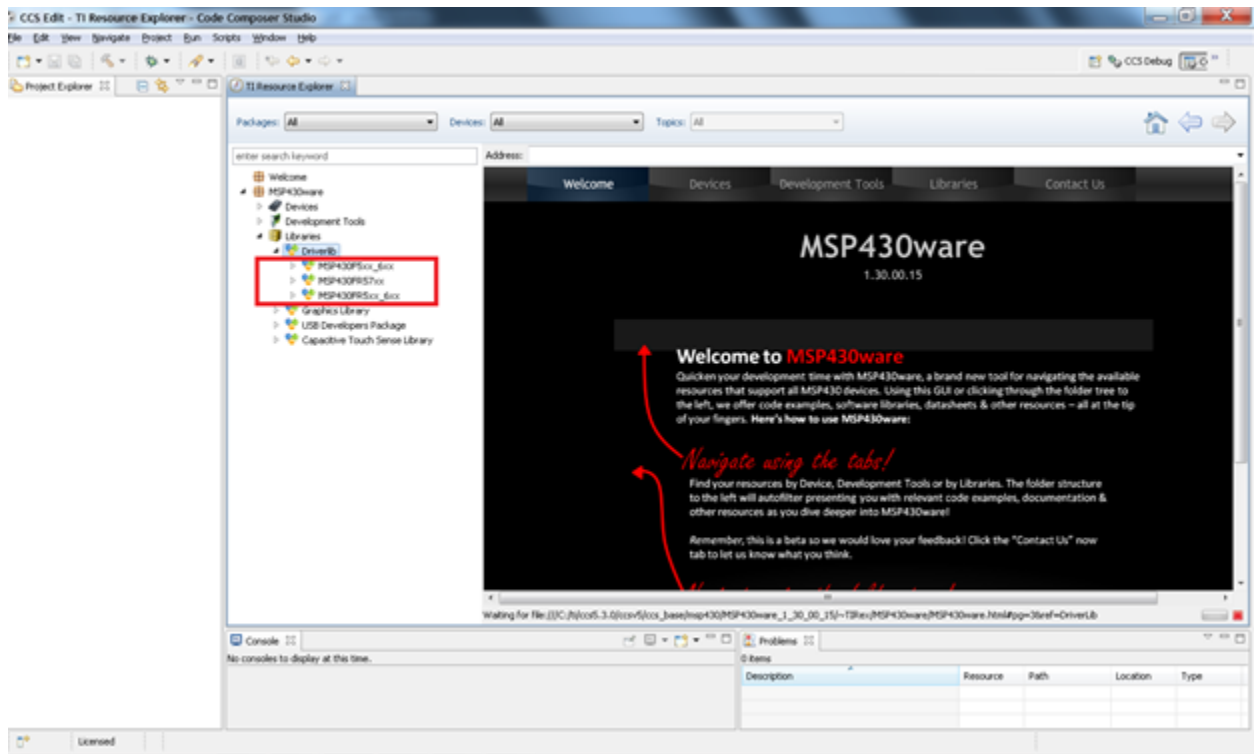


In Resource Explorer View, click on MSP430ware

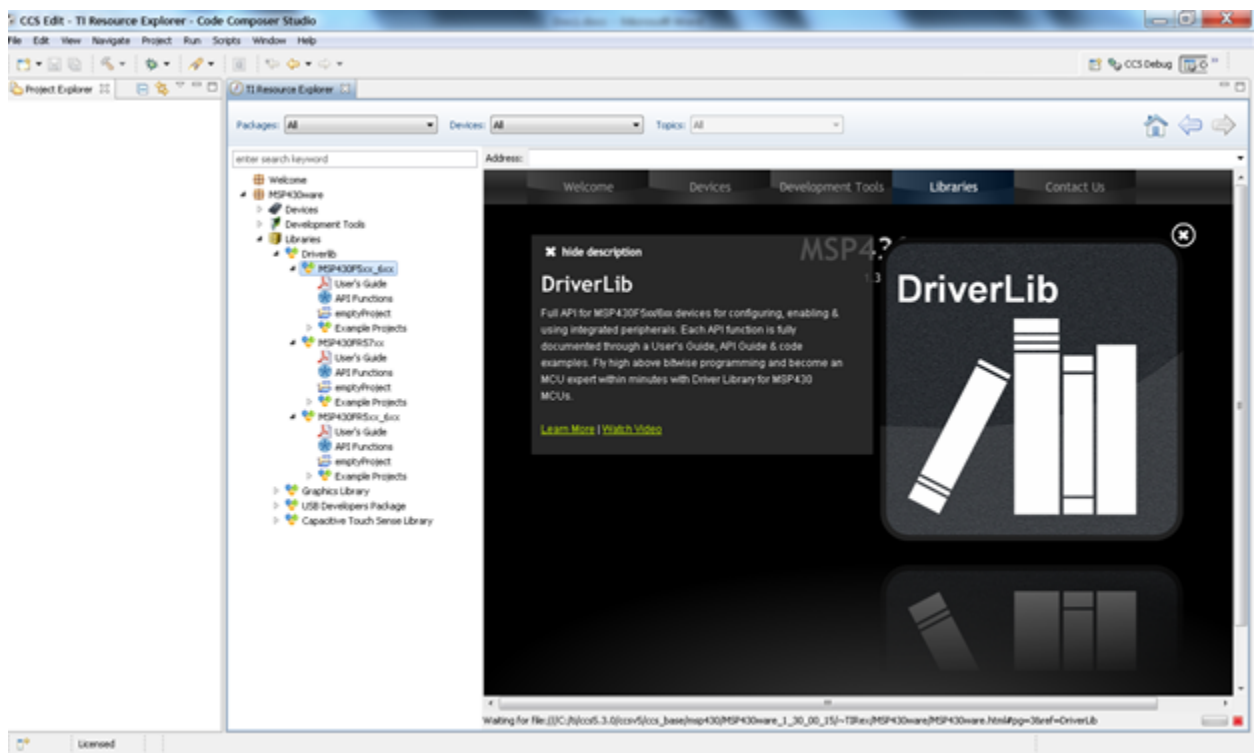


Clicking MSP430ware takes you to the introductory page. The version of the latest MSP430ware installed is available in this page. In this screenshot the version is 1.30.00.15 The various software, collateral, code examples, datasheets and user guides can be navigated by clicking the different topics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

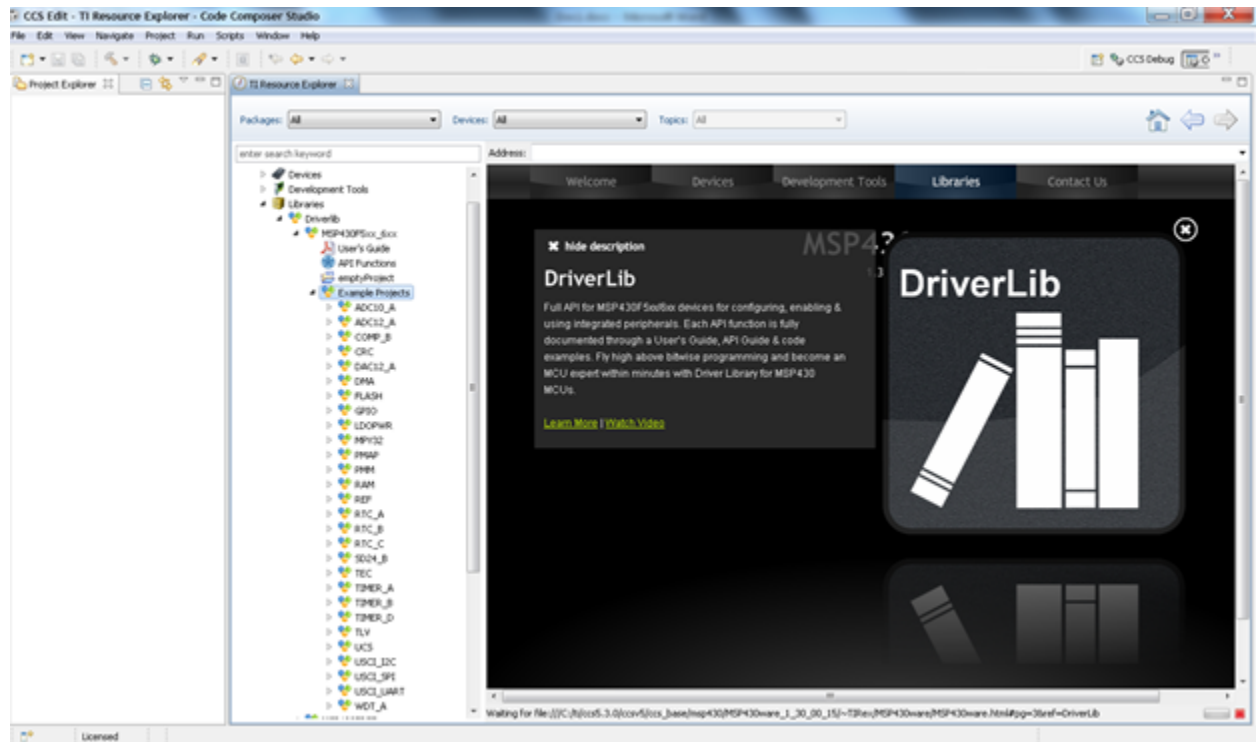




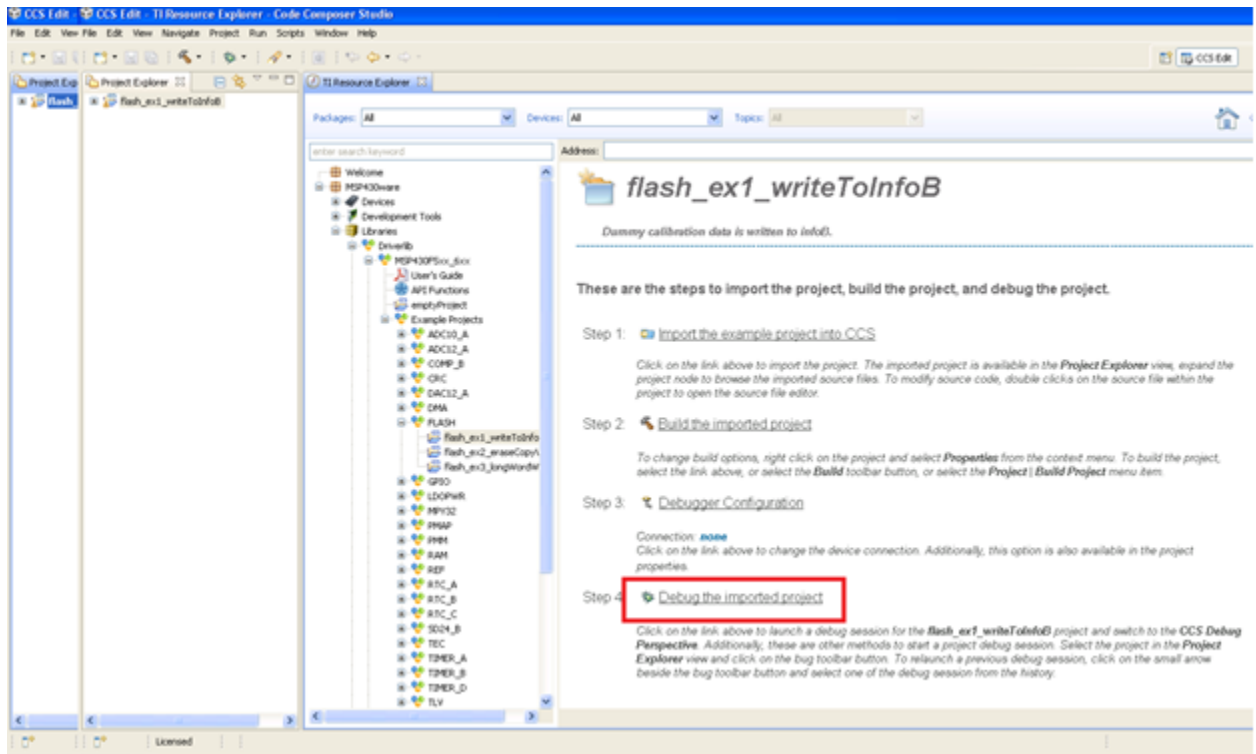
Driverlib is designed per Family. If a common device family user's guide exists for a group of devices, these devices belong to the same 'family'. Currently driverlib is available for the following family of devices. MSP430F5xx\_6xx MSP430FR57xx MSP430FR5xx\_6xx



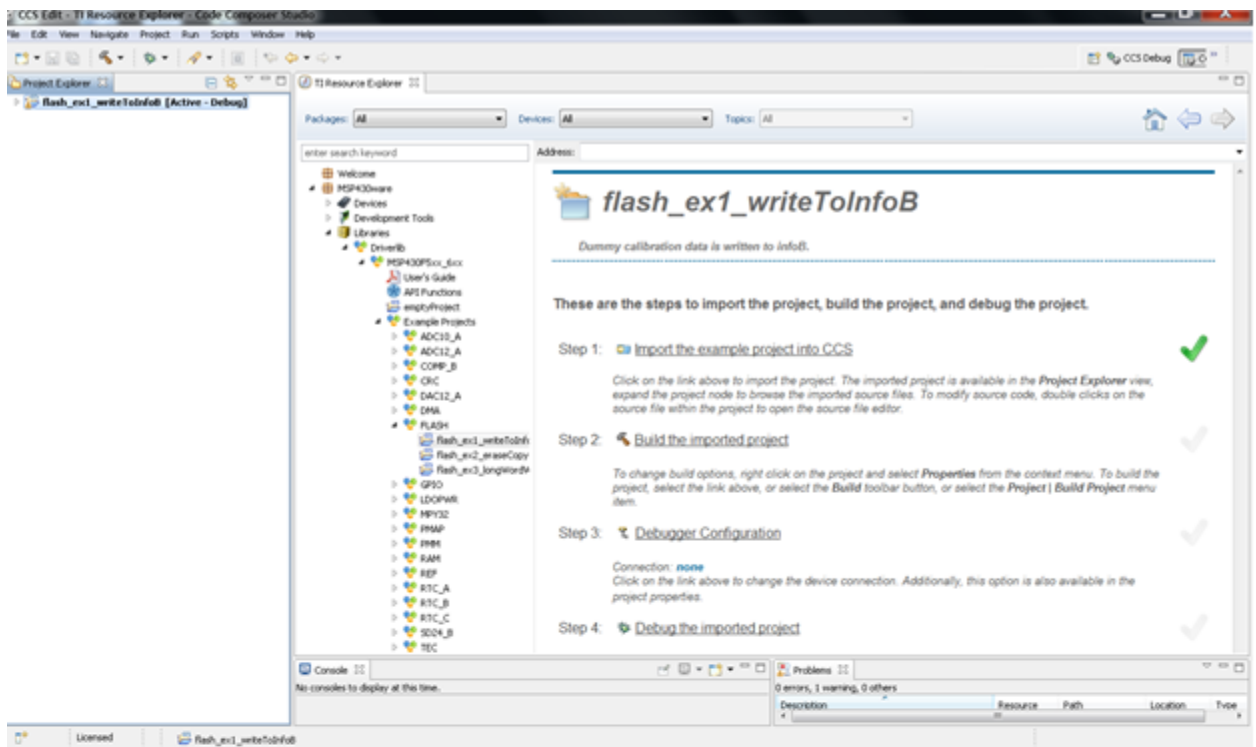
Click on the MSP430F5xx\_6xx to navigate to the driverlib based example code for that family.

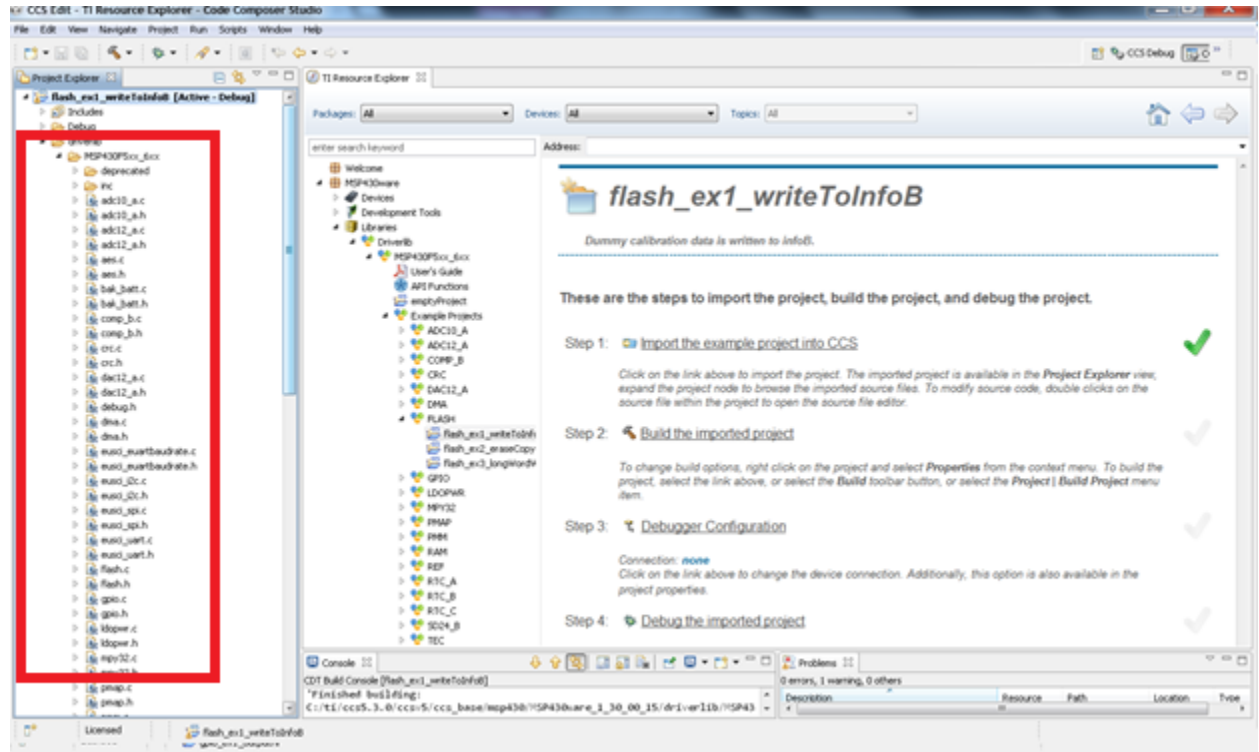


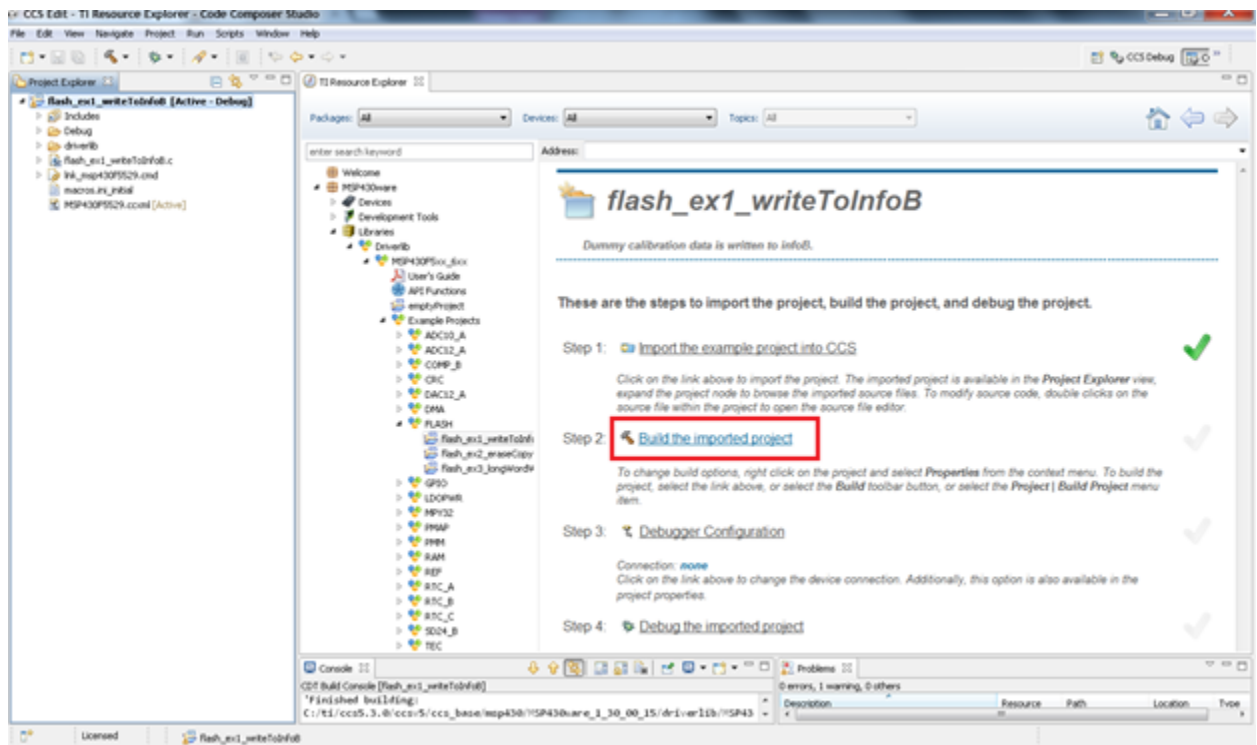
The various peripherals are listed in alphabetical order. The names of peripherals are as in device family user's guide. Clicking on a peripheral name lists the driverlib example code for that peripheral. The screenshot below shows an example when the user clicks on GPIO peripheral.



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

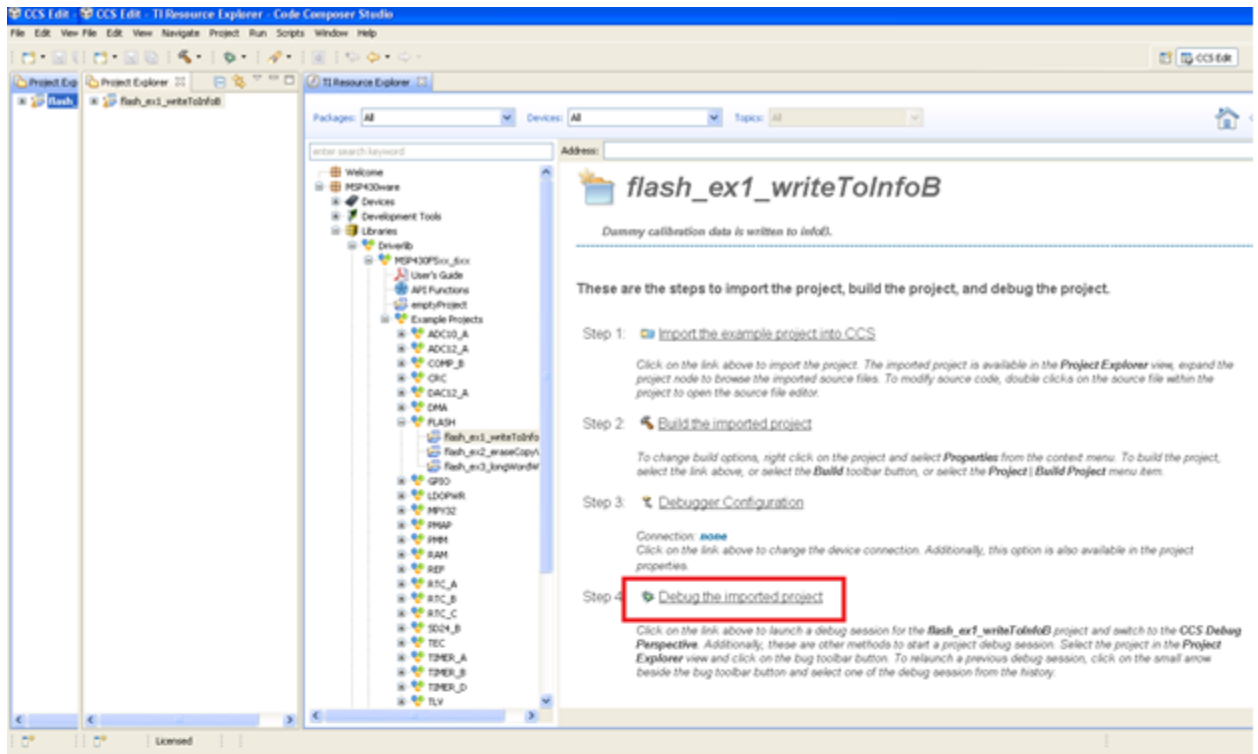






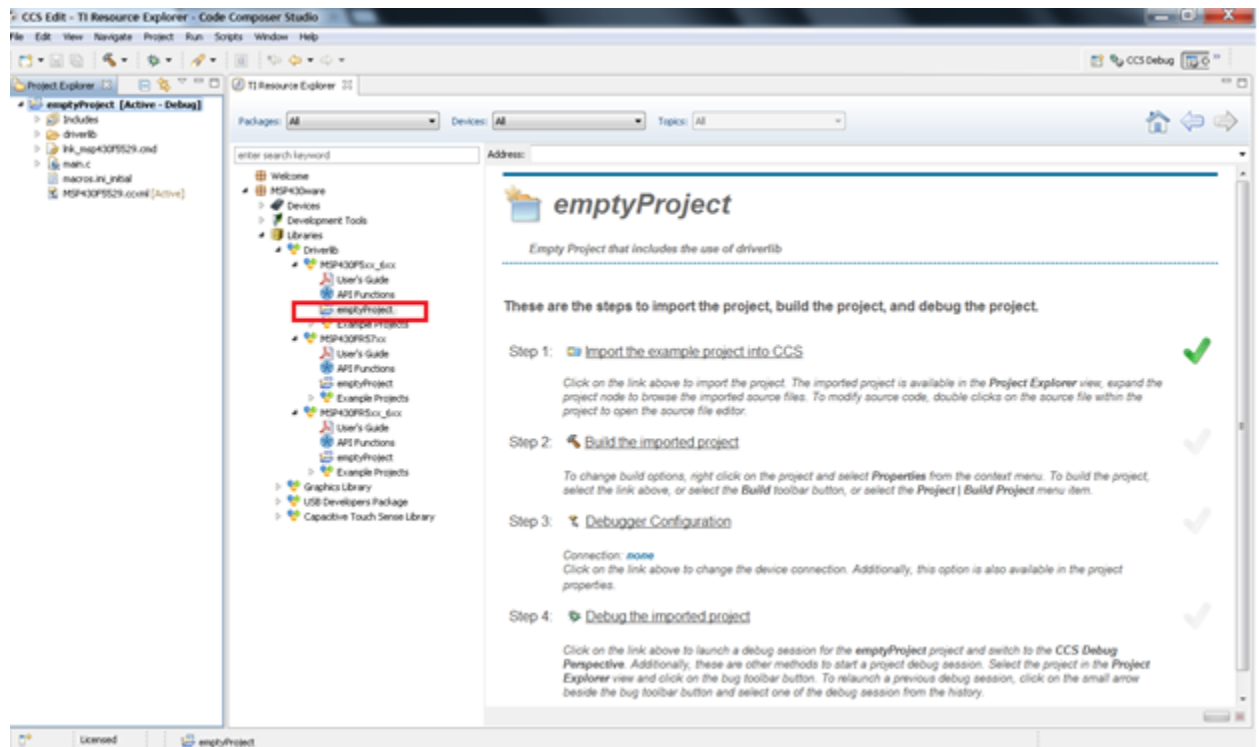
Now click on Build the imported project on the right to build the example project.





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

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

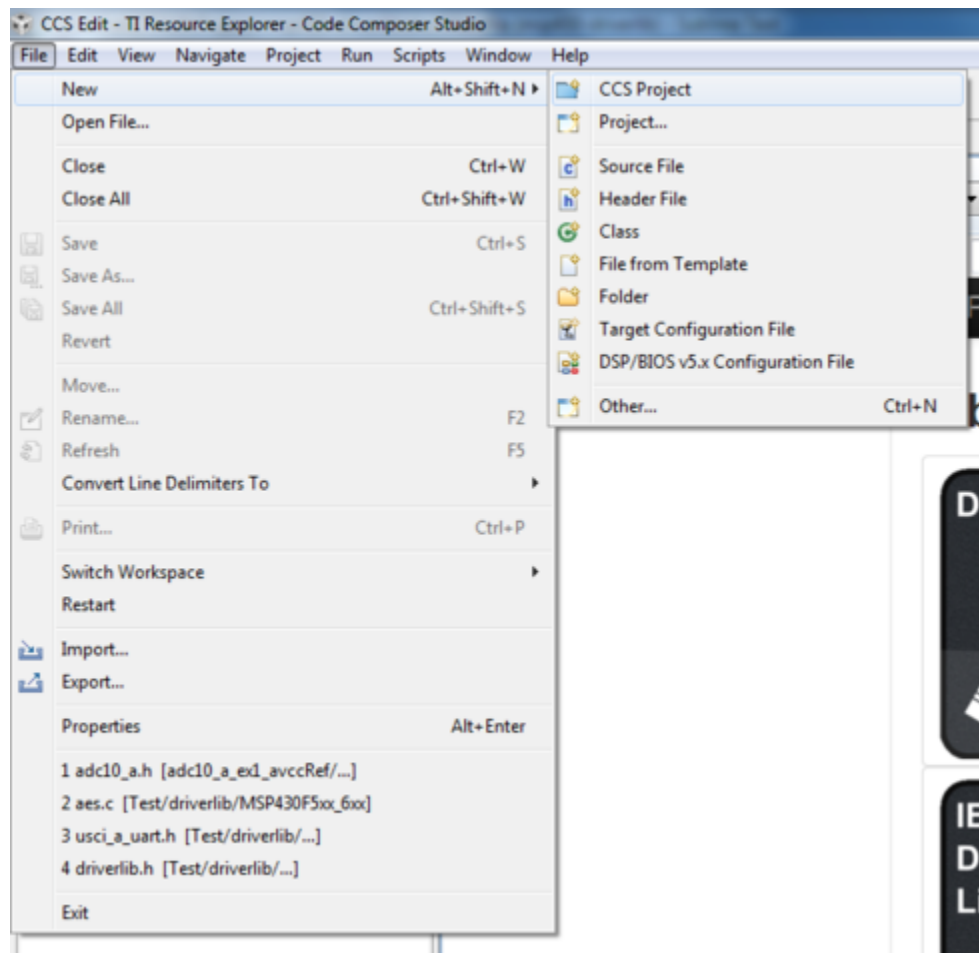


The main.c included with the empty project can be modified to include user code.

### 3 How to create a new user project that uses Driverlib

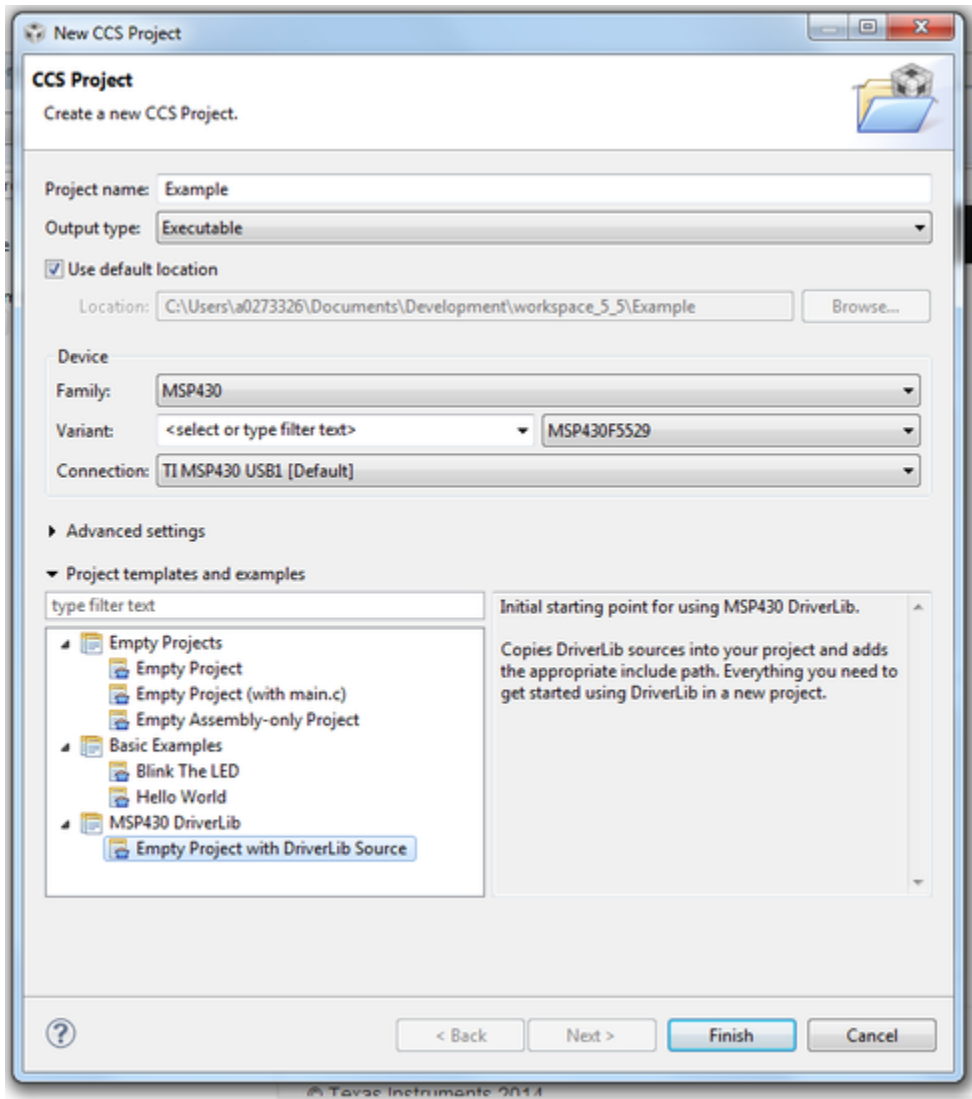
To get started on a new project we recommend using the new project wizard. For driver library to work with the new project wizard CCS must have discovered the driver library RTSC product. For more information refer to the installation steps of the release notes. The new project wizard adds the needed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

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



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

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

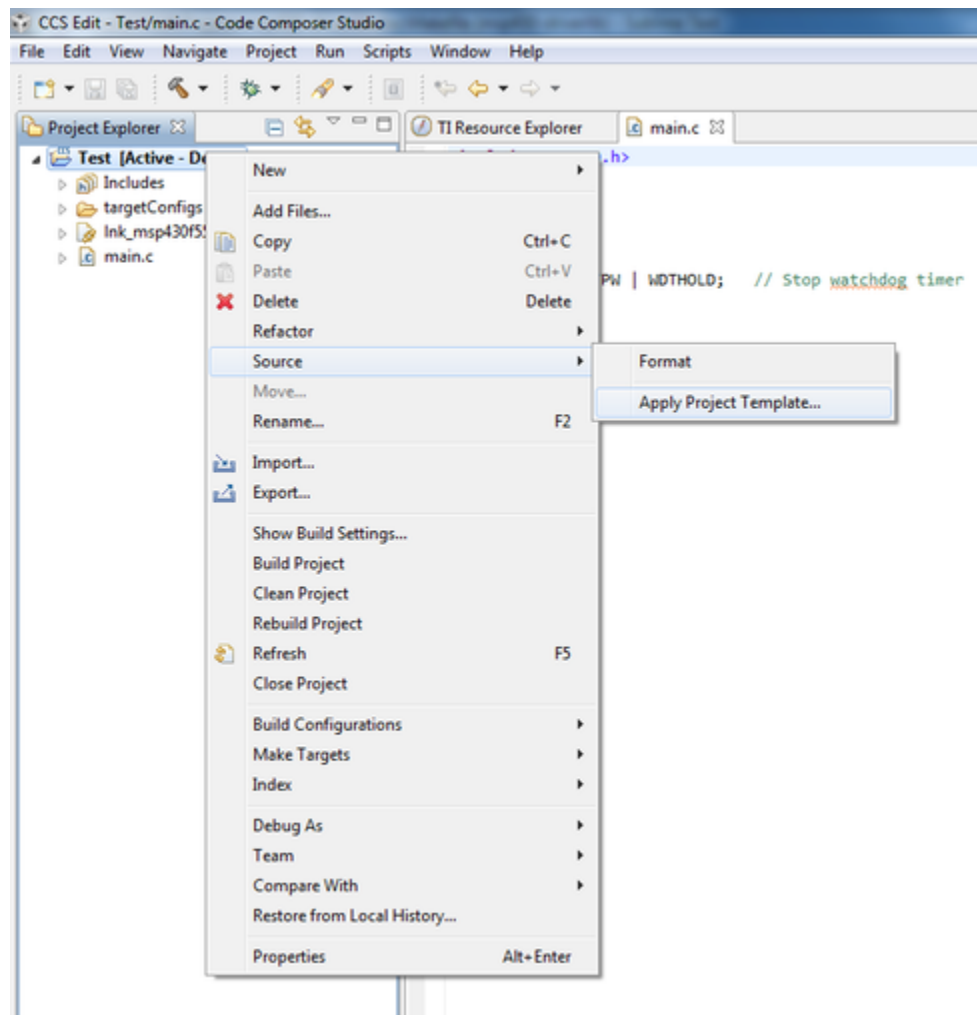


Finally click "Finish" and begin developing with your Driver Library enabled project.

## 4 How to include driverlib into your existing project

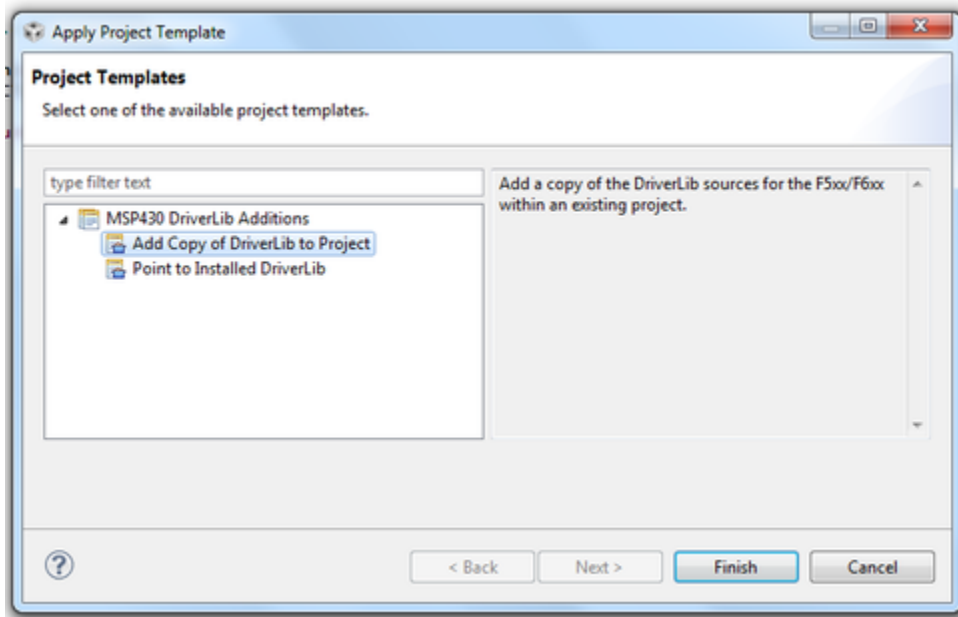
To add driver library to an existing project we recommend using CCS project templates. For driver library to work with project templates CCS must have discovered the driver library RTSC product. For more information refer to the installation steps of the release notes. CCS project templates adds the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

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



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

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



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

## 5 10-Bit Analog-to-Digital Converter (ADC10\_A)

Introduction .....	19
API Functions .....	20
Programming Example .....	41

### 5.1 Introduction

The 10-Bit Analog-to-Digital (ADC10\_A) API provides a set of functions for using the MSP430Ware ADC10\_A modules. Functions are provided to initialize the ADC10\_A modules, setup signal sources and reference voltages, and manage interrupts for the ADC10\_A modules.

The ADC10\_A module provides the ability to convert analog signals into a digital value in respect to given reference voltages. The ADC10\_A can generate digital values from 0 to Vcc with an 8- or 10-bit resolution. It operates in 2 different sampling modes, and 4 different conversion modes. The sampling modes are extended sampling and pulse sampling, in extended sampling the sample/hold signal must stay high for the duration of sampling, while in pulse mode a sampling timer is setup to start on a rising edge of the sample/hold signal and sample for a specified amount of clock cycles. The 4 conversion modes are single-channel single conversion, sequence of channels single-conversion, repeated single channel conversions, and repeated sequence of channels conversions.

The ADC10\_A module can generate multiple interrupts. An interrupt can be asserted when a conversion is complete, when a conversion is about to overwrite the converted data in the memory buffer before it has been read out, and/or when a conversion is about to start before the last conversion is complete. The ADC10\_A also has a window comparator feature which asserts interrupts when the input signal is above a high threshold, below a low threshold, or between the two at any given moment.

This driver is contained in `adc10_a.c`, with `adc10_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	846
CCS 4.2.1	Size	296
CCS 4.2.1	Speed	294
IAR 5.51.6	None	448
IAR 5.51.6	Size	338
IAR 5.51.6	Speed	338
MSPGCC 4.8.0	None	1192
MSPGCC 4.8.0	Size	336
MSPGCC 4.8.0	Speed	338

## 5.2 API Functions

### Functions

- void [ADC10\\_A\\_clearInterrupt](#) (uint32\_t baseAddress, uint8\_t interruptFlagMask)
- void [ADC10\\_A\\_disable](#) (uint32\_t baseAddress)
- void [ADC10\\_A\\_disableConversions](#) (uint32\_t baseAddress, bool preempt)
- void [ADC10\\_A\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t interruptMask)
- void [ADC10\\_A\\_disableReferenceBurst](#) (uint32\_t baseAddress)
- void [ADC10\\_A\\_disableSamplingTimer](#) (uint32\_t baseAddress)
- void [ADC10\\_A\\_enable](#) (uint32\_t baseAddress)
- void [ADC10\\_A\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t interruptMask)
- void [ADC10\\_A\\_enableReferenceBurst](#) (uint32\_t baseAddress)
- uint8\_t [ADC10\\_A\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t interruptFlagMask)
- uint32\_t [ADC10\\_A\\_getMemoryAddressForDMA](#) (uint32\_t baseAddress)
- int16\_t [ADC10\\_A\\_getResults](#) (uint32\_t baseAddress)
- bool [ADC10\\_A\\_init](#) (uint32\_t baseAddress, uint16\_t sampleHoldSignalSourceSelect, uint8\_t clockSourceSelect, uint16\_t clockSourceDivider)
- uint16\_t [ADC10\\_A\\_isBusy](#) (uint32\_t baseAddress)
- void [ADC10\\_A\\_memoryConfigure](#) (uint32\_t baseAddress, uint8\_t inputSourceSelect, uint8\_t positiveRefVoltageSourceSelect, uint8\_t negativeRefVoltageSourceSelect)
- void [ADC10\\_A\\_setDataReadBackFormat](#) (uint32\_t baseAddress, uint16\_t readBackFormat)
- void [ADC10\\_A\\_setReferenceBufferSamplingRate](#) (uint32\_t baseAddress, uint16\_t samplingRateSelect)
- void [ADC10\\_A\\_setResolution](#) (uint32\_t baseAddress, uint8\_t resolutionSelect)
- void [ADC10\\_A\\_setSampleHoldSignalInversion](#) (uint32\_t baseAddress, uint16\_t invertedSignal)
- void [ADC10\\_A\\_setupSamplingTimer](#) (uint32\_t baseAddress, uint16\_t clockCycleHoldCount, uint16\_t multipleSamplesEnabled)
- void [ADC10\\_A\\_setWindowComp](#) (uint32\_t baseAddress, uint16\_t highThreshold, uint16\_t lowThreshold)
- void [ADC10\\_A\\_startConversion](#) (uint32\_t baseAddress, uint8\_t conversionSequenceModeSelect)



## 5.2.1 Detailed Description

The ADC10\_A 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\_A.

The ADC10\_A initialization and conversion functions are

- [ADC10\\_A\\_init\(\)](#)
- [ADC10\\_A\\_memoryConfigure\(\)](#)
- [ADC10\\_A\\_setupSamplingTimer\(\)](#)
- [ADC10\\_A\\_disableSamplingTimer\(\)](#)
- [ADC10\\_A\\_setWindowComp\(\)](#)
- [ADC10\\_A\\_startConversion\(\)](#)
- [ADC10\\_A\\_disableConversions\(\)](#)
- [ADC10\\_A\\_getResults\(\)](#)
- [ADC10\\_A\\_isBusy\(\)](#)

The ADC10\_A interrupts are handled by

- [ADC10\\_A\\_enableInterrupt\(\)](#)
- [ADC10\\_A\\_disableInterrupt\(\)](#)
- [ADC10\\_A\\_clearInterrupt\(\)](#)
- [ADC10\\_A\\_getInterruptStatus\(\)](#)

Auxiliary features of the ADC10\_A are handled by

- [ADC10\\_A\\_setResolution\(\)](#)
- [ADC10\\_A\\_setSampleHoldSignalInversion\(\)](#)
- [ADC10\\_A\\_setDataReadBackFormat\(\)](#)
- [ADC10\\_A\\_enableReferenceBurst\(\)](#)
- [ADC10\\_A\\_disableReferenceBurst\(\)](#)
- [ADC10\\_A\\_setReferenceBufferSamplingRate\(\)](#)
- [ADC10\\_A\\_getMemoryAddressForDMA\(\)](#)
- [ADC10\\_A\\_enable\(\)](#)
- [ADC10\\_A\\_disable\(\)](#)

## 5.2.2 Function Documentation

### 5.2.2.1 ADC10\_A\_clearInterrupt

Clears ADC10\_A 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_A_clearInterrupt(uint32_t baseAddress,
                      uint8_t interruptFlagMask)
```

**Description:**

The selected ADC10\_A 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 ADC10\_A 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\_A\_TIMEOVERFLOW\_INTFLAG** - Interrupts flag when a new conversion is starting before the previous one has finished
- **ADC10\_A\_OVERFLOW\_INTFLAG** - Interrupts flag when a new conversion is about to overwrite the previous one
- **ADC10\_A\_ABOVEHRESHOLD\_INTFLAG** - Interrupts flag when the input signal has gone above the high threshold of the window comparator
- **ADC10\_A\_BELOWTHRESHOLD\_INTFLAG** - Interrupts flag when the input signal has gone below the low threshold of the low window comparator
- **ADC10\_A\_INSIDEWINDOW\_INTFLAG** - Interrupts flag when the input signal is in between the high and low thresholds of the window comparator
- **ADC10\_A\_COMPLETED\_INTFLAG** - Interrupt flag for new conversion data in the memory buffer

Modified bits of **ADC10IFG** register.

**Returns:**

None

### 5.2.2.2 ADC10\_A\_disable

Disables the ADC10\_A 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_A_disable(uint32_t baseAddress)
```

**Description:**

This will disable operation of the ADC10\_A block.

**Parameters:**

**baseAddress** is the base address of the ADC10\_A module.

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

**Returns:**

None

### 5.2.2.3 ADC10\_A\_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_A_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_A_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 `ADC10_A_startConversion()` has been called, then this function has to be called to re-initialize the ADC, reconfigure a memory buffer control, enable/disable the sampling pulse mode, or change the internal reference voltage.

**Parameters:**

**baseAddress** is the base address of the ADC10\_A module.

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

- **ADC10\_A\_COMPLETECONVERSION** - Allows the ADC10\_A to end the current conversion before disabling conversions.
- **ADC10\_A\_PREEMPTCONVERSION** - Stops the ADC10\_A immediately, with unpredictable 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\_A\_disableInterrupt

Disables selected ADC10\_A 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_A_disableInterrupt(uint32_t baseAddress,
                        uint8_t interruptMask)
```

**Description:**

Disables the indicated ADC10\_A 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 ADC10\_A 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\_A\_TIMEOVERFLOW\_INT** - Interrupts when a new conversion is starting before the previous one has finished
- **ADC10\_A\_OVERFLOW\_INT** - Interrupts when a new conversion is about to overwrite the previous one
- **ADC10\_A\_ABOVETHRESHOLD\_INT** - Interrupts when the input signal has gone above the high threshold of the window comparator
- **ADC10\_A\_BELOWTHRESHOLD\_INT** - Interrupts when the input signal has gone below the low threshold of the low window comparator
- **ADC10\_A\_INSIDEWINDOW\_INT** - Interrupts when the input signal is in between the high and low thresholds of the window comparator
- **ADC10\_A\_COMPLETED\_INT** - Interrupt for new conversion data in the memory buffer

Modified bits of **ADC10IE** register.

**Returns:**

None

### 5.2.2.5 ADC10\_A\_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_A_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 ADC10\_A module.

**Returns:**

None

### 5.2.2.6 ADC10\_A\_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_A_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 ADC10\_A module.

**Returns:**

None

### 5.2.2.7 ADC10\_A\_enable

Enables the ADC10\_A 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_A_enable(uint32_t baseAddress)
```

**Description:**

This will enable operation of the ADC10\_A block.

**Parameters:**

**baseAddress** is the base address of the ADC10\_A module.

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

**Returns:**

None

### 5.2.2.8 ADC10\_A\_enableInterrupt

Enables selected ADC10\_A 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_A_enableInterrupt(uint32_t baseAddress,
                        uint8_t interruptMask)
```

**Description:**

Enables the indicated ADC10\_A 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 ADC10\_A 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\_A\_TIMEOVERFLOW\_INT** - Interrupts when a new conversion is starting before the previous one has finished
- **ADC10\_A\_OVERFLOW\_INT** - Interrupts when a new conversion is about to overwrite the previous one
- **ADC10\_A\_ABOVETHRESHOLD\_INT** - Interrupts when the input signal has gone above the high threshold of the window comparator
- **ADC10\_A\_BELOWTHRESHOLD\_INT** - Interrupts when the input signal has gone below the low threshold of the low window comparator
- **ADC10\_A\_INSIDEWINDOW\_INT** - Interrupts when the input signal is in between the high and low thresholds of the window comparator
- **ADC10\_A\_COMPLETED\_INT** - Interrupt for new conversion data in the memory buffer

Modified bits of **ADC10IE** register.

**Returns:**

None

### 5.2.2.9 ADC10\_A\_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_A_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 ADC10\_A module.

**Returns:**

None

## 5.2.2.10 ADC10\_A\_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_A_getInterruptStatus(uint32_t baseAddress,
                           uint8_t interruptFlagMask)
```

**Description:**

Returns the status of the selected interrupt flags.

**Parameters:**

**baseAddress** is the base address of the ADC10\_A 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\_A\_TIMEOVERFLOW\_INTFLAG** - Interrupts flag when a new conversion is starting before the previous one has finished
- **ADC10\_A\_OVERFLOW\_INTFLAG** - Interrupts flag when a new conversion is about to overwrite the previous one
- **ADC10\_A\_ABOVETHRESHOLD\_INTFLAG** - Interrupts flag when the input signal has gone above the high threshold of the window comparator
- **ADC10\_A\_BELOWTHRESHOLD\_INTFLAG** - Interrupts flag when the input signal has gone below the low threshold of the low window comparator
- **ADC10\_A\_INSIDEWINDOW\_INTFLAG** - Interrupts flag when the input signal is in between the high and low thresholds of the window comparator

- **ADC10\_A\_COMPLETED\_INTFLAG** - Interrupt flag for new conversion data in the memory buffer

**Returns:**

The current interrupt flag status for the corresponding mask.

### 5.2.2.11 ADC10\_A\_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_A_getMemoryAddressForDMA(uint32_t baseAddress)
```

**Description:**

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

**Parameters:**

***baseAddress*** is the base address of the ADC10\_A module.

**Returns:**

The memory address of the memory buffer

### 5.2.2.12 ADC10\_A\_getResults

Returns the raw contents of the specified memory buffer.

**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:**

```
int16_t
ADC10_A_getResults(uint32_t baseAddress)
```

**Description:**

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.

**Parameters:**

**baseAddress** is the base address of the ADC10\_A module.

**Returns:**

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

### 5.2.2.13 ADC10\_A\_init

Initializes the ADC10\_A Module.

**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

**Prototype:**

```
bool
ADC10_A_init(uint32_t baseAddress,
             uint16_t sampleHoldSignalSourceSelect,
```

```
uint8_t clockSourceSelect,
uint16_t clockSourceDivider)
```

**Description:**

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.

**Parameters:**

**baseAddress** is the base address of the ADC10\_A 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\_A\_SAMPLEHOLDSOURCE\_SC
- ADC10\_A\_SAMPLEHOLDSOURCE\_1
- ADC10\_A\_SAMPLEHOLDSOURCE\_2
- ADC10\_A\_SAMPLEHOLDSOURCE\_3

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

**clockSourceSelect** selects the clock that will be used by the ADC10\_A core and the sampling timer if a sampling pulse mode is enabled. Valid values are:

- **ADC10\_A\_CLOCKSOURCE\_ADC10OSC** [Default] - MODOSC 5 MHz oscillator from the UCS
- **ADC10\_A\_CLOCKSOURCE\_ACLK** - The Auxiliary Clock
- **ADC10\_A\_CLOCKSOURCE\_MCLK** - The Master Clock
- **ADC10\_A\_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\_A\_CLOCKDIVIDER\_1** [Default]
- **ADC10\_A\_CLOCKDIVIDER\_2**
- **ADC10\_A\_CLOCKDIVIDER\_3**
- **ADC10\_A\_CLOCKDIVIDER\_4**
- **ADC10\_A\_CLOCKDIVIDER\_5**
- **ADC10\_A\_CLOCKDIVIDER\_6**
- **ADC10\_A\_CLOCKDIVIDER\_7**
- **ADC10\_A\_CLOCKDIVIDER\_8**
- **ADC10\_A\_CLOCKDIVIDER\_12**
- **ADC10\_A\_CLOCKDIVIDER\_16**
- **ADC10\_A\_CLOCKDIVIDER\_20**
- **ADC10\_A\_CLOCKDIVIDER\_24**
- **ADC10\_A\_CLOCKDIVIDER\_28**
- **ADC10\_A\_CLOCKDIVIDER\_32**
- **ADC10\_A\_CLOCKDIVIDER\_64**
- **ADC10\_A\_CLOCKDIVIDER\_128**

- **ADC10\_A\_CLOCKDIVIDER\_192**
- **ADC10\_A\_CLOCKDIVIDER\_256**
- **ADC10\_A\_CLOCKDIVIDER\_320**
- **ADC10\_A\_CLOCKDIVIDER\_384**
- **ADC10\_A\_CLOCKDIVIDER\_448**
- **ADC10\_A\_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 ADC10\_A\_isBusy

Returns the busy status of the ADC10\_A core.

**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
ADC10_A_isBusy(uint32_t baseAddress)
```

**Description:**

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

**Parameters:**

**baseAddress** is the base address of the ADC10\_A module.

**Returns:**

One of the following:

- **ADC10\_A\_BUSY**
- **ADC10\_A\_NOTBUSY**  
indicating if there is a conversion currently taking place

### 5.2.2.15 ADC10\_A\_memoryConfigure

Configures the controls of the selected memory buffer.

#### 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

#### Prototype:

```
void
ADC10_A_memoryConfigure(uint32_t baseAddress,
                        uint8_t inputSourceSelect,
                        uint8_t positiveRefVoltageSourceSelect,
                        uint8_t negativeRefVoltageSourceSelect)
```

#### Description:

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.

#### Parameters:

**baseAddress** is the base address of the ADC10\_A module.

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

- ADC10\_A\_INPUT\_A0 [Default]
- ADC10\_A\_INPUT\_A1
- ADC10\_A\_INPUT\_A2
- ADC10\_A\_INPUT\_A3
- ADC10\_A\_INPUT\_A4
- ADC10\_A\_INPUT\_A5
- ADC10\_A\_INPUT\_A6
- ADC10\_A\_INPUT\_A7
- ADC10\_A\_INPUT\_A8
- ADC10\_A\_INPUT\_A9
- ADC10\_A\_INPUT\_TEMPSENSOR
- ADC10\_A\_INPUT\_BATTERYMONITOR
- ADC10\_A\_INPUT\_A12
- ADC10\_A\_INPUT\_A13

- **ADC10\_A\_INPUT\_A14**

- **ADC10\_A\_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\_A\_VREFPOS\_AVCC** [Default]

- **ADC10\_A\_VREFPOS\_EXT**

- **ADC10\_A\_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\_A\_VREFNEG\_AVSS**

- **ADC10\_A\_VREFNEG\_EXT**

Modified bits are **ADC10SREF** of **ADC10CTL0** register.

**Returns:**

None

### 5.2.2.16 ADC10\_A\_setDataReadBackFormat

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

**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

**Prototype:**

```
void
ADC10_A_setDataReadBackFormat(uint32_t baseAddress,
                               uint16_t readBackFormat)
```

**Description:**

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.

**Parameters:**

**baseAddress** is the base address of the ADC10\_A module.

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

- **ADC10\_A\_UNSIGNED\_BINARY** [Default]
  - **ADC10\_A\_SIGNED\_2SCOMPLEMENT**
- Modified bits are **ADC10DF** of **ADC10CTL2** register.

**Returns:**

None

### 5.2.2.17 ADC10\_A\_setReferenceBufferSamplingRate

Use to set the reference buffer's sampling rate.

**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

**Prototype:**

```
void
ADC10_A_setReferenceBufferSamplingRate(uint32_t baseAddress,
                                       uint16_t samplingRateSelect)
```

**Description:**

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.

**Parameters:**

**baseAddress** is the base address of the ADC10\_A module.

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

- **ADC10\_A\_MAXSAMPLINGRATE\_200KSPS** [Default]
- **ADC10\_A\_MAXSAMPLINGRATE\_50KSPS**

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

**Returns:**

None



### 5.2.2.18 ADC10\_A\_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_A_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 ADC10\_A module.

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

- **ADC10\_A\_RESOLUTION\_8BIT**
  - **ADC10\_A\_RESOLUTION\_10BIT** [Default]
- Modified bits are **ADC10RES** of **ADC10CTL2** register.

#### Returns:

None

### 5.2.2.19 ADC10\_A\_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_A_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 ADC10\_A module.

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

- **ADC10\_A\_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\_A\_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\_A\_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_A_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 ADC10\_A module.

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

- **ADC10\_A\_CYCLEHOLD\_4\_CYCLES** [Default]
- **ADC10\_A\_CYCLEHOLD\_8\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_16\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_32\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_64\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_96\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_128\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_192\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_256\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_384\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_512\_CYCLES**
- **ADC10\_A\_CYCLEHOLD\_768\_CYCLES**
- **ADC10\_A\_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\_A\_MULTIPLESAMPLESDISABLE** - a timer trigger will be needed to start every ADC conversion.
- **ADC10\_A\_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\_A\_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_A_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 ADC10\_A module.

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

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

**Returns:**

None

### 5.2.2.22 ADC10\_A\_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_A_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\\_A\\_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_A_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 ADC10\_A module.

**conversionSequenceModeSelect** determines the ADC operating mode. Valid values are:

- **ADC10\_A\_SINGLECHANNEL** [Default] - one-time conversion of a single channel into a single memory buffer
  - **ADC10\_A\_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\_A\_REPEATED\_SINGLECHANNEL** - repeated conversions of one channel into a single memory buffer
  - **ADC10\_A\_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\_A API to start a single channel, single conversion.

```
// Initialize ADC10_A with ADC10_A's built-in oscillator
ADC10_A_init (ADC10_A_BASE,
              ADC10_A_SAMPLEHOLDSOURCE_SC,
              ADC10_A_CLOCKSOURCE_ADC10_AOSC,
              ADC10_A_CLOCKDIVIDEBY_1);

//Switch ON ADC10_A
ADC10_A_enable(ADC10_A_BASE);
```

```
// Setup sampling timer to sample-and-hold for 16 clock cycles
ADC10_A_setupSamplingTimer (ADC10_A_BASE,
                            ADC10_A_CYCLEHOLD_16_CYCLES,
                            FALSE);

// Configure the Input to the Memory Buffer with the specified Reference Voltages
ADC10_A_memoryConfigure (ADC10_A_BASE,
                         ADC10_A_INPUT_A0,
                         ADC10_A_VREF_AVCC, // Vref+ = AVcc
                         ADC10_A_VREF_AVSS  // Vref- = AVss
                         );

while (1)
{
    // Start a single conversion, no repeating or sequences.
    ADC10_A_startConversion (ADC10_A_BASE,
                             ADC10_A_SINGLECHANNEL);

    // Wait for the Interrupt Flag to assert
    while( !(ADC10_A_getInterruptStatus(ADC10_A_BASE,ADC10_A_IFG0)) );

    // Clear the Interrupt Flag and start another conversion
    ADC10_A_clearInterrupt (ADC10_A_BASE,ADC10_A_IFG0);
}
```

## 6 12-Bit Analog-to-Digital Converter (ADC12\_A)

Introduction .....	43
API Functions .....	43
Programming Example .....	62

### 6.1 Introduction

The 12-Bit Analog-to-Digital (ADC12\_A) API provides a set of functions for using the MSP430Ware ADC12\_A modules. Functions are provided to initialize the ADC12\_A modules, setup signal sources and reference voltages for each memory buffer, and manage interrupts for the ADC12\_A modules.

The ADC12\_A module provides the ability to convert analog signals into a digital value in respect to given reference voltages. The ADC12\_A can generate digital values from 0 to V<sub>cc</sub> with an 8-, 10- or 12-bit resolution, with 16 different memory buffers to store conversion results. It operates in 2 different sampling modes, and 4 different conversion modes. The sampling modes are extended sampling and pulse sampling, in extended sampling the sample/hold signal must stay high for the duration of sampling, while in pulse mode a sampling timer is setup to start on a rising edge of the sample/hold signal and sample for a specified amount of clock cycles. The 4 conversion modes are single-channel single conversion, sequence of channels single-conversion, repeated single channel conversions, and repeated sequence of channels conversions.

The ADC12\_A module can generate multiple interrupts. An interrupt can be asserted for each memory buffer when a conversion is complete, or when a conversion is about to overwrite the converted data in any of the memory buffers before it has been read out, and/or when a conversion is about to start before the last conversion is complete.

This driver is contained in `adc12_a.c`, with `adc12_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	1038
CCS 4.2.1	Size	442
CCS 4.2.1	Speed	434
IAR 5.51.6	None	568
IAR 5.51.6	Size	440
IAR 5.51.6	Speed	440
MSPGCC 4.8.0	None	1632
MSPGCC 4.8.0	Size	430
MSPGCC 4.8.0	Speed	432

### 6.2 API Functions

#### Functions

- void [ADC12\\_A\\_clearInterrupt](#) (uint32\_t baseAddress, uint16\_t memoryInterruptFlagMask)
- void [ADC12\\_A\\_disable](#) (uint32\_t baseAddress)
- void [ADC12\\_A\\_disableConversions](#) (uint32\_t baseAddress, bool preempt)
- void [ADC12\\_A\\_disableInterrupt](#) (uint32\_t baseAddress, uint32\_t interruptMask)
- void [ADC12\\_A\\_disableReferenceBurst](#) (uint32\_t baseAddress)

- void [ADC12\\_A\\_disableSamplingTimer](#) (uint32\_t baseAddress)
- void [ADC12\\_A\\_enable](#) (uint32\_t baseAddress)
- void [ADC12\\_A\\_enableInterrupt](#) (uint32\_t baseAddress, uint32\_t interruptMask)
- void [ADC12\\_A\\_enableReferenceBurst](#) (uint32\_t baseAddress)
- uint8\_t [ADC12\\_A\\_getInterruptStatus](#) (uint32\_t baseAddress, uint16\_t memoryInterruptFlagMask)
- uint32\_t [ADC12\\_A\\_getMemoryAddressForDMA](#) (uint32\_t baseAddress, uint8\_t memoryIndex)
- uint16\_t [ADC12\\_A\\_getResults](#) (uint32\_t baseAddress, uint8\_t memoryBufferIndex)
- bool [ADC12\\_A\\_init](#) (uint32\_t baseAddress, uint16\_t sampleHoldSignalSourceSelect, uint8\_t clockSourceSelect, uint16\_t clockSourceDivider)
- uint16\_t [ADC12\\_A\\_isBusy](#) (uint32\_t baseAddress)
- void [ADC12\\_A\\_memoryConfigure](#) (uint32\_t baseAddress, uint8\_t memoryBufferControlIndex, uint8\_t inputSourceSelect, uint8\_t positiveRefVoltageSourceSelect, uint8\_t negativeRefVoltageSourceSelect, uint8\_t endOfSequence)
- void [ADC12\\_A\\_setDataReadBackFormat](#) (uint32\_t baseAddress, uint8\_t readBackFormat)
- void [ADC12\\_A\\_setReferenceBufferSamplingRate](#) (uint32\_t baseAddress, uint8\_t samplingRateSelect)
- void [ADC12\\_A\\_setResolution](#) (uint32\_t baseAddress, uint8\_t resolutionSelect)
- void [ADC12\\_A\\_setSampleHoldSignalInversion](#) (uint32\_t baseAddress, uint16\_t invertedSignal)
- void [ADC12\\_A\\_setupSamplingTimer](#) (uint32\_t baseAddress, uint16\_t clockCycleHoldCountLowMem, uint16\_t clockCycleHoldCountHighMem, uint16\_t multipleSamplesEnabled)
- void [ADC12\\_A\\_startConversion](#) (uint32\_t baseAddress, uint16\_t startingMemoryBufferIndex, uint8\_t conversionSequenceModeSelect)

## 6.2.1 Detailed Description

The ADC12\_A API is broken into three groups of functions: those that deal with initialization and conversions, those that handle interrupts, and those that handle auxiliary features of the ADC12\_A.

The ADC12\_A initialization and conversion functions are

- [ADC12\\_A\\_init\(\)](#)
- [ADC12\\_A\\_memoryConfigure\(\)](#)
- [ADC12\\_A\\_setupSamplingTimer\(\)](#)
- [ADC12\\_A\\_disableSamplingTimer\(\)](#)
- [ADC12\\_A\\_startConversion\(\)](#)
- [ADC12\\_A\\_disableConversions\(\)](#)
- [ADC12\\_A\\_readResults\(\)](#)
- [ADC12\\_A\\_isBusy\(\)](#)

The ADC12\_A interrupts are handled by

- [ADC12\\_A\\_enableInterrupt\(\)](#)
- [ADC12\\_A\\_disableInterrupt\(\)](#)
- [ADC12\\_A\\_clearInterrupt\(\)](#)
- [ADC12\\_A\\_getInterruptStatus\(\)](#)

Auxiliary features of the ADC12\_A are handled by

- [ADC12\\_A\\_setResolution\(\)](#)
- [ADC12\\_A\\_setSampleHoldSignalInversion\(\)](#)
- [ADC12\\_A\\_setDataReadBackFormat\(\)](#)
- [ADC12\\_A\\_enableReferenceBurst\(\)](#)
- [ADC12\\_A\\_disableReferenceBurst\(\)](#)
- [ADC12\\_A\\_setReferenceBufferSamplingRate\(\)](#)
- [ADC12\\_A\\_getMemoryAddressForDMA\(\)](#)
- [ADC12\\_A\\_enable\(\)](#)
- [ADC12\\_A\\_disable\(\)](#)



## 6.2.2 Function Documentation

### 6.2.2.1 ADC12\_A\_clearInterrupt

Clears ADC12\_A selected 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	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	50
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

#### Prototype:

```
void
ADC12_A_clearInterrupt(uint32_t baseAddress,
                      uint16_t memoryInterruptFlagMask)
```

#### Description:

The selected ADC12\_A interrupt flags are cleared, so that it no longer asserts. The memory buffer interrupt flags are only cleared when the memory buffer is accessed. Note that the overflow interrupts do not have an interrupt flag to clear; they must be accessed directly from the interrupt vector.

#### Parameters:

**baseAddress** is the base address of the ADC12\_A module.

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

- ADC12\_A\_IFG0
- ADC12\_A\_IFG1
- ADC12\_A\_IFG2
- ADC12\_A\_IFG3
- ADC12\_A\_IFG4
- ADC12\_A\_IFG5
- ADC12\_A\_IFG6
- ADC12\_A\_IFG7
- ADC12\_A\_IFG8
- ADC12\_A\_IFG9
- ADC12\_A\_IFG10
- ADC12\_A\_IFG11
- ADC12\_A\_IFG12
- ADC12\_A\_IFG13
- ADC12\_A\_IFG14
- ADC12\_A\_IFG15

Modified bits of **ADC12IFG** register.

#### Returns:

None

### 6.2.2.2 ADC12\_A\_disable

Disables the ADC12\_A 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	36
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

#### Prototype:

```
void
ADC12_A_disable(uint32_t baseAddress)
```

#### Description:

This will disable operation of the ADC12\_A block.

#### Parameters:

**baseAddress** is the base address of the ADC12\_A module.

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

#### Returns:

None

### 6.2.2.3 ADC12\_A\_disableConversions

Disables the ADC from converting any more signals.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	76
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	50
IAR 5.51.6	Size	46
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	114
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	34

#### Prototype:

```
void
ADC12_A_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 TRUE, by changing the conversion mode to single-channel, single-conversion and disabling conversions. If the conversion mode is set as single-channel, single-conversion and this

function is called without preemption, then the ADC core conversion status is polled until the conversion is complete before disabling conversions to prevent unpredictable data. If the `ADC12_A_startConversion()` has been called, then this function has to be called to re-initialize the ADC, reconfigure a memory buffer control, enable/disable the sampling pulse mode, or change the internal reference voltage.

**Parameters:**

**baseAddress** is the base address of the ADC12\_A module.

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

- **ADC12\_A\_COMPLETECONVERSION** - Allows the ADC12\_A to end the current conversion before disabling conversions.
- **ADC12\_A\_PREEMPTCONVERSION** - Stops the ADC12\_A immediately, with unpredictable results of the current conversion.

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

**Returns:**

None

## 6.2.2.4 ADC12\_A\_disableInterrupt

Disables selected ADC12\_A interrupt sources.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	106
CCS 4.2.1	Size	52
CCS 4.2.1	Speed	52
IAR 5.51.6	None	40
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	138
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

**Prototype:**

```
void
ADC12_A_disableInterrupt (uint32_t baseAddress,
                          uint32_t interruptMask)
```

**Description:**

Disables the indicated ADC12\_A 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 ADC12\_A module.

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

- **ADC12\_A\_IE0**
- **ADC12\_A\_IE1**
- **ADC12\_A\_IE2**
- **ADC12\_A\_IE3**
- **ADC12\_A\_IE4**
- **ADC12\_A\_IE5**
- **ADC12\_A\_IE6**
- **ADC12\_A\_IE7**
- **ADC12\_A\_IE8**
- **ADC12\_A\_IE9**
- **ADC12\_A\_IE10**

- ADC12\_A\_IE11
- ADC12\_A\_IE12
- ADC12\_A\_IE13
- ADC12\_A\_IE14
- ADC12\_A\_IE15
- ADC12\_A\_OVERFLOW\_IE
- ADC12\_A\_CONVERSION\_TIME\_OVERFLOW\_IE

Modified bits of **ADC12IE** register and bits of **ADC12CTL0** register.

**Returns:**

None

### 6.2.2.5 ADC12\_A\_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	40
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
ADC12_A_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 ADC12\_A module.

**Returns:**

None

### 6.2.2.6 ADC12\_A\_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
ADC12_A_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 ADC12\_A module.

Modified bits are **ADC12SHP** of **ADC12CTL0** register.

**Returns:**

None

### 6.2.2.7 ADC12\_A\_enable

Enables the ADC12\_A 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	36
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
void
ADC12_A_enable(uint32_t baseAddress)
```

**Description:**

This will enable operation of the ADC12\_A block.

**Parameters:**

**baseAddress** is the base address of the ADC12\_A module.

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

**Returns:**

None

### 6.2.2.8 ADC12\_A\_enableInterrupt

Enables selected ADC12\_A interrupt sources.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	106
CCS 4.2.1	Size	52
CCS 4.2.1	Speed	52
IAR 5.51.6	None	40
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	136
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

**Prototype:**

```
void
ADC12_A_enableInterrupt(uint32_t baseAddress,
                        uint32_t interruptMask)
```

**Description:**

Enables the indicated ADC12\_A 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 ADC12\_A module.

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

- ADC12\_A\_IE0
- ADC12\_A\_IE1
- ADC12\_A\_IE2
- ADC12\_A\_IE3
- ADC12\_A\_IE4
- ADC12\_A\_IE5
- ADC12\_A\_IE6
- ADC12\_A\_IE7
- ADC12\_A\_IE8
- ADC12\_A\_IE9
- ADC12\_A\_IE10
- ADC12\_A\_IE11
- ADC12\_A\_IE12
- ADC12\_A\_IE13
- ADC12\_A\_IE14
- ADC12\_A\_IE15
- ADC12\_A\_OVERFLOW\_IE
- ADC12\_A\_CONVERSION\_TIME\_OVERFLOW\_IE

Modified bits of **ADC12IE** register and bits of **ADC12CTL0** register.

**Returns:**

None

### 6.2.2.9 ADC12\_A\_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	38
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
ADC12_A_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 ADC12\_A module.

**Returns:**

None

### 6.2.2.10 ADC12\_A\_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	52
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

**Prototype:**

```
uint8_t
ADC12_A_getInterruptStatus(uint32_t baseAddress,
                           uint16_t memoryInterruptFlagMask)
```

**Description:**

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

**Parameters:**

**baseAddress** is the base address of the ADC12\_A module.

**memoryInterruptFlagMask** is a bit mask of the interrupt flags status to be returned. Mask value is the logical OR of any of the following:

- ADC12\_A\_IFG0
- ADC12\_A\_IFG1
- ADC12\_A\_IFG2

- ADC12\_A\_IFG3
- ADC12\_A\_IFG4
- ADC12\_A\_IFG5
- ADC12\_A\_IFG6
- ADC12\_A\_IFG7
- ADC12\_A\_IFG8
- ADC12\_A\_IFG9
- ADC12\_A\_IFG10
- ADC12\_A\_IFG11
- ADC12\_A\_IFG12
- ADC12\_A\_IFG13
- ADC12\_A\_IFG14
- ADC12\_A\_IFG15

**Returns:**

The current interrupt flag status for the corresponding mask.

### 6.2.2.11 ADC12\_A\_getMemoryAddressForDMA

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

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	46
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	22
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	60
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

**Prototype:**

```
uint32_t
ADC12_A_getMemoryAddressForDMA(uint32_t baseAddress,
                               uint8_t memoryIndex)
```

**Description:**

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

**Parameters:**

**baseAddress** is the base address of the ADC12\_A module.

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

- ADC12\_A\_MEMORY\_0 [Default]
- ADC12\_A\_MEMORY\_1
- ADC12\_A\_MEMORY\_2
- ADC12\_A\_MEMORY\_3
- ADC12\_A\_MEMORY\_4
- ADC12\_A\_MEMORY\_5
- ADC12\_A\_MEMORY\_6
- ADC12\_A\_MEMORY\_7
- ADC12\_A\_MEMORY\_8
- ADC12\_A\_MEMORY\_9
- ADC12\_A\_MEMORY\_10



- ADC12\_A\_MEMORY\_11
- ADC12\_A\_MEMORY\_12
- ADC12\_A\_MEMORY\_13
- ADC12\_A\_MEMORY\_14
- ADC12\_A\_MEMORY\_15

**Returns:**

address of the specified memory buffer

### 6.2.2.12 ADC12\_A\_getResults

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

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	14
CCS 4.2.1	Speed	12
IAR 5.51.6	None	14
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	40
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

**Prototype:**

```
uint16_t
ADC12_A_getResults(uint32_t baseAddress,
                   uint8_t memoryBufferIndex)
```

**Description:**

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.

**Parameters:**

**baseAddress** is the base address of the ADC12\_A module.

**memoryBufferIndex** is the specified Memory Buffer to read. Valid values are:

- ADC12\_A\_MEMORY\_0 [Default]
- ADC12\_A\_MEMORY\_1
- ADC12\_A\_MEMORY\_2
- ADC12\_A\_MEMORY\_3
- ADC12\_A\_MEMORY\_4
- ADC12\_A\_MEMORY\_5
- ADC12\_A\_MEMORY\_6
- ADC12\_A\_MEMORY\_7
- ADC12\_A\_MEMORY\_8
- ADC12\_A\_MEMORY\_9
- ADC12\_A\_MEMORY\_10
- ADC12\_A\_MEMORY\_11
- ADC12\_A\_MEMORY\_12
- ADC12\_A\_MEMORY\_13
- ADC12\_A\_MEMORY\_14
- ADC12\_A\_MEMORY\_15

**Returns:**

A signed integer of the contents of the specified memory buffer

### 6.2.2.13 ADC12\_A\_init

Initializes the ADC12\_A Module.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	114
CCS 4.2.1	Size	64
CCS 4.2.1	Speed	62
IAR 5.51.6	None	104
IAR 5.51.6	Size	84
IAR 5.51.6	Speed	84
MSPGCC 4.8.0	None	176
MSPGCC 4.8.0	Size	80
MSPGCC 4.8.0	Speed	80

#### Prototype:

```
bool
ADC12_A_init(uint32_t baseAddress,
             uint16_t sampleHoldSignalSourceSelect,
             uint8_t clockSourceSelect,
             uint16_t clockSourceDivider)
```

#### Description:

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.

#### Parameters:

**baseAddress** is the base address of the ADC12\_A 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:

- **ADC12\_A\_SAMPLEHOLDSOURCE\_SC** [Default]
- **ADC12\_A\_SAMPLEHOLDSOURCE\_1**
- **ADC12\_A\_SAMPLEHOLDSOURCE\_2**
- **ADC12\_A\_SAMPLEHOLDSOURCE\_3** - This parameter is device specific and sources should be found in the device's datasheet. Modified bits are **ADC12SHSx** of **ADC12CTL1** register.

**clockSourceSelect** selects the clock that will be used by the ADC12\_A core, and the sampling timer if a sampling pulse mode is enabled. Valid values are:

- **ADC12\_A\_CLOCKSOURCE\_ADC12OSC** [Default] - MODOSC 5 MHz oscillator from the UCS
  - **ADC12\_A\_CLOCKSOURCE\_ACLK** - The Auxiliary Clock
  - **ADC12\_A\_CLOCKSOURCE\_MCLK** - The Master Clock
  - **ADC12\_A\_CLOCKSOURCE\_SMCLK** - The Sub-Master Clock
- Modified bits are **ADC12SSELx** of **ADC12CTL1** register.

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

- **ADC12\_A\_CLOCKDIVIDER\_1** [Default]
- **ADC12\_A\_CLOCKDIVIDER\_2**
- **ADC12\_A\_CLOCKDIVIDER\_3**
- **ADC12\_A\_CLOCKDIVIDER\_4**
- **ADC12\_A\_CLOCKDIVIDER\_5**
- **ADC12\_A\_CLOCKDIVIDER\_6**
- **ADC12\_A\_CLOCKDIVIDER\_7**
- **ADC12\_A\_CLOCKDIVIDER\_8**
- **ADC12\_A\_CLOCKDIVIDER\_12**

- **ADC12\_A\_CLOCKDIVIDER\_16**
- **ADC12\_A\_CLOCKDIVIDER\_20**
- **ADC12\_A\_CLOCKDIVIDER\_24**
- **ADC12\_A\_CLOCKDIVIDER\_28**
- **ADC12\_A\_CLOCKDIVIDER\_32**

Modified bits are **ADC12PDIV** of **ADC12CTL2** register; bits **ADC12DIVx** of **ADC12CTL1** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of the initialization process.

#### 6.2.2.14 ADC12\_A\_isBusy

Returns the busy status of the ADC12\_A core.

**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	26
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

**Prototype:**

```
uint16_t
ADC12_A_isBusy(uint32_t baseAddress)
```

**Description:**

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

**Parameters:**

**baseAddress** is the base address of the ADC12\_A module.

**Returns:**

One of the following:

- **ADC12\_A\_NOTBUSY**
- **ADC12\_A\_BUSY**  
indicating if a conversion is taking place

#### 6.2.2.15 ADC12\_A\_memoryConfigure

Configures the controls of the selected memory buffer.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	56
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	32
IAR 5.51.6	None	40
IAR 5.51.6	Size	36
IAR 5.51.6	Speed	36
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

**Prototype:**

```
void
ADC12_A_memoryConfigure(uint32_t baseAddress,
                        uint8_t memoryBufferControlIndex,
                        uint8_t inputSourceSelect,
                        uint8_t positiveRefVoltageSourceSelect,
                        uint8_t negativeRefVoltageSourceSelect,
                        uint8_t endOfSequence)
```

**Description:**

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

**Parameters:**

**baseAddress** is the base address of the ADC12\_A module.

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

- ADC12\_A\_MEMORY\_0 [Default]
- ADC12\_A\_MEMORY\_1
- ADC12\_A\_MEMORY\_2
- ADC12\_A\_MEMORY\_3
- ADC12\_A\_MEMORY\_4
- ADC12\_A\_MEMORY\_5
- ADC12\_A\_MEMORY\_6
- ADC12\_A\_MEMORY\_7
- ADC12\_A\_MEMORY\_8
- ADC12\_A\_MEMORY\_9
- ADC12\_A\_MEMORY\_10
- ADC12\_A\_MEMORY\_11
- ADC12\_A\_MEMORY\_12
- ADC12\_A\_MEMORY\_13
- ADC12\_A\_MEMORY\_14
- ADC12\_A\_MEMORY\_15

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

- ADC12\_A\_INPUT\_A0 [Default]
- ADC12\_A\_INPUT\_A1
- ADC12\_A\_INPUT\_A2
- ADC12\_A\_INPUT\_A3
- ADC12\_A\_INPUT\_A4
- ADC12\_A\_INPUT\_A5
- ADC12\_A\_INPUT\_A6
- ADC12\_A\_INPUT\_A7
- ADC12\_A\_INPUT\_A8
- ADC12\_A\_INPUT\_A9
- ADC12\_A\_INPUT\_TEMPSENSOR

- **ADC12\_A\_INPUT\_BATTERYMONITOR**
- **ADC12\_A\_INPUT\_A12**
- **ADC12\_A\_INPUT\_A13**
- **ADC12\_A\_INPUT\_A14**
- **ADC12\_A\_INPUT\_A15**

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

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

- **ADC12\_A\_VREFPOS\_AVCC** [Default]
- **ADC12\_A\_VREFPOS\_EXT**
- **ADC12\_A\_VREFPOS\_INT**

Modified bits are **ADC12SREF** of **ADC12MCTLx** register.

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

- **ADC12\_A\_VREFNEG\_AVSS** [Default]
- **ADC12\_A\_VREFNEG\_EXT**

Modified bits are **ADC12SREF** of **ADC12MCTLx** register.

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

- **ADC12\_A\_NOTENDOFSEQUENCE** [Default] - The specified memory buffer will NOT be the end of the sequence OR a sequenced conversion mode is not selected.
  - **ADC12\_A\_ENDOFSEQUENCE** - The specified memory buffer will be the end of the sequence.
- Modified bits are **ADC12EOS** of **ADC12MCTLx** register.

**Returns:**

None

## 6.2.2.16 ADC12\_A\_setDataReadBackFormat

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

**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	82
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

**Prototype:**

```
void
ADC12_A_setDataReadBackFormat (uint32_t baseAddress,
                               uint8_t readBackFormat)
```

**Description:**

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.

**Parameters:**

**baseAddress** is the base address of the ADC12\_A module.

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

- **ADC12\_A\_UNSIGNED\_BINARY** [Default]
- **ADC12\_A\_SIGNED\_2SCOMPLEMENT**  
Modified bits are **ADC12DF** of **ADC12CTL2** register.

**Returns:**  
None

### 6.2.2.17 ADC12\_A\_setReferenceBufferSamplingRate

Use to set the reference buffer's sampling rate.

**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	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	82
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

**Prototype:**

```
void
ADC12_A_setReferenceBufferSamplingRate(uint32_t baseAddress,
                                       uint8_t samplingRateSelect)
```

**Description:**

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

**Parameters:**

**baseAddress** is the base address of the ADC12\_A module.

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

- **ADC12\_A\_MAXSAMPLINGRATE\_200KSPS** [Default]
- **ADC12\_A\_MAXSAMPLINGRATE\_50KSPS**  
Modified bits are **ADC12SR** of **ADC12CTL2** register.

**Returns:**  
None

### 6.2.2.18 ADC12\_A\_setResolution

Use to change the resolution of the converted data.

**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	82
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

**Prototype:**

```
void
ADC12_A_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 ADC12\_A module.

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

- **ADC12\_A\_RESOLUTION\_8BIT**
- **ADC12\_A\_RESOLUTION\_10BIT**
- **ADC12\_A\_RESOLUTION\_12BIT** [Default]

Modified bits are **ADC12RESx** of **ADC12CTL2** register.

**Returns:**

None

### 6.2.2.19 ADC12\_A\_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	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

**Prototype:**

```
void
ADC12_A_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 ADC12\_A module.

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

- **ADC12\_A\_NONINVERTEDSIGNAL** [Default] - a sample-and-hold of an input signal for conversion will be started on a rising edge of the sample/hold signal.
- **ADC12\_A\_INVERTEDSIGNAL** - a sample-and-hold of an input signal for conversion will be started on a falling edge of the sample/hold signal.

Modified bits are **ADC12ISSH** of **ADC12CTL1** register.

**Returns:**

None

### 6.2.2.20 ADC12\_A\_setupSamplingTimer

Sets up and enables the Sampling Timer Pulse Mode.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	60
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	40
IAR 5.51.6	Size	30
IAR 5.51.6	Speed	36
MSPGCC 4.8.0	None	98
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

#### Prototype:

```
void
ADC12_A_setupSamplingTimer(uint32_t baseAddress,
                           uint16_t clockCycleHoldCountLowMem,
                           uint16_t clockCycleHoldCountHighMem,
                           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 ADC12\_A module.

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

- ADC12\_A\_CYCLEHOLD\_4\_CYCLES [Default]
- ADC12\_A\_CYCLEHOLD\_8\_CYCLES
- ADC12\_A\_CYCLEHOLD\_16\_CYCLES
- ADC12\_A\_CYCLEHOLD\_32\_CYCLES
- ADC12\_A\_CYCLEHOLD\_64\_CYCLES
- ADC12\_A\_CYCLEHOLD\_96\_CYCLES
- ADC12\_A\_CYCLEHOLD\_128\_CYCLES
- ADC12\_A\_CYCLEHOLD\_192\_CYCLES
- ADC12\_A\_CYCLEHOLD\_256\_CYCLES
- ADC12\_A\_CYCLEHOLD\_384\_CYCLES
- ADC12\_A\_CYCLEHOLD\_512\_CYCLES
- ADC12\_A\_CYCLEHOLD\_768\_CYCLES
- ADC12\_A\_CYCLEHOLD\_1024\_CYCLES

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

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

- ADC12\_A\_CYCLEHOLD\_4\_CYCLES [Default]
- ADC12\_A\_CYCLEHOLD\_8\_CYCLES
- ADC12\_A\_CYCLEHOLD\_16\_CYCLES
- ADC12\_A\_CYCLEHOLD\_32\_CYCLES
- ADC12\_A\_CYCLEHOLD\_64\_CYCLES
- ADC12\_A\_CYCLEHOLD\_96\_CYCLES
- ADC12\_A\_CYCLEHOLD\_128\_CYCLES
- ADC12\_A\_CYCLEHOLD\_192\_CYCLES
- ADC12\_A\_CYCLEHOLD\_256\_CYCLES



- **ADC12\_A\_CYCLEHOLD\_384\_CYCLES**
- **ADC12\_A\_CYCLEHOLD\_512\_CYCLES**
- **ADC12\_A\_CYCLEHOLD\_768\_CYCLES**
- **ADC12\_A\_CYCLEHOLD\_1024\_CYCLES**

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

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

- **ADC12\_A\_MULTIPLESAMPLESDISABLE** [Default] - a timer trigger will be needed to start every ADC conversion.
- **ADC12\_A\_MULTIPLESAMPLESENABLE** - during a sequenced and/or repeated conversion mode, after the first conversion, no sample/hold signal is necessary to start subsequent sample/hold and convert processes.

Modified bits are **ADC12MSC** of **ADC12CTL0** register.

**Returns:**

None

### 6.2.2.21 ADC12\_A\_startConversion

Enables/Starts an Analog-to-Digital Conversion.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	74
CCS 4.2.1	Size	34
CCS 4.2.1	Speed	34
IAR 5.51.6	None	58
IAR 5.51.6	Size	44
IAR 5.51.6	Speed	44
MSPGCC 4.8.0	None	178
MSPGCC 4.8.0	Size	58
MSPGCC 4.8.0	Speed	58

**Prototype:**

```
void
ADC12_A_startConversion(uint32_t baseAddress,
                        uint16_t startingMemoryBufferIndex,
                        uint8_t conversionSequenceModeSelect)
```

**Description:**

This function enables/starts the conversion process of the ADC. If the sample/hold signal source chosen during initialization was ADC12OSC, then the conversion is started immediately, otherwise the chosen sample/hold signal source starts the conversion by a rising edge of the signal. Keep in mind when selecting conversion modes, that for sequenced and/or repeated modes, to keep the sample/hold-and-convert process continuing without a trigger from the sample/hold signal source, the multiple samples must be enabled using the [ADC12\\_A\\_setupSamplingTimer\(\)](#) function. Note that after this function is called, the [ADC12\\_A\\_disableConversions\(\)](#) 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 ADC12\_A module.

**startingMemoryBufferIndex** is the memory buffer that will hold the first or only conversion. Valid values are:

- **ADC12\_A\_MEMORY\_0** [Default]
- **ADC12\_A\_MEMORY\_1**
- **ADC12\_A\_MEMORY\_2**
- **ADC12\_A\_MEMORY\_3**
- **ADC12\_A\_MEMORY\_4**
- **ADC12\_A\_MEMORY\_5**

- ADC12\_A\_MEMORY\_6
- ADC12\_A\_MEMORY\_7
- ADC12\_A\_MEMORY\_8
- ADC12\_A\_MEMORY\_9
- ADC12\_A\_MEMORY\_10
- ADC12\_A\_MEMORY\_11
- ADC12\_A\_MEMORY\_12
- ADC12\_A\_MEMORY\_13
- ADC12\_A\_MEMORY\_14
- ADC12\_A\_MEMORY\_15

Modified bits are **ADC12STARTADDx** of **ADC12CTL1** register.

**conversionSequenceModeSelect** determines the ADC operating mode. Valid values are:

- **ADC12\_A\_SINGLECHANNEL** [Default] - one-time conversion of a single channel into a single memory buffer.
  - **ADC12\_A\_SEQOFCHANNELS** - one time conversion of multiple channels into the specified starting memory buffer and each subsequent memory buffer up until the conversion is stored in a memory buffer dedicated as the end-of-sequence by the memory's control register.
  - **ADC12\_A\_REPEATED\_SINGLECHANNEL** - repeated conversions of one channel into a single memory buffer.
  - **ADC12\_A\_REPEATED\_SEQOFCHANNELS** - repeated conversions of multiple channels into the specified starting memory buffer and each subsequent memory buffer up until the conversion is stored in a memory buffer dedicated as the end-of-sequence by the memory's control register.
- Modified bits are **ADC12CONSEQx** of **ADC12CTL1** register.

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

**Returns:**  
None

## 6.3 Programming Example

The following example shows how to initialize and use the ADC12 API to start a single channel, single conversion.

```
// Initialize ADC12 with ADC12's built-in oscillator
ADC12_A_init (ADC12_A_BASE,
              ADC12_A_SAMPLEHOLDSOURCE_SC,
              ADC12_A_CLOCKSOURCE_ADC12OSC,
              ADC12_A_CLOCKDIVIDEBY_1);

//Switch ON ADC12
ADC12_A_enable(ADC12_A_BASE);

// Setup sampling timer to sample-and-hold for 16 clock cycles
ADC12_A_setupSamplingTimer (ADC12_A_BASE,
                            ADC12_A_CYCLEHOLD_64_CYCLES,
                            ADC12_A_CYCLEHOLD_4_CYCLES,
                            FALSE);

// Configure the Input to the Memory Buffer with the specified Reference Voltages
ADC12_A_memoryConfigure (ADC12_A_BASE,
                         ADC12_A_MEMORY_0,
                         ADC12_A_INPUT_A0,
                         ADC12_A_VREF_AVCC, // Vref+ = AVcc
                         ADC12_A_VREF_AVSS, // Vref- = AVss
                         FALSE
                         );

while (1)
{
    // Start a single conversion, no repeating or sequences.
```

```
ADC12_A_startConversion (ADC12_A_BASE,
                        ADC12_A_MEMORY_0,
                        ADC12_A_SINGLECHANNEL);

// Wait for the Interrupt Flag to assert
while( !(ADC12_A_getInterruptStatus(ADC12_A_BASE,ADC12IFG0)) );

// Clear the Interrupt Flag and start another conversion
ADC12_A_clearInterrupt (ADC12_A_BASE,ADC12IFG0);
}
```

# 7 Advanced Encryption Standard (AES)

Introduction .....	64
API Functions .....	64
Programming Example .....	75

## 7.1 Introduction

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

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

This driver is contained in `aes.c`, with `aes.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	1554
CCS 4.2.1	Size	622
CCS 4.2.1	Speed	2132
IAR 5.51.6	None	1106
IAR 5.51.6	Size	364
IAR 5.51.6	Speed	446
MSPGCC 4.8.0	None	1956
MSPGCC 4.8.0	Size	726
MSPGCC 4.8.0	Speed	2060

## 7.2 API Functions

### Functions

- void [AES\\_clearErrorFlag](#) (uint32\_t baseAddress)
- void [AES\\_clearInterruptFlag](#) (uint32\_t baseAddress)
- uint8\_t [AES\\_decryptData](#) (uint32\_t baseAddress, const uint8\_t \*Data, uint8\_t \*decryptedData)

- uint8\_t [AES\\_decryptDataUsingEncryptionKey](#) (uint32\_t baseAddress, const uint8\_t \*Data, uint8\_t \*decryptedData)
- void [AES\\_disableInterrupt](#) (uint32\_t baseAddress)
- void [AES\\_enableInterrupt](#) (uint32\_t baseAddress)
- uint8\_t [AES\\_encryptData](#) (uint32\_t baseAddress, const uint8\_t \*Data, uint8\_t \*encryptedData)
- uint8\_t [AES\\_getDataOut](#) (uint32\_t baseAddress, uint8\_t \*OutputData)
- uint32\_t [AES\\_getErrorFlagStatus](#) (uint32\_t baseAddress)
- uint32\_t [AES\\_getInterruptFlagStatus](#) (uint32\_t baseAddress)
- uint8\_t [AES\\_isBusy](#) (uint32\_t baseAddress)
- void [AES\\_reset](#) (uint32\_t baseAddress)
- uint8\_t [AES\\_setCipherKey](#) (uint32\_t baseAddress, const uint8\_t \*CipherKey)
- uint8\_t [AES\\_setDecipherKey](#) (uint32\_t baseAddress, const uint8\_t \*CipherKey)
- uint8\_t [AES\\_startDecryptData](#) (uint32\_t baseAddress, const uint8\_t \*Data)
- uint8\_t [AES\\_startDecryptDataUsingEncryptionKey](#) (uint32\_t baseAddress, const uint8\_t \*Data)
- uint8\_t [AES\\_startEncryptData](#) (uint32\_t baseAddress, const uint8\_t \*Data, uint8\_t \*encryptedData)
- uint8\_t [AES\\_startSetDecipherKey](#) (uint32\_t baseAddress, const uint8\_t \*CipherKey)

## 7.2.1 Detailed Description

The AES module APIs are

- [AES\\_setCipherKey\(\)](#)
- [AES\\_encryptData\(\)](#)
- [AES\\_decryptDataUsingEncryptionKey\(\)](#)
- [AES\\_generateFirstRoundKey\(\)](#)
- [AES\\_decryptData\(\)](#)
- [AES\\_reset\(\)](#)
- [AES\\_startEncryptData\(\)](#)
- [AES\\_startDecryptDataUsingEncryptionKey\(\)](#)
- [AES\\_startDecryptData\(\)](#)
- [AES\\_startGenerateFirstRoundKey\(\)](#)
- [AES\\_getDataOut\(\)](#)

The AES interrupt handler functions

- [AES\\_enableInterrupt\(\)](#)
- [AES\\_disableInterrupt\(\)](#)
- [AES\\_clearInterruptFlag\(\)](#)

## 7.2.2 Function Documentation

### 7.2.2.1 AES\_clearErrorFlag

Clears the AES error 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	6
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:**

```
void
AES_clearErrorFlag(uint32_t baseAddress)
```

**Description:**

Modified bit is AESERRFG of AESACTL0 register.

**Parameters:**

**baseAddress** is the base address of the AES module.

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

**Returns:**

None

### 7.2.2.2 AES\_clearInterruptFlag

Clears the AES ready 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	6
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:**

```
void
AES_clearInterruptFlag(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the AES module.

**Description:**

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

**Returns:**

None

### 7.2.2.3 AES\_decryptData

Decrypts a block of data using the AES module.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	194
CCS 4.2.1	Size	80
CCS 4.2.1	Speed	306
IAR 5.51.6	None	150
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	242
MSPGCC 4.8.0	Size	94
MSPGCC 4.8.0	Speed	290

**Prototype:**

```
uint8_t
AES_decryptData(uint32_t baseAddress,
               const uint8_t *Data,
               uint8_t *decryptedData)
```

**Description:**

This function requires a pre-generated decryption key. A key can be loaded and pre-generated by using function [AES\\_startSetDecipherKey\(\)](#) or [AES\\_setDecipherKey\(\)](#). The decryption takes 167 MCLK.

**Parameters:**

**baseAddress** is the base address of the AES module.

**Data** is a pointer to an uint8\_t array with a length of 16 bytes that contains encrypted data to be decrypted.

**decryptedData** is a pointer to an uint8\_t array with a length of 16 bytes in that the decrypted data will be written.

**Returns:**

STATUS\_SUCCESS

## 7.2.2.4 AES\_decryptDataUsingEncryptionKey

DEPRECATED Decrypts a block of data using the AES module.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	198
CCS 4.2.1	Size	82
CCS 4.2.1	Speed	308
IAR 5.51.6	None	156
IAR 5.51.6	Size	92
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	256
MSPGCC 4.8.0	Size	96
MSPGCC 4.8.0	Speed	292

**Prototype:**

```
uint8_t
AES_decryptDataUsingEncryptionKey(uint32_t baseAddress,
                                 const uint8_t *Data,
                                 uint8_t *decryptedData)
```

**Description:**

This function can be used to decrypt data by using the same key as used for a previous performed encryption. The decryption takes 214 MCLK.

**Parameters:**

**baseAddress** is the base address of the AES module.

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

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

**Returns:**

`STATUS_SUCCESS`

### 7.2.2.5 AES\_disableInterrupt

Disables AES ready 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	6
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:**

```
void
AES_disableInterrupt (uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the AES module.

**Description:**

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

**Returns:**

None

### 7.2.2.6 AES\_enableInterrupt

Enables AES ready 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	6
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:**

```
void
AES_enableInterrupt (uint32_t baseAddress)
```



**Description:**

Does not clear interrupt flags.

**Parameters:**

**baseAddress** is the base address of the AES module.

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

**Returns:**

None

### 7.2.2.7 AES\_encryptData

Encrypts a block of data using the AES module.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	194
CCS 4.2.1	Size	80
CCS 4.2.1	Speed	304
IAR 5.51.6	None	150
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	56
MSPGCC 4.8.0	None	242
MSPGCC 4.8.0	Size	94
MSPGCC 4.8.0	Speed	290

**Prototype:**

```
uint8_t
AES_encryptData(uint32_t baseAddress,
                const uint8_t *Data,
                uint8_t *encryptedData)
```

**Description:**

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

**Parameters:**

**baseAddress** is the base address of the AES module.

**Data** is a pointer to an uint8\_t array with a length of 16 bytes that contains data to be encrypted.

**encryptedData** is a pointer to an uint8\_t array with a length of 16 bytes in that the encrypted data will be written.

**Returns:**

STATUS\_SUCCESS

### 7.2.2.8 AES\_getDataOut

Reads back the output data from AES module.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	108
CCS 4.2.1	Size	42
CCS 4.2.1	Speed	146
IAR 5.51.6	None	84
IAR 5.51.6	Size	56
IAR 5.51.6	Speed	62
MSPGCC 4.8.0	None	132
MSPGCC 4.8.0	Size	52
MSPGCC 4.8.0	Speed	130

**Prototype:**

```
uint8_t
AES_getDataOut (uint32_t baseAddress,
                uint8_t *OutputData)
```

**Description:**

This function is meant to use after an encryption or decryption process that was started and finished by initiating an interrupt by use of the [AES\\_startEncryptData\(\)](#) or [AES\\_startDecryptData\(\)](#) functions.

**Parameters:**

**baseAddress** is the base address of the AES module.

**OutputData** is a pointer to an uint8\_t array with a length of 16 bytes in which the output data of the AES module is available. If AES module is busy returns NULL.

**Returns:**

STATUS\_SUCCESS if AES is not busy, STATUS\_FAIL if it is busy

### 7.2.2.9 AES\_getErrorFlagStatus

Gets the AES error flag status.

**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	8
IAR 5.51.6	Size	0
IAR 5.51.6	Speed	0
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint32_t
AES_getErrorFlagStatus (uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the AES module.

**Returns:**

One of the following:

- **AES\_ERROR\_OCCURRED**
- **AES\_NO\_ERROR**  
indicating if AESKEY or AESADIN were written while an AES operation was in progress

### 7.2.2.10 uint32\_t AES\_getInterruptFlagStatus (uint32\_t *baseAddress*)

Gets the AES ready interrupt flag 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	8
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
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 AES module.

**Returns:**

uint32\_t - AES\_READY\_INTERRUPT or 0x00.

### 7.2.2.11 uint8\_t AES\_isBusy (uint32\_t *baseAddress*)

Gets the AES module busy status.

**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 AES module.

**Returns:**

One of the following:

- **AES\_BUSY**
- **AES\_NOT\_BUSY**  
indicating if encryption/decryption/key generation is taking place

### 7.2.2.12 void AES\_reset (uint32\_t *baseAddress*)

Resets AES Module immediately.

**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	36
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

**Parameters:**

**baseAddress** is the base address of the AES module.

**Description:**

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

**Returns:**

None

### 7.2.2.13 AES\_setCipherKey

Loads a 128 bit cipher key to AES module.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	118
CCS 4.2.1	Size	48
CCS 4.2.1	Speed	172
IAR 5.51.6	None	90
IAR 5.51.6	Size	26
IAR 5.51.6	Speed	74
MSPGCC 4.8.0	None	132
MSPGCC 4.8.0	Size	58
MSPGCC 4.8.0	Speed	170

**Prototype:**

```
uint8_t
AES_setCipherKey(uint32_t baseAddress,
                 const uint8_t *CipherKey)
```

**Parameters:**

**baseAddress** is the base address of the AES module.

**CipherKey** is a pointer to an uint8\_t array with a length of 16 bytes that contains a 128 bit cipher key.

**Returns:**

STATUS\_SUCCESS

### 7.2.2.14 uint8\_t AES\_setDecipherKey (uint32\_t baseAddress, const uint8\_t \* CipherKey)

Sets the decipher key The API.

The API [AES\\_startSetDecipherKey\(\)](#) or [AES\\_setDecipherKey\(\)](#) must be invoked before invoking [AES\\_setDecipherKey\(\)](#).

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	118
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	172
IAR 5.51.6	None	90
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	144
MSPGCC 4.8.0	Size	60
MSPGCC 4.8.0	Speed	172

**Parameters:**

**baseAddress** is the base address of the AES module.

**CipherKey** is a pointer to an uint8\_t array with a length of 16 bytes that contains the initial AES key.

**Returns:**

STATUS\_SUCCESS

### 7.2.2.15 uint8\_t AES\_startDecryptData (uint32\_t baseAddress, const uint8\_t \* Data)

Decrypts a block of data using the AES module.

This is the non-blocking equivalent of [AES\\_decryptData\(\)](#). This function requires a pre-generated decryption key. A key can be loaded and pre-generated by using function [AES\\_setDecipherKey\(\)](#) or [AES\\_startSetDecipherKey\(\)](#). The decryption takes 167 MCLK. It is recommended to use interrupt to check for procedure completion then using [AES\\_getDataOut\(\)](#) API to retrieve the decrypted data.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	110
CCS 4.2.1	Size	46
CCS 4.2.1	Speed	170
IAR 5.51.6	None	80
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	132
MSPGCC 4.8.0	Size	52
MSPGCC 4.8.0	Speed	164

**Parameters:**

**baseAddress** is the base address of the AES module.

**Data** is a pointer to an uint8\_t array with a length of 16 bytes that contains encrypted data to be decrypted.

**Returns:**

STATUS\_SUCCESS

### 7.2.2.16 uint8\_t AES\_startDecryptDataUsingEncryptionKey (uint32\_t baseAddress, const uint8\_t \* Data)

DEPRECATED Starts an decryption process on the AES module.

This is the non-blocking equivalent of [AES\\_decryptDataUsingEncryptionKey\(\)](#). This function can be used to decrypt data by using the same key as used for a previous performed encryption. The decryption takes 214 MCLK.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	114
CCS 4.2.1	Size	48
CCS 4.2.1	Speed	172
IAR 5.51.6	None	86
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	146
MSPGCC 4.8.0	Size	54
MSPGCC 4.8.0	Speed	166

**Parameters:**

**baseAddress** is the base address of the AES module.

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

**Returns:**

STATUS\_SUCCESS

### 7.2.2.17 `uint8_t AES_startEncryptData (uint32_t baseAddress, const uint8_t * Data, uint8_t * encryptedData)`

Starts an encryption process on the AES module.

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

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	114
CCS 4.2.1	Size	46
CCS 4.2.1	Speed	166
IAR 5.51.6	None	82
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	56
MSPGCC 4.8.0	None	140
MSPGCC 4.8.0	Size	52
MSPGCC 4.8.0	Speed	164

**Parameters:**

**baseAddress** is the base address of the AES module.

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

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

**Returns:**

STATUS\_SUCCESS

### 7.2.2.18 `uint8_t AES_startSetDecipherKey (uint32_t baseAddress, const uint8_t * CipherKey)`

Loads the decipher key.

This is the non-blocking equivalent of [AES\\_setDecipherKey\(\)](#). The API [AES\\_startSetDecipherKey\(\)](#) or [AES\\_setDecipherKey\(\)](#) must be invoked before invoking [AES\\_startSetDecipherKey\(\)](#).

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	106
CCS 4.2.1	Size	44
CCS 4.2.1	Speed	160
IAR 5.51.6	None	80
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	62
MSPGCC 4.8.0	None	128
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	158

**Parameters:**

**baseAddress** is the base address of the AES module.

**CipherKey** is a pointer to an uint8\_t array with a length of 16 bytes that contains the initial AES key.

**Returns:**

STATUS\_SUCCESS

## 7.3 Programming Example

The following example shows some AES operations using the APIs

```

unsigned char Data[16] = { 0x30, 0x31, 0x32, 0x33,
                           0x34, 0x35, 0x36, 0x37,
                           0x38, 0x39, 0x0A, 0x0B,
                           0x0C, 0x0D, 0x0E, 0x0F };

unsigned char CipherKey[16] = { 0xAA, 0xBB, 0x02, 0x03,
                                0x04, 0x05, 0x06, 0x07,
                                0x08, 0x09, 0x0A, 0x0B,
                                0x0C, 0x0D, 0x0E, 0x0F };

unsigned char DataAES[16];           // Encrypted data
unsigned char DataunAES[16];         // Decrypted data

// Load a cipher key to module
AES_setCipherKey(AES_BASE, CipherKey);

// Encrypt data with preloaded cipher key
AES_encryptData(AES_BASE, Data, DataAES);

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

```

## 8 Battery Backup System

Introduction .....	76
API Functions .....	76
Programming Example .....	80

### 8.1 Introduction

The Battery Backup System (BATBCK) API provides a set of functions for using the MSP430Ware BATBCK modules. Functions are provided to handle the backup Battery sub-system, initialize and enable the backup Battery charger, and control access to and from the backup RAM space.

The BATBCK module offers no interrupt, and is used only to control the Battery backup sub-system, Battery charger, and backup RAM space.

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

### 8.2 API Functions

#### Functions

- void [BAK\\_BATT\\_chargerInitAndEnable](#) (uint32\_t baseAddress, uint8\_t chargerEndVoltage, uint8\_t chargeCurrent)
- void [BAK\\_BATT\\_disable](#) (uint32\_t baseAddress)
- void [BAK\\_BATT\\_disableBackupSupplyToADC](#) (uint32\_t baseAddress)
- void [BAK\\_BATT\\_disableCharger](#) (uint32\_t baseAddress)
- void [BAK\\_BATT\\_enableBackupSupplyToADC](#) (uint32\_t baseAddress)
- uint16\_t [BAK\\_BATT\\_getBackupRAMData](#) (uint32\_t baseAddress, uint8\_t backupRAMSelect)
- void [BAK\\_BATT\\_manuallySwitchToBackupSupply](#) (uint32\_t baseAddress)
- void [BAK\\_BATT\\_setBackupRAMData](#) (uint32\_t baseAddress, uint8\_t backupRAMSelect, uint16\_t data)
- uint16\_t [BAK\\_BATT\\_unlockBackupSubSystem](#) (uint32\_t baseAddress)

#### 8.2.1 Detailed Description

The BATBCK API is divided into three groups: one that handles the Battery backup sub-system, one that controls the charger, and one that controls access to and from the backup RAM space.

The BATBCK sub-system controls are handled by

- [BAK\\_BATT\\_unlockBackupSubSystem\(\)](#)
- [BAK\\_BATT\\_enableBackupSupplyToADC\(\)](#)
- [BAK\\_BATT\\_disableBackupSupplyToADC\(\)](#)
- [BAK\\_BATT\\_manuallySwitchToBackupSupply\(\)](#)
- [BAK\\_BATT\\_disable\(\)](#)

The BATBCK charger is controlled by

- [BAK\\_BATT\\_chargerInitAndEnable\(\)](#)
- [BAK\\_BATT\\_disableCharger\(\)](#)

The backup RAM space is accessed by

- [BAK\\_BATT\\_setBackupRAMData\(\)](#)
- [BAK\\_BATT\\_getBackupRAMData\(\)](#)



## 8.2.2 Function Documentation

### 8.2.2.1 BAK\_BATT\_chargerInitAndEnable

Initializes and enables the backup battery charger.

**Prototype:**

```
void
BAK_BATT_chargerInitAndEnable(uint32_t baseAddress,
                               uint8_t chargerEndVoltage,
                               uint8_t chargeCurrent)
```

**Description:**

This function initializes the backup battery charger with the selected settings.

**Parameters:**

**baseAddress** is the base address of the BAK\_BATT module.

**chargerEndVoltage** is the maximum voltage to charge the backup battery to. Valid values are:

- **BAK\_BATT\_CHARGERENDVOLTAGE\_VCC** - charges backup battery up to Vcc
- **BAK\_BATT\_CHARGERENDVOLTAGE2\_7V** - charges backup battery up to 2.7V OR up to Vcc if Vcc is less than 2.7V.

Modified bits are **BAKCHVx** of **BAKCHCTL** register.

**chargeCurrent** is the maximum current to charge the backup battery at. Valid values are:

- **BAK\_BATT\_CHARGECURRENT\_5KOHM**
- **BAK\_BATT\_CHARGECURRENT\_10KOHM**
- **BAK\_BATT\_CHARGECURRENT\_20KOHM**

Modified bits are **BAKCHCx** of **BAKCHCTL** register.

**Returns:**

None

### 8.2.2.2 BAK\_BATT\_disable

Disables backup battery system.

**Prototype:**

```
void
BAK_BATT_disable(uint32_t baseAddress)
```

**Description:**

This function disables the battery backup system from being used. The battery backup system is re-enabled after a power cycle.

**Parameters:**

**baseAddress** is the base address of the BAK\_BATT module.

**Returns:**

None

### 8.2.2.3 BAK\_BATT\_disableBackupSupplyToADC

Disables the backup supply input to the ADC module.

**Prototype:**

```
void
BAK_BATT_disableBackupSupplyToADC(uint32_t baseAddress)
```

**Description:**

This function disables the ability to monitor the backup supply voltage from the ADC battery monitor input.

**Parameters:**

**baseAddress** is the base address of the BAK\_BATT module.

**Returns:**

None

#### 8.2.2.4 BAK\_BATT\_disableCharger

Disables and resets backup battery charger settings.

**Prototype:**

```
void  
BAK_BATT_disableCharger(uint32_t baseAddress)
```

**Description:**

This function clears all backup battery charger settings and disables it. To re-enable the charger, a call to [BAK\\_BATT\\_chargerInitAndEnable\(\)](#) is required.

**Parameters:**

**baseAddress** is the base address of the BAK\_BATT module.

**Returns:**

None

#### 8.2.2.5 BAK\_BATT\_enableBackupSupplyToADC

Enables the backup supply to be measured by the ADC battery monitor input.

**Prototype:**

```
void  
BAK_BATT_enableBackupSupplyToADC(uint32_t baseAddress)
```

**Description:**

This function enables the backup supply signal to be monitored by the ADC battery supply monitor input, to allow a measurement of the voltage from the backup battery.

**Parameters:**

**baseAddress** is the base address of the BAK\_BATT module.

**Returns:**

None

#### 8.2.2.6 BAK\_BATT\_getBackupRAMData

Returns the data from the selected backup RAM space.

**Prototype:**

```
uint16_t  
BAK_BATT_getBackupRAMData(uint32_t baseAddress,  
                           uint8_t backupRAMSelect)
```

**Description:**

This function returns the 16-bit data currently residing in the selected backup RAM space.

**Parameters:**

**baseAddress** is the base address of the BAK\_BATT module.

**backupRAMSelect** is the backup RAM space to read out from. Valid values are:

- BAK\_BATT\_RAMSELECT\_0
- BAK\_BATT\_RAMSELECT\_1
- BAK\_BATT\_RAMSELECT\_2
- BAK\_BATT\_RAMSELECT\_3

**Returns:**

Data residing in the selected backup RAM space.

### 8.2.2.7 BAK\_BATT\_manuallySwitchToBackupSupply

Manually switches to backup supply.

**Prototype:**

```
void  
BAK_BATT_manuallySwitchToBackupSupply(uint32_t baseAddress)
```

**Description:**

This function uses software to manually switch to the backup battery supply. Once this bit is set, it will be automatically reset by a POR and the system returns to an automatic switch to backup supply.

**Parameters:**

**baseAddress** is the base address of the BAK\_BATT module.

**Returns:**

None

### 8.2.2.8 BAK\_BATT\_setBackupRAMData

Sets data into the selected backup RAM space.

**Prototype:**

```
void  
BAK_BATT_setBackupRAMData(uint32_t baseAddress,  
                           uint8_t backupRAMSelect,  
                           uint16_t data)
```

**Description:**

This function sets the given 16-bit data into the selected backup RAM space.

**Parameters:**

**baseAddress** is the base address of the BAK\_BATT module.

**backupRAMSelect** is the backup RAM space to set data into. Valid values are:

- BAK\_BATT\_RAMSELECT\_0
- BAK\_BATT\_RAMSELECT\_1
- BAK\_BATT\_RAMSELECT\_2
- BAK\_BATT\_RAMSELECT\_3

**data** is the data to set into the selected backup RAM space.

**Returns:**

None

### 8.2.2.9 BAK\_BATT\_unlockBackupSubSystem

Unlocks any pending backup input pins and RTC\_B interrupts to be serviced.

**Prototype:**

```
uint16_t  
BAK_BATT_unlockBackupSubSystem(uint32_t baseAddress)
```

**Description:**

This function unlocks the ability to view and service any pending backup input pin interrupts, as well as pending RTC\_B interrupts. The backup sub- system can only be unlocked when the backup domain has settled, so this function returns the status of the unlock bit after it has been to be verified by user code. Please note, the backup sub-system should only be unlocked after modifying the RTC\_B registers.

**Parameters:**

***baseAddress*** is the base address of the BAK\_BATT module.

**Returns:**

One of the following:

- **BAK\_BATT\_UNLOCKFAILURE** backup system has not yet settled
- **BAK\_BATT\_UNLOCKSUCCESS** successfully unlocked  
indicating if the backup system has been successfully unlocked

## 8.3 Programming Example

## 9 Comparator (COMP\_B)

Introduction .....	81
API Functions .....	82
Programming Example .....	96

### 9.1 Introduction

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

The COMP\_B 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\_B 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\_B module also has control over the REF module to generate a reference voltage as an input.

The COMP\_B 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_b.c`, with `comp_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	752
CCS 4.2.1	Size	348
CCS 4.2.1	Speed	348
IAR 5.51.6	None	500
IAR 5.51.6	Size	362
IAR 5.51.6	Speed	390
MSPGCC 4.8.0	None	1134
MSPGCC 4.8.0	Size	422
MSPGCC 4.8.0	Speed	502

## 9.2 API Functions

### Functions

- void [COMP\\_B\\_clearInterrupt](#) (uint32\_t baseAddress, uint16\_t interruptFlagMask)
- void [COMP\\_B\\_disable](#) (uint32\_t baseAddress)
- void [COMP\\_B\\_disableInputBuffer](#) (uint32\_t baseAddress, uint8\_t inputPort)
- void [COMP\\_B\\_disableInterrupt](#) (uint32\_t baseAddress, uint16\_t interruptMask)
- void [COMP\\_B\\_enable](#) (uint32\_t baseAddress)
- void [COMP\\_B\\_enableInputBuffer](#) (uint32\_t baseAddress, uint8\_t inputPort)
- void [COMP\\_B\\_enableInterrupt](#) (uint32\_t baseAddress, uint16\_t interruptMask)
- uint8\_t [COMP\\_B\\_getInterruptStatus](#) (uint32\_t baseAddress, uint16\_t interruptFlagMask)
- bool [COMP\\_B\\_init](#) (uint32\_t baseAddress, uint8\_t positiveTerminalInput, uint8\_t negativeTerminalInput, uint16\_t powerModeSelect, uint8\_t outputFilterEnableAndDelayLevel, uint16\_t invertedOutputPolarity)
- void [COMP\\_B\\_interruptSetEdgeDirection](#) (uint32\_t baseAddress, uint16\_t edgeDirection)
- void [COMP\\_B\\_interruptToggleEdgeDirection](#) (uint32\_t baseAddress)
- void [COMP\\_B\\_IOSwap](#) (uint32\_t baseAddress)
- uint16\_t [COMP\\_B\\_outputValue](#) (uint32\_t baseAddress)
- void [COMP\\_B\\_setReferenceVoltage](#) (uint32\_t baseAddress, uint16\_t supplyVoltageReferenceBase, uint16\_t lowerLimitSupplyVoltageFractionOf32, uint16\_t upperLimitSupplyVoltageFractionOf32, uint16\_t referenceAccuracy)
- void [COMP\\_B\\_shortInputs](#) (uint32\_t baseAddress)
- void [COMP\\_B\\_unshortInputs](#) (uint32\_t baseAddress)

### 9.2.1 Detailed Description

The COMP\_B 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\_B.

The COMP\_B initialization and output functions are

- [COMP\\_B\\_init\(\)](#)
- [COMP\\_B\\_setReferenceVoltage\(\)](#)
- [COMP\\_B\\_enable\(\)](#)
- [COMP\\_B\\_disable\(\)](#)
- [COMP\\_B\\_outputValue\(\)](#)

The COMP\_B interrupts are handled by

- [COMP\\_B\\_enableInterrupt\(\)](#)
- [COMP\\_B\\_disableInterrupt\(\)](#)
- [COMP\\_B\\_clearInterrupt\(\)](#)
- [COMP\\_B\\_getInterruptStatus\(\)](#)
- [COMP\\_B\\_interruptSetEdgeDirection\(\)](#)
- [COMP\\_B\\_interruptToggleEdgeDirection\(\)](#)

Auxiliary features of the COMP\_B are handled by

- COMP\_B\_enableShortOfInputs()
- COMP\_B\_disableShortOfInputs()
- COMP\_B\_disableInputBuffer()
- COMP\_B\_enableInputBuffer()
- COMP\_B\_IOSwap()

## 9.2.2 Function Documentation

### 9.2.2.1 COMP\_B\_clearInterrupt

Clears COMP\_B 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_B_clearInterrupt (uint32_t baseAddress,
                      uint16_t interruptFlagMask)
```

#### Description:

The COMP\_B 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\_B module.

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

- **COMP\_B\_OUTPUT\_FLAG** - Output interrupt
  - **COMP\_B\_OUTPUTINVERTED\_FLAG** - Output interrupt inverted polarity
- Modified bits of **CBINT** register.

#### Returns:

None

### 9.2.2.2 COMP\_B\_disable

Turns off the COMP\_B 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_B_disable(uint32_t baseAddress)
```

#### Description:

This function clears the CBON bit disabling the operation of the COMP\_B module, saving from excess power consumption.

#### Parameters:

**baseAddress** is the base address of the COMP\_B module.

#### Returns:

None

### 9.2.2.3 COMP\_B\_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_B_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 COMP\_B input pins. This bit is automatically set when the input is initialized to be used with the COMP\_B 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\_B module.

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

- COMP\_B\_INPUT0 [Default]
- COMP\_B\_INPUT1
- COMP\_B\_INPUT2
- COMP\_B\_INPUT3
- COMP\_B\_INPUT4
- COMP\_B\_INPUT5
- COMP\_B\_INPUT6
- COMP\_B\_INPUT7
- COMP\_B\_INPUT8
- COMP\_B\_INPUT9
- COMP\_B\_INPUT10
- COMP\_B\_INPUT11
- COMP\_B\_INPUT12
- COMP\_B\_INPUT13
- COMP\_B\_INPUT14
- COMP\_B\_INPUT15
- COMP\_B\_VREF

Modified bits are CBPDx of CBCTL3 register.

**Returns:**

None

#### 9.2.2.4 COMP\_B\_disableInterrupt

Disables selected COMP\_B 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_B_disableInterrupt (uint32_t baseAddress,
                        uint16_t interruptMask)
```

**Description:**

Disables the indicated COMP\_B 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\_B module.

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

- **COMP\_B\_OUTPUT\_INT** - Output interrupt
- **COMP\_B\_OUTPUTINVERTED\_INT** - Output interrupt inverted polarity  
Modified bits of **CBINT** register.

**Returns:**

None

### 9.2.2.5 COMP\_B\_enable

Turns on the COMP\_B 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_B_enable(uint32_t baseAddress)
```

**Description:**

This function sets the bit that enables the operation of the COMP\_B module.

**Parameters:**

**baseAddress** is the base address of the COMP\_B module.

**Returns:**

None

### 9.2.2.6 COMP\_B\_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_B_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 COMP\_B 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\_B module.

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

- COMP\_B\_INPUT0 [Default]
- COMP\_B\_INPUT1
- COMP\_B\_INPUT2
- COMP\_B\_INPUT3
- COMP\_B\_INPUT4
- COMP\_B\_INPUT5
- COMP\_B\_INPUT6

- COMP\_B\_INPUT7
- COMP\_B\_INPUT8
- COMP\_B\_INPUT9
- COMP\_B\_INPUT10
- COMP\_B\_INPUT11
- COMP\_B\_INPUT12
- COMP\_B\_INPUT13
- COMP\_B\_INPUT14
- COMP\_B\_INPUT15
- COMP\_B\_VREF

Modified bits are **CBPDx** of **CBCTL3** register.

**Returns:**

None

### 9.2.2.7 COMP\_B\_enableInterrupt

Enables selected COMP\_B 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	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

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

**Description:**

Enables the indicated COMP\_B 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\_B module.

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

- **COMP\_B\_OUTPUT\_INT** - Output interrupt
  - **COMP\_B\_OUTPUTINVERTED\_INT** - Output interrupt inverted polarity
- Modified bits of **CBINT** register.

**Returns:**

None

## 9.2.2.8 COMP\_B\_getInterruptStatus

Gets the current COMP\_B 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_B_getInterruptStatus(uint32_t baseAddress,
                          uint16_t interruptFlagMask)
```

**Parameters:****baseAddress** is the base address of the COMP\_B module.**interruptFlagMask** is the masked interrupt flag status to be returned. Mask value is the logical OR of any of the following:

- **COMP\_B\_OUTPUT\_FLAG** - Output interrupt
- **COMP\_B\_OUTPUTINVERTED\_FLAG** - Output interrupt inverted polarity

**Returns:**

Logical OR of any of the following:

- **COMP\_B\_OUTPUT\_FLAG** Output interrupt
- **COMP\_B\_OUTPUTINVERTED\_FLAG** Output interrupt inverted polarity indicating the status of the masked interrupts

9.2.2.9 bool COMP\_B\_init (uint32\_t *baseAddress*, uint8\_t *positiveTerminalInput*, uint8\_t *negativeTerminalInput*, uint16\_t *powerModeSelect*, uint8\_t *outputFilterEnableAndDelayLevel*, uint16\_t *invertedOutputPolarity*)

Initializes the COMP\_B Module.

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

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	180
CCS 4.2.1	Size	124
CCS 4.2.1	Speed	124
IAR 5.51.6	None	170
IAR 5.51.6	Size	142
IAR 5.51.6	Speed	150
MSPGCC 4.8.0	None	284
MSPGCC 4.8.0	Size	158
MSPGCC 4.8.0	Speed	180

**Parameters:**

**baseAddress** is the base address of the COMP\_B module.

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

- COMP\_B\_INPUT0 [Default]
- COMP\_B\_INPUT1
- COMP\_B\_INPUT2
- COMP\_B\_INPUT3
- COMP\_B\_INPUT4
- COMP\_B\_INPUT5
- COMP\_B\_INPUT6
- COMP\_B\_INPUT7
- COMP\_B\_INPUT8
- COMP\_B\_INPUT9
- COMP\_B\_INPUT10
- COMP\_B\_INPUT11
- COMP\_B\_INPUT12
- COMP\_B\_INPUT13
- COMP\_B\_INPUT14
- COMP\_B\_INPUT15
- COMP\_B\_VREF

Modified bits are **CBIPEN** and **CBIPSEL** of **CBCTL0** register; bits **CBRSEL** of **CBCTL2** register; bits **CBPDx** of **CBCTL3** register.

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

- COMP\_B\_INPUT0 [Default]
- COMP\_B\_INPUT1
- COMP\_B\_INPUT2
- COMP\_B\_INPUT3
- COMP\_B\_INPUT4
- COMP\_B\_INPUT5
- COMP\_B\_INPUT6
- COMP\_B\_INPUT7
- COMP\_B\_INPUT8
- COMP\_B\_INPUT9
- COMP\_B\_INPUT10

- COMP\_B\_INPUT11
- COMP\_B\_INPUT12
- COMP\_B\_INPUT13
- COMP\_B\_INPUT14
- COMP\_B\_INPUT15
- COMP\_B\_VREF

Modified bits are **CBIMEN** and **CBIMSEL** of **CBCTL0** register; bits **CBRSEL** of **CBCTL2** register; bits **CBPDx** of **CBCTL3** register.

**powerModeSelect** selects the power mode at which the COMP\_B module will operate at.

Valid values are:

- COMP\_B\_POWERMODE\_HIGHSPEED [Default]
- COMP\_B\_POWERMODE\_NORMALMODE
- COMP\_B\_POWERMODE\_ULTRALOWPOWER

Modified bits are **CBWRMD** of **CBCTL1** 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\_B\_FILTEROUTPUT\_OFF [Default]
- COMP\_B\_FILTEROUTPUT\_DLYLVL1
- COMP\_B\_FILTEROUTPUT\_DLYLVL2
- COMP\_B\_FILTEROUTPUT\_DLYLVL3
- COMP\_B\_FILTEROUTPUT\_DLYLVL4

Modified bits are **CBFDLY** and **CBF** of **CBCTL1** register.

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

- COMP\_B\_NORMALOUTPUTPOLARITY [Default]
- COMP\_B\_INVERTEDOUTPUTPOLARITY

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of the initialization process.

9.2.2.10 void COMP\_B\_interruptSetEdgeDirection (uint32\_t *baseAddress*, uint16\_t *edgeDirection*)

Explicitly sets the edge direction that would trigger an interrupt.

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

**Code Metrics:**

Compiler	Optimization	Code Size
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

**Parameters:**

**baseAddress** is the base address of the COMP\_B 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\_B\_FALLINGEDGE** [Default] - sets the bit to generate an interrupt when the output of the COMP\_B 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\_B\_RISINGEDGE** - sets the bit to generate an interrupt when the output of the COMP\_B 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 **CBIES** of **CBCTL1** register.

**Returns:**

None

#### 9.2.2.11 void COMP\_B\_interruptToggleEdgeDirection (uint32\_t baseAddress)

Toggles the edge direction that would trigger an interrupt.

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

**Code Metrics:**

Compiler	Optimization	Code Size
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

**Parameters:**

**baseAddress** is the base address of the COMP\_B module.



**Returns:**

None

**9.2.2.12 void COMP\_B\_IOSwap (uint32\_t *baseAddress*)**

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

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

**Code Metrics:**

Compiler	Optimization	Code Size
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

**Parameters:**

***baseAddress*** is the base address of the COMP\_B module.

**Returns:**

None

**9.2.2.13 uint16\_t COMP\_B\_outputValue (uint32\_t *baseAddress*)**

Returns the output value of the COMP\_B module.

Returns the output value of the COMP\_B 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

**Parameters:**

**baseAddress** is the base address of the COMP\_B module.

**Returns:**

One of the following:

- **COMP\_B\_LOW**
  - **COMP\_B\_HIGH**
- indicating the output value of the COMP\_B module

9.2.2.14 void COMP\_B\_setReferenceVoltage (uint32\_t baseAddress, uint16\_t supplyVoltageReferenceBase, uint16\_t lowerLimitSupplyVoltageFractionOf32, uint16\_t upperLimitSupplyVoltageFractionOf32, uint16\_t referenceAccuracy)

Generates a Reference Voltage to the terminal selected during initialization.

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

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	138
CCS 4.2.1	Size	90
CCS 4.2.1	Speed	90
IAR 5.51.6	None	122
IAR 5.51.6	Size	90
IAR 5.51.6	Speed	88
MSPGCC 4.8.0	None	236
MSPGCC 4.8.0	Size	84
MSPGCC 4.8.0	Speed	144

**Parameters:**

**baseAddress** is the base address of the COMP\_B module.

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

- **COMP\_B\_VREFBASE\_VCC**
- **COMP\_B\_VREFBASE1\_5V**
- **COMP\_B\_VREFBASE2\_0V**
- **COMP\_B\_VREFBASE2\_5V**

Modified bits are **CBREFL** of **CBCTL2** register.

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

Modified bits are **CBREF0** of **CBCTL2** register.

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

Modified bits are **CBREF1** of **CBCTL2** register.

**referenceAccuracy** is the reference accuracy setting of the COMP\_B. Clocked is for low power/low accuracy. Valid values are:

- **COMP\_B\_ACCURACY\_STATIC**
  - **COMP\_B\_ACCURACY\_CLOCKED**
- Modified bits are **CDREFACC** of **CDCTL2** register.

**Returns:**

None

#### 9.2.2.15 void COMP\_B\_shortInputs (uint32\_t *baseAddress*)

Shorts the two input pins chosen during initialization.

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

**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

**Parameters:**

***baseAddress*** is the base address of the COMP\_B module.

**Returns:**

None

#### 9.2.2.16 void COMP\_B\_unshortInputs (uint32\_t *baseAddress*)

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

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

**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

**Parameters:**

**baseAddress** is the base address of the COMP\_B module.

**Returns:**

None

## 9.3 Programming Example

The following example shows how to initialize and use the COMP\_B 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 B module
/* Base Address of Comparator B,
   Pin CB0 to Positive(+) Terminal,
   Reference Voltage to Negative(-) Terminal,
   Normal Power Mode,
   Output Filter On with minimal delay,
   Non-Inverted Output Polarity
*/
COMP_B_init(COMP_B_BASE,
            COMP_B_INPUT0,
            COMP_B_VREF,
            COMP_B_POWERMODE_NORMALMODE,
            COMP_B_FILTEROUTPUT_DLYLVL1,
            COMP_B_NORMALOUTPUTPOLARITY
            );

// Set the reference voltage that is being supplied to the (-) terminal
/* Base Address of Comparator B,
   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_B_setReferenceVoltage(COMP_B_BASE,
                           COMP_B_VREFBASE2_5V,
                           32,
                           32
                           );

// Allow power to Comparator module
COMP_B_enable(COMP_B_BASE);

// delay for the reference to settle
__delay_cycles(75);
```

# 10 Cyclical Redundancy Check (CRC)

Introduction .....	97
API Functions .....	97
Programming Example .....	102

## 10.1 Introduction

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

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

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

T

he following code metrics were performed with the 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

## 10.2 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)`

### 10.2.1 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()

## 10.2.2 Function Documentation

### 10.2.2.1 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

### 10.2.2.2 CRC\_getResult

Returns the value 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	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

### 10.2.2.3 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

### 10.2.2.4 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

### 10.2.2.5 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

### 10.2.2.6 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

### 10.2.2.7 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

## 10.3 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);
```

# 11 12-bit Digital-to-Analog Converter (DAC12\_A)

Introduction .....	103
API Functions .....	103
Programming Example .....	113

## 11.1 Introduction

The 12-Bit Digital-to-Analog (DAC12\_A) API provides a set of functions for using the MSP430Ware DAC12\_A modules. Functions are provided to initialize setup the DAC12\_A modules, calibrate the output signal, and manage the interrupts for the DAC12\_A modules.

The DAC12\_A module provides the ability to convert digital values into an analog signal for output to a pin. The DAC12\_A can generate signals from 0 to Vcc from an 8- or 12-bit value. There can be one or two DAC12\_A modules in a device, and if there are two they can be grouped together to create two analog signals in simultaneously. There are 3 ways to latch data in to the DAC module, and those are by software with the startConversion API function call, as well as by the Timer A output of CCR1 or Timer B output of CCR2.

The calibration API will unlock and start calibration, then wait for the calibration to end before locking it back up, all in one API. There are also functions to read out the calibration data, as well as be able to set it manually.

The DAC12\_A module can generate one interrupt for each DAC module. It will generate the interrupt when the data has been latched into the DAC module to be output into an analog signal.

This driver is contained in `dac12_a.c`, with `dac12_a.h` containing the API definitions for use by applications.

## 11.2 API Functions

### Functions

- void [DAC12\\_A\\_calibrateOutput](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- void [DAC12\\_A\\_clearInterrupt](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- void [DAC12\\_A\\_disable](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- void [DAC12\\_A\\_disableConversions](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- void [DAC12\\_A\\_disableGrouping](#) (uint32\_t baseAddress)
- void [DAC12\\_A\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- void [DAC12\\_A\\_enableConversions](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- void [DAC12\\_A\\_enableGrouping](#) (uint32\_t baseAddress)
- void [DAC12\\_A\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- uint16\_t [DAC12\\_A\\_getCalibrationData](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- uint32\_t [DAC12\\_A\\_getDataBufferMemoryAddressForDMA](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- uint16\_t [DAC12\\_A\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t submoduleSelect)
- bool [DAC12\\_A\\_init](#) (uint32\_t baseAddress, uint8\_t submoduleSelect, uint16\_t outputSelect, uint16\_t positiveReferenceVoltage, uint16\_t outputVoltageMultiplier, uint8\_t amplifierSetting, uint16\_t conversionTriggerSelect)
- void [DAC12\\_A\\_setAmplifierSetting](#) (uint32\_t baseAddress, uint8\_t submoduleSelect, uint8\_t amplifierSetting)
- void [DAC12\\_A\\_setCalibrationOffset](#) (uint32\_t baseAddress, uint8\_t submoduleSelect, uint16\_t calibrationOffsetValue)
- void [DAC12\\_A setData](#) (uint32\_t baseAddress, uint8\_t submoduleSelect, uint16\_t data)
- void [DAC12\\_A setInputDataFormat](#) (uint32\_t baseAddress, uint8\_t submoduleSelect, uint8\_t inputJustification, uint8\_t inputSign)
- void [DAC12\\_A\\_setResolution](#) (uint32\_t baseAddress, uint8\_t submoduleSelect, uint16\_t resolutionSelect)

## 11.2.1 Detailed Description

The DAC12\_A API is broken into three groups of functions: those that deal with initialization and conversions, those that deal with calibration of the output, and those that handle interrupts.

The DAC12\_A initialization and conversion functions are

- [DAC12\\_A\\_init\(\)](#)
- [DAC12\\_A\\_setAmplifierSetting\(\)](#)
- [DAC12\\_A\\_disable\(\)](#)
- [DAC12\\_A\\_enableGrouping\(\)](#)
- [DAC12\\_A\\_disableGrouping\(\)](#)
- [DAC12\\_A\\_enableConversions\(\)](#)
- [DAC12\\_A\\_setData\(\)](#)
- [DAC12\\_A\\_disableConversions\(\)](#)
- [DAC12\\_A\\_setResolution\(\)](#)
- [DAC12\\_A\\_setInputDataFormat\(\)](#)
- [DAC12\\_A\\_getDataBufferMemoryAddressForDMA\(\)](#)

Calibration features of the DAC12\_A are handled by

- [DAC12\\_A\\_calibrateOutput\(\)](#)
- [DAC12\\_A\\_getCalibrationData\(\)](#)
- [DAC12\\_A\\_setCalibrationOffset\(\)](#)

The DAC12\_A interrupts are handled by

- [DAC12\\_A\\_enableInterrupt\(\)](#)
- [DAC12\\_A\\_disableInterrupt\(\)](#)
- [DAC12\\_A\\_getInterruptStatus\(\)](#)
- [DAC12\\_A\\_clearInterrupt\(\)](#)

## 11.2.2 Function Documentation

### 11.2.2.1 DAC12\_A\_calibrateOutput

Calibrates the output offset.

**Prototype:**

```
void
DAC12_A_calibrateOutput(uint32_t baseAddress,
                        uint8_t submoduleSelect)
```

**Description:**

This function disables the calibration lock, starts the calibration, waits for the calibration to complete, and then re-locks the calibration lock. Please note, this function should be called after initializing the dac12 module, and before using it.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- [DAC12\\_A\\_SUBMODULE\\_0](#)
- [DAC12\\_A\\_SUBMODULE\\_1](#)

Modified bits are **DAC12CALON** of **DAC12\_xCTL0** register; bits **DAC12PW** of **DAC12\_xCALCTL** register.

**Returns:**

None

### 11.2.2.2 DAC12\_A\_clearInterrupt

Clears the DAC12\_A module interrupt flag.

**Prototype:**

```
void  
DAC12_A_clearInterrupt(uint32_t baseAddress,  
                       uint8_t submoduleSelect)
```

**Description:**

The DAC12\_A module interrupt flag is cleared, so that it no longer asserts. Note that an interrupt is not thrown when DAC12\_A\_TRIGGER\_AUTO has been set for the parameter conversionTriggerSelect in initialization.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- DAC12\_A\_SUBMODULE\_0
- DAC12\_A\_SUBMODULE\_1

Modified bits are **DAC12IFG** of **DAC12\_xCTL0** register.

**Returns:**

None

### 11.2.2.3 DAC12\_A\_disable

Clears the amplifier settings to disable the DAC12\_A module.

**Prototype:**

```
void  
DAC12_A_disable(uint32_t baseAddress,  
                uint8_t submoduleSelect)
```

**Description:**

This function clears the amplifier settings for the selected DAC12\_A module to disable the DAC12\_A module.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- DAC12\_A\_SUBMODULE\_0
- DAC12\_A\_SUBMODULE\_1

Modified bits are **DAC12AMP\_7** of **DAC12\_xCTL0** register.

**Returns:**

None

### 11.2.2.4 DAC12\_A\_disableConversions

Disables triggers to start conversions.

**Prototype:**

```
void  
DAC12_A_disableConversions(uint32_t baseAddress,  
                           uint8_t submoduleSelect)
```

**Description:**

This function is used to disallow triggers to start a conversion. Note that this function does not have any affect if DAC12\_A\_TRIGGER\_AUTO was set for the conversionTriggerSelect parameter during initialization.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- DAC12\_A\_SUBMODULE\_0
- DAC12\_A\_SUBMODULE\_1

Modified bits are **DAC12ENC** of **DAC12\_xCTL0** register.

**Returns:**

None

### 11.2.2.5 DAC12\_A\_disableGrouping

Disables grouping of two DAC12\_A modules in a dual DAC12\_A system.

**Prototype:**

```
void  
DAC12_A_disableGrouping(uint32_t baseAddress)
```

**Description:**

This function disables grouping of two DAC12\_A modules in a dual DAC12\_A system.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**Returns:**

None

### 11.2.2.6 DAC12\_A\_disableInterrupt

Disables the DAC12\_A module interrupt source.

**Prototype:**

```
void  
DAC12_A_disableInterrupt(uint32_t baseAddress,  
                          uint8_t submoduleSelect)
```

**Description:**

Enables the DAC12\_A module 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 DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- DAC12\_A\_SUBMODULE\_0
- DAC12\_A\_SUBMODULE\_1

**Returns:**

None

### 11.2.2.7 DAC12\_A\_enableConversions

Enables triggers to start conversions.

**Prototype:**

```
void  
DAC12_A_enableConversions(uint32_t baseAddress,  
                           uint8_t submoduleSelect)
```

**Description:**

This function is used to allow triggers to start a conversion. Note that this function does not need to be used if DAC12\_A\_TRIGGER\_AUTO was set for the conversionTriggerSelect parameter during initialization. If DAC grouping is enabled, this has to be called for both DAC's.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- DAC12\_A\_SUBMODULE\_0
- DAC12\_A\_SUBMODULE\_1

Modified bits are **DAC12ENC** of **DAC12\_xCTL0** register.

**Returns:**

None

### 11.2.2.8 DAC12\_A\_enableGrouping

Enables grouping of two DAC12\_A modules in a dual DAC12\_A system.

**Prototype:**

```
void  
DAC12_A_enableGrouping(uint32_t baseAddress)
```

**Description:**

This function enables grouping two DAC12\_A modules in a dual DAC12\_A system. Both DAC12\_A modules will work in sync, converting data at the same time. To convert data, the same trigger should be set for both DAC12\_A modules during initialization (which should not be DAC12\_A\_TRIGGER\_ENCBYPASS), the enableConversions() function needs to be called with both DAC12\_A modules, and data needs to be set for both DAC12\_A modules separately.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

Modified bits are **DAC12GRP** of **DAC12\_xCTL0** register.

**Returns:**

None

### 11.2.2.9 DAC12\_A\_enableInterrupt

Enables the DAC12\_A module interrupt source.

**Prototype:**

```
void  
DAC12_A_enableInterrupt(uint32_t baseAddress,  
                        uint8_t submoduleSelect)
```

**Description:**

This function to enable the DAC12\_A module interrupt, which throws an interrupt when the data buffer is available for new data to be set. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor. Note that an interrupt is not thrown when DAC12\_A\_TRIGGER\_AUTO has been set for the parameter conversionTriggerSelect in initialization. Does not clear interrupt flags.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- DAC12\_A\_SUBMODULE\_0
- DAC12\_A\_SUBMODULE\_1

**Returns:**

None

### 11.2.2.10 DAC12\_A\_getCalibrationData

Returns the calibrated offset of the output buffer.

**Prototype:**

```
uint16_t  
DAC12_A_getCalibrationData(uint32_t baseAddress,  
                           uint8_t submoduleSelect)
```

**Description:**

This function returns the calibrated offset of the output buffer. The output buffer offset is used to obtain accurate results from the output pin. This function should only be used while the calibration lock is enabled. Only the lower byte of the word of the register is returned, and the value is between -128 and +127.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- DAC12\_A\_SUBMODULE\_0
- DAC12\_A\_SUBMODULE\_1

**Returns:**

The calibrated offset of the output buffer.

### 11.2.2.11 DAC12\_A\_getDataBufferMemoryAddressForDMA

Returns the address of the specified DAC12\_A data buffer for the DMA module.

**Prototype:**

```
uint32_t  
DAC12_A_getDataBufferMemoryAddressForDMA(uint32_t baseAddress,  
                                           uint8_t submoduleSelect)
```

**Description:**

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

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- DAC12\_A\_SUBMODULE\_0
- DAC12\_A\_SUBMODULE\_1

**Returns:**

The address of the specified memory buffer

### 11.2.2.12 DAC12\_A\_getInterruptStatus

Returns the status of the DAC12\_A module interrupt flag.

**Prototype:**

```
uint16_t  
DAC12_A_getInterruptStatus(uint32_t baseAddress,  
                           uint8_t submoduleSelect)
```

**Description:**

This function returns the status of the DAC12\_A module interrupt flag. Note that an interrupt is not thrown when DAC12\_A\_TRIGGER\_AUTO has been set for the conversionTriggerSelect parameter in initialization.



**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- **DAC12\_A\_SUBMODULE\_0**
- **DAC12\_A\_SUBMODULE\_1**

**Returns:**

One of the following:

- **DAC12\_A\_INT\_ACTIVE**
  - **DAC12\_A\_INT\_INACTIVE**
- indicating the status for the selected DAC12\_A module

### 11.2.2.13 DAC12\_A\_init

Initializes the DAC12\_A module with the specified settings.

**Prototype:**

```
bool
DAC12_A_init (uint32_t baseAddress,
              uint8_t submoduleSelect,
              uint16_t outputSelect,
              uint16_t positiveReferenceVoltage,
              uint16_t outputVoltageMultiplier,
              uint8_t amplifierSetting,
              uint16_t conversionTriggerSelect)
```

**Description:**

This function initializes the DAC12\_A module with the specified settings. Upon successful completion of the initialization of this module the control registers and interrupts of this module are all reset, and the specified variables will be set. Please note, that if conversions are enabled with the `enableConversions()` function, then `disableConversions()` must be called before re-initializing the DAC12\_A module with this function.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- **DAC12\_A\_SUBMODULE\_0**
- **DAC12\_A\_SUBMODULE\_1**

**outputSelect** selects the output pin that the selected DAC12\_A module will output to. Valid values are:

- **DAC12\_A\_OUTPUT\_1** [Default]
- **DAC12\_A\_OUTPUT\_2**

Modified bits are **DAC12OPS** of **DAC12\_xCTL0** register.

**positiveReferenceVoltage** is the upper limit voltage that the data can be converted in to. Valid values are:

- **DAC12\_A\_VREF\_AVCC** [Default]
- **DAC12\_A\_VREF\_INT**
- **DAC12\_A\_VREF\_EXT**

Modified bits are **DAC12SREFx** of **DAC12\_xCTL0** register.

**outputVoltageMultiplier** is the multiplier of the Vout voltage. Valid values are:

- **DAC12\_A\_VREFx1** [Default]
- **DAC12\_A\_VREFx2**
- **DAC12\_A\_VREFx3**

Modified bits are **DAC12IR** of **DAC12\_xCTL0** register; bits **DAC12OG** of **DAC12\_xCTL1** register.

**amplifierSetting** is the setting of the settling speed and current of the Vref+ and the Vout buffer. Valid values are:

- **DAC12\_A\_AMP\_OFF\_PINOUTHIGHZ** [Default] - Initialize the DAC12\_A Module with settings, but do not turn it on.
- **DAC12\_A\_AMP\_OFF\_PINOUTLOW** - Initialize the DAC12\_A Module with settings, and allow it to take control of the selected output pin to pull it low (Note: this takes control away port mapping module).
- **DAC12\_A\_AMP\_LOWIN\_LOWOUT** - Select a slow settling speed and current for Vref+ input buffer and for Vout output buffer.

- **DAC12\_A\_AMP\_LOWIN\_MEDOUT** - Select a slow settling speed and current for Vref+ input buffer and a medium settling speed and current for Vout output buffer.
  - **DAC12\_A\_AMP\_LOWIN\_HIGHOUT** - Select a slow settling speed and current for Vref+ input buffer and a high settling speed and current for Vout output buffer.
  - **DAC12\_A\_AMP\_MEDIN\_MEDOUT** - Select a medium settling speed and current for Vref+ input buffer and for Vout output buffer.
  - **DAC12\_A\_AMP\_MEDIN\_HIGHOUT** - Select a medium settling speed and current for Vref+ input buffer and a high settling speed and current for Vout output buffer.
  - **DAC12\_A\_AMP\_HIGHIN\_HIGHOUT** - Select a high settling speed and current for Vref+ input buffer and for Vout output buffer.
- Modified bits are **DAC12AMPx** of **DAC12\_xCTL0** register.

**conversionTriggerSelect** selects the trigger that will start a conversion. Valid values are:

- **DAC12\_A\_TRIGGER\_ENCBYPASS** [Default] - Automatically converts data as soon as it is written into the data buffer. (Note: Do not use this selection if grouping DAC's).
  - **DAC12\_A\_TRIGGER\_ENC** - Requires a call to `enableConversions()` to allow a conversion, but starts a conversion as soon as data is written to the data buffer (Note: with DAC12\_A module's grouped, data has to be set in BOTH DAC12\_A data buffers to start a conversion).
  - **DAC12\_A\_TRIGGER\_TA** - Requires a call to `enableConversions()` to allow a conversion, and a rising edge of Timer\_A's Out1 (TA1) to start a conversion.
  - **DAC12\_A\_TRIGGER\_TB** - Requires a call to `enableConversions()` to allow a conversion, and a rising edge of Timer\_B's Out2 (TB2) to start a conversion.
- Modified bits are **DAC12LSELx** of **DAC12\_xCTL0** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of the initialization process.

### 11.2.2.14 DAC12\_A\_setAmplifierSetting

Sets the amplifier settings for the Vref+ and Vout buffers.

**Prototype:**

```
void
DAC12_A_setAmplifierSetting(uint32_t baseAddress,
                           uint8_t submoduleSelect,
                           uint8_t amplifierSetting)
```

**Description:**

This function sets the amplifier settings of the DAC12\_A module for the Vref+ and Vout buffers without re-initializing the DAC12\_A module. This can be used to disable the control of the pin by the DAC12\_A module.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- **DAC12\_A\_SUBMODULE\_0**
- **DAC12\_A\_SUBMODULE\_1**

**amplifierSetting** is the setting of the settling speed and current of the Vref+ and the Vout buffer. Valid values are:

- **DAC12\_A\_AMP\_OFF\_PINOUTHIGHZ** [Default] - Initialize the DAC12\_A Module with settings, but do not turn it on.
- **DAC12\_A\_AMP\_OFF\_PINOUTLOW** - Initialize the DAC12\_A Module with settings, and allow it to take control of the selected output pin to pull it low (Note: this takes control away port mapping module).
- **DAC12\_A\_AMP\_LOWIN\_LOWOUT** - Select a slow settling speed and current for Vref+ input buffer and for Vout output buffer.
- **DAC12\_A\_AMP\_LOWIN\_MEDOUT** - Select a slow settling speed and current for Vref+ input buffer and a medium settling speed and current for Vout output buffer.
- **DAC12\_A\_AMP\_LOWIN\_HIGHOUT** - Select a slow settling speed and current for Vref+ input buffer and a high settling speed and current for Vout output buffer.
- **DAC12\_A\_AMP\_MEDIN\_MEDOUT** - Select a medium settling speed and current for Vref+ input buffer and for Vout output buffer.
- **DAC12\_A\_AMP\_MEDIN\_HIGHOUT** - Select a medium settling speed and current for Vref+ input buffer and a high settling speed and current for Vout output buffer.

- **DAC12\_A\_AMP\_HIGHIN\_HIGHOUT** - Select a high settling speed and current for Vref+ input buffer and for Vout output buffer.

**Returns:**  
None

### 11.2.2.15 DAC12\_A\_setCalibrationOffset

Returns the calibrated offset of the output buffer.

**Prototype:**

```
void
DAC12_A_setCalibrationOffset(uint32_t baseAddress,
                             uint8_t submoduleSelect,
                             uint16_t calibrationOffsetValue)
```

**Description:**

This function is used to manually set the calibration offset value. The calibration is automatically unlocked and re-locked to be able to allow for the offset value to be set.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- **DAC12\_A\_SUBMODULE\_0**
- **DAC12\_A\_SUBMODULE\_1**

**calibrationOffsetValue** calibration offset value

Modified bits are **DAC12LOCK** of **DAC12\_xCALDAT** register; bits **DAC12PW** of **DAC12\_xCTL0** register; bits **DAC12PW** of **DAC12\_xCALCTL** register.

**Returns:**  
None

### 11.2.2.16 DAC12\_A\_setData

Sets the given data into the buffer to be converted.

**Prototype:**

```
void
DAC12_A_setData(uint32_t baseAddress,
                 uint8_t submoduleSelect,
                 uint16_t data)
```

**Description:**

This function is used to set the given data into the data buffer of the DAC12\_A module. The data given should be in the format set (12-bit Unsigned, Right-justified by default). Note if **DAC12\_A\_TRIGGER\_AUTO** was set for the **conversionTriggerSelect** during initialization then using this function will set the data and automatically trigger a conversion. If any other trigger was set during initialization, then the [DAC12\\_A\\_enableConversions\(\)](#) function needs to be called before a conversion can be started. If grouping DAC's and **DAC12\_A\_TRIGGER\_ENC** was set during initialization, then both data buffers must be set before a conversion will be started.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- **DAC12\_A\_SUBMODULE\_0**
- **DAC12\_A\_SUBMODULE\_1**

**data** is the data to be set into the DAC12\_A data buffer to be converted.

Modified bits are **DAC12\_DATA** of **DAC12\_xDAT** register.

Modified bits of **DAC12\_xDAT** register.

**Returns:**  
None

### 11.2.2.17 DAC12\_A\_setInputDataFormat

Sets the input data format for the DAC12\_A module.

**Prototype:**

```
void
DAC12_A_setInputDataFormat(uint32_t baseAddress,
                           uint8_t submoduleSelect,
                           uint8_t inputJustification,
                           uint8_t inputSign)
```

**Description:**

This function sets the input format for the binary data to be converted.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- **DAC12\_A\_SUBMODULE\_0**
- **DAC12\_A\_SUBMODULE\_1**

**inputJustification** is the justification of the data to be converted. Valid values are:

- **DAC12\_A\_JUSTIFICATION\_RIGHT** [Default]
- **DAC12\_A\_JUSTIFICATION\_LEFT**

Modified bits are **DAC12DFJ** of **DAC12\_xCTL1** register.

**inputSign** is the sign of the data to be converted. Valid values are:

- **DAC12\_A\_UNSIGNED\_BINARY** [Default]
- **DAC12\_A\_SIGNED\_2SCOMPLEMENT**

Modified bits are **DAC12DF** of **DAC12\_xCTL0** register.

**Returns:**  
None

### 11.2.2.18 DAC12\_A\_setResolution

Sets the resolution to be used by the DAC12\_A module.

**Prototype:**

```
void
DAC12_A_setResolution(uint32_t baseAddress,
                      uint8_t submoduleSelect,
                      uint16_t resolutionSelect)
```

**Description:**

This function sets the resolution of the data to be converted.

**Parameters:**

**baseAddress** is the base address of the DAC12\_A module.

**submoduleSelect** decides which DAC12\_A sub-module to configure. Valid values are:

- **DAC12\_A\_SUBMODULE\_0**
- **DAC12\_A\_SUBMODULE\_1**

**resolutionSelect** is the resolution to use for conversions. Valid values are:

- **DAC12\_A\_RESOLUTION\_8BIT**
- **DAC12\_A\_RESOLUTION\_12BIT** [Default]

Modified bits are **DAC12RES** of **DAC12\_xCTL0** register.

Modified bits are **DAC12ENC** and **DAC12RES** of **DAC12\_xCTL0** register.

**Returns:**  
None

## 11.3 Programming Example

The following example shows how to initialize and use the DAC12\_A API to output a 1.5V analog signal.

```
DAC12_A_init (DAC12_A_BASE,
              DAC12_A_SUBMODULE_0,           // Initialize DAC12_A_0
              DAC12_A_OUTPUT_1,              // Choose P6.6 as output
              DAC12_A_VREF_AVCC,             // Use AVcc as Vref+
              DAC12_A_VREFx1,                // Multiply Vout by 1
              DAC12_A_AMP_MEDIN_MEDOUT,      // Use medium settling speed/current
              DAC12_A_TRIGGER_ENCBYPASS      // Auto trigger as soon as data is set
              );

// Calibrate output buffer for DAC12_A_0
DAC12_A_calibrateOutput (DAC12_A_BASE,
                        DAC12_A_SUBMODULE_0);

DAC12_A_setData (DAC12_A_BASE,
                 DAC12_A_SUBMODULE_0,        // Set 0x7FF (~1.5V)
                 0x7FF                       // into data buffer for DAC12_A_0
                 );
```

## 12 Direct Memory Access (DMA)

Introduction .....	114
API Functions .....	114
Programming Example .....	129

### 12.1 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

### 12.2 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)

## 12.2.1 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\(\)](#)

## 12.2.2 Function Documentation

### 12.2.2.1 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

### 12.2.2.2 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

### 12.2.2.3 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

## 12.2.2.4 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

### 12.2.2.5 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

### 12.2.2.6 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

### 12.2.2.7 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

### 12.2.2.8 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

### 12.2.2.9 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

### 12.2.2.10 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

### 12.2.2.11 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

### 12.2.2.12 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

### 12.2.2.13 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

### 12.2.2.14 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 **DMA DT** 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 **DMA SRCBYTE** and **DMA DSTBYTE** 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.

### 12.2.2.15 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

### 12.2.2.16 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

### 12.2.2.17 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

## 12.2.2.18 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

### 12.2.2.19 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

## 12.3 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);
}
```

# 13 EUSCI Universal Asynchronous Receiver/Transmitter (EUSCI\_A\_UART)

Introduction .....	130
API Functions .....	130
Programming Example .....	139

## 13.1 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.

## 13.2 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 clockPrescaler, uint8\_t firstModReg, uint8\_t secondModReg, uint8\_t parity, uint16\_t msborLsbFirst, uint16\_t numberOfStopBits, uint16\_t uartMode, uint8\_t overSampling)
- uint8\_t [EUSCI\\_A\\_UART\\_queryStatusFlags](#) (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)

## 13.2.1 Detailed Description

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

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

Configuration and control of the EUSCI\_UART are handled by the

- `EUSCI_A_UART_init()`
- `EUSCI_A_UART_initAdvance()`
- `EUSCI_A_UART_enable()`
- `EUSCI_A_UART_disable()`
- `EUSCI_A_UART_setDormant()`
- `EUSCI_A_UART_resetDormant()`
- `EUSCI_A_UART_selectDeglitchTime()`

Sending and receiving data via the EUSCI\_UART is handled by the

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

Managing the EUSCI\_UART interrupts and status are handled by the

- `EUSCI_A_UART_enableInterrupt()`
- `EUSCI_A_UART_disableInterrupt()`
- `EUSCI_A_UART_getInterruptStatus()`
- `EUSCI_A_UART_clearInterruptFlag()`
- `EUSCI_A_UART_queryStatusFlags()`

DMA related

- `EUSCI_A_UART_getReceiveBufferAddressForDMA()`
- `EUSCI_A_UART_getTransmitBufferAddressForDMA()`

## 13.2.2 Function Documentation

### 13.2.2.1 EUSCI\_A\_UART\_clearInterruptFlag

Clears UART interrupt sources.

**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

### 13.2.2.2 EUSCI\_A\_UART\_disable

Disables the UART block.

**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

### 13.2.2.3 EUSCI\_A\_UART\_disableInterrupt

Disables individual UART interrupt sources.

**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



### 13.2.2.4 EUSCI\_A\_UART\_enable

Enables the UART block.

**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

### 13.2.2.5 EUSCI\_A\_UART\_enableInterrupt

Enables individual UART interrupt sources.

**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

### 13.2.2.6 EUSCI\_A\_UART\_getInterruptStatus

Gets the current UART interrupt status.

**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

### 13.2.2.7 EUSCI\_A\_UART\_getReceiveBufferAddress

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

**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

### 13.2.2.8 EUSCI\_A\_UART\_getTransmitBufferAddress

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

**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

### 13.2.2.9 EUSCI\_A\_UART\_initAdvance

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

**Prototype:**

```
bool
EUSCI_A_UART_initAdvance(uint32_t baseAddress,
                        uint8_t selectClockSource,
                        uint16_t clockPrescaler,
                        uint8_t firstModReg,
                        uint8_t secondModReg,
                        uint8_t parity,
                        uint16_t msborLsbFirst,
                        uint16_t numberOfStopBits,
                        uint16_t uartMode,
                        uint8_t overSampling)
```

**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 clockPrescaler, firstModReg, secondModReg and overSampling please use the link below.

[http://software-dl.ti.com/msp430/msp430\\_public\\_sw/mcu/msp430/MSP430BaudRateConverter/index.html](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

**clockPrescaler** 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 UCxMCTLW.

**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 UCxMCTLW.

**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 UCxCTL0 register; bits UCSSELx and UCSWRST of UCxCTL1 register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL of the initialization process

### 13.2.2.10 EUSCI\_A\_UART\_queryStatusFlags

Gets the current UART status flags.

**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

### 13.2.2.11 EUSCI\_A\_UART\_receiveData

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

**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.

### 13.2.2.12 EUSCI\_A\_UART\_resetDormant

Re-enables UART module from dormant mode.

**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

### 13.2.2.13 EUSCI\_A\_UART\_selectDeglitchTime

Sets the deglitch time.

**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

### 13.2.2.14 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.

**Parameters:**

**baseAddress** is the base address of the EUSCI\_A\_UART module.

**Description:**

Modified bits of **UCAxCTL1** register.

**Returns:**

None

### 13.2.2.15 EUSCI\_A\_UART\_transmitAddress

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

**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

### 13.2.2.16 EUSCI\_A\_UART\_transmitBreak

Transmit break.

**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

### 13.2.2.17 EUSCI\_A\_UART\_transmitData

Transmits a byte from the UART Module.

**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

## 13.3 Programming Example

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

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

EUSCI_A_UART_enable(EUSCI_A0_BASE);

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

# 14 EUSCI Synchronous Peripheral Interface (EUSCI\_A\_SPI)

Introduction .....	140
API Functions .....	140
Programming Example .....	148

## 14.1 Introduction

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

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

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

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

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

## 14.2 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)

### 14.2.1 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()`

## 14.2.2 Function Documentation

### 14.2.2.1 EUSCI\_A\_SPI\_changeClockPhasePolarity

Changes the SPI clock phase and polarity. At the end of this function call, SPI module is left enabled.

#### 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

### 14.2.2.2 EUSCI\_A\_SPI\_clearInterruptFlag

Clears the selected SPI interrupt status flag.

**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

### 14.2.2.3 EUSCI\_A\_SPI\_disable

Disables the SPI block.

**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

### 14.2.2.4 EUSCI\_A\_SPI\_disableInterrupt

Disables individual SPI interrupt sources.

**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

### 14.2.2.5 EUSCI\_A\_SPI\_enable

Enables the SPI block.

**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

### 14.2.2.6 EUSCI\_A\_SPI\_enableInterrupt

Enables individual SPI interrupt sources.

**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

### 14.2.2.7 EUSCI\_A\_SPI\_getInterruptStatus

Gets the current SPI interrupt status.

**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

### 14.2.2.8 EUSCI\_A\_SPI\_getReceiveBufferAddress

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

**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

### 14.2.2.9 EUSCI\_A\_SPI\_getTransmitBufferAddress

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

**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

### 14.2.2.10 EUSCI\_A\_SPI\_isBusy

Indicates whether or not the SPI bus is busy.

**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

### 14.2.2.11 EUSCI\_A\_SPI\_masterChangeClock

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

**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

### 14.2.2.12 EUSCI\_A\_SPI\_masterInit

Initializes the SPI Master block.

**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

### 14.2.2.13 EUSCI\_A\_SPI\_receiveData

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

**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.

### 14.2.2.14 EUSCI\_A\_SPI\_select4PinFunctionality

Selects 4Pin Functionality.

**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

### 14.2.2.15 EUSCI\_A\_SPI\_slaveInit

Initializes the SPI Slave block.

**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

### 14.2.2.16 EUSCI\_A\_SPI\_transmitData

Transmits a byte from the SPI Module.

**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

## 14.3 Programming Example

The following example shows how to use the SPI API to configure the SPI module as a master device, and how to do a simple send of data.

```
//Initialize slave to MSB first, inactive high clock polarity and 3 wire SPI
returnValue = EUSCI_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
);
```



# 15 EUSCI Synchronous Peripheral Interface (EUSCI\_B\_SPI)

Introduction .....	149
API Functions .....	149
Programming Example .....	157

## 15.1 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.

## 15.2 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)

### 15.2.1 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()`

## 15.2.2 Function Documentation

### 15.2.2.1 EUSCI\_B\_SPI\_changeClockPhasePolarity

Changes the SPI clock phase and polarity. At the end of this function call, SPI module is left enabled.

#### 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

### 15.2.2.2 EUSCI\_B\_SPI\_clearInterruptFlag

Clears the selected SPI interrupt status flag.

**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

### 15.2.2.3 EUSCI\_B\_SPI\_disable

Disables the SPI block.

**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

### 15.2.2.4 EUSCI\_B\_SPI\_disableInterrupt

Disables individual SPI interrupt sources.

**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

### 15.2.2.5 EUSCI\_B\_SPI\_enable

Enables the SPI block.

**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

### 15.2.2.6 EUSCI\_B\_SPI\_enableInterrupt

Enables individual SPI interrupt sources.

**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

### 15.2.2.7 EUSCI\_B\_SPI\_getInterruptStatus

Gets the current SPI interrupt status.

**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

### 15.2.2.8 EUSCI\_B\_SPI\_getReceiveBufferAddress

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

**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

### 15.2.2.9 EUSCI\_B\_SPI\_getTransmitBufferAddress

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

**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

### 15.2.2.10 EUSCI\_B\_SPI\_isBusy

Indicates whether or not the SPI bus is busy.

**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

### 15.2.2.11 EUSCI\_B\_SPI\_masterChangeClock

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

**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

### 15.2.2.12 EUSCI\_B\_SPI\_masterInit

Initializes the SPI Master block.

**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

### 15.2.2.13 EUSCI\_B\_SPI\_receiveData

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

**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.

### 15.2.2.14 EUSCI\_B\_SPI\_select4PinFunctionality

Selects 4Pin Functionality.

**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

### 15.2.2.15 EUSCI\_B\_SPI\_slaveInit

Initializes the SPI Slave block.

**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

### 15.2.2.16 EUSCI\_B\_SPI\_transmitData

Transmits a byte from the SPI Module.

**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

## 15.3 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
);
```

# 16 EUSCI Inter-Integrated Circuit (EUSCI\_B\_I2C)

Introduction .....	158
API Functions .....	159
Programming Example .....	176

## 16.1 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.

### 16.1.1 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.

## 16.1.2 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.

## 16.2 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)

## 16.2.1 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

## 16.2.2 Function Documentation

### 16.2.2.1 EUSCI\_B\_I2C\_clearInterruptFlag

Clears I2C interrupt sources.

**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

### 16.2.2.2 EUSCI\_B\_I2C\_disable

Disables the I2C block.

**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

### 16.2.2.3 EUSCI\_B\_I2C\_disableInterrupt

Disables individual I2C interrupt sources.

**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

#### 16.2.2.4 EUSCI\_B\_I2C\_disableMultiMasterMode

Disables Multi Master Mode.

**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

#### 16.2.2.5 EUSCI\_B\_I2C\_enable

Enables the I2C block.

**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

#### 16.2.2.6 EUSCI\_B\_I2C\_enableInterrupt

Enables individual I2C interrupt sources.

**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

### 16.2.2.7 EUSCI\_B\_I2C\_enableMultiMasterMode

Enables Multi Master Mode.

**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

### 16.2.2.8 EUSCI\_B\_I2C\_getInterruptStatus

Gets the current I2C interrupt status.

**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

### 16.2.2.9 EUSCI\_B\_I2C\_getMode

Gets the mode of the I2C device.

**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

### 16.2.2.10 EUSCI\_B\_I2C\_getReceiveBufferAddress

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

**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

### 16.2.2.11 EUSCI\_B\_I2C\_getTransmitBufferAddress

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

**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

### 16.2.2.12 EUSCI\_B\_I2C\_isBusBusy

Indicates whether or not the I2C bus is busy.

**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

### 16.2.2.13 EUSCI\_B\_I2C\_masterInit

Initializes the I2C Master block.

**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

### 16.2.2.14 EUSCI\_B\_I2C\_masterIsStartSent

Indicates whether Start got sent.

**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

### 16.2.2.15 EUSCI\_B\_I2C\_masterIsStopSent

Indicates whether STOP got sent.

**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

### 16.2.2.16 EUSCI\_B\_I2C\_masterMultiByteReceiveFinish

Finishes multi-byte reception at the Master end.

**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.

### 16.2.2.17 EUSCI\_B\_I2C\_masterMultiByteReceiveFinishWithTimeout

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

**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

### 16.2.2.18 EUSCI\_B\_I2C\_masterMultiByteReceiveNext

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

**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.

### 16.2.2.19 EUSCI\_B\_I2C\_masterMultiByteReceiveStop

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

**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

### 16.2.2.20 EUSCI\_B\_I2C\_masterMultiByteSendFinish

Finishes multi-byte transmission from Master to Slave.

**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

### 16.2.2.21 EUSCI\_B\_I2C\_masterMultiByteSendFinishWithTimeout

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

**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.

### 16.2.2.22 EUSCI\_B\_I2C\_masterMultiByteSendNext

Continues multi-byte transmission from Master to Slave.

**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

### 16.2.2.23 EUSCI\_B\_I2C\_masterMultiByteSendNextWithTimeout

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

**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.

### 16.2.2.24 EUSCI\_B\_I2C\_masterMultiByteSendStart

Starts multi-byte transmission from Master to Slave.

**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

### 16.2.2.25 EUSCI\_B\_I2C\_masterMultiByteSendStartWithTimeout

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

**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.

### 16.2.2.26 EUSCI\_B\_I2C\_masterMultiByteSendStop

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

**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

### 16.2.2.27 EUSCI\_B\_I2C\_masterMultiByteSendStopWithTimeout

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

**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.



### 16.2.2.28 EUSCI\_B\_I2C\_masterReceiveSingleByte

Does single byte reception from Slave.

**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.

### 16.2.2.29 EUSCI\_B\_I2C\_masterReceiveStart

Starts reception at the Master end.

**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

### 16.2.2.30 EUSCI\_B\_I2C\_masterSendSingleByte

Does single byte transmission from Master to Slave.

**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

### 16.2.2.31 EUSCI\_B\_I2C\_masterSendSingleByteWithTimeout

Does single byte transmission from Master to Slave with timeout.

**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.

### 16.2.2.32 EUSCI\_B\_I2C\_masterSendStart

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

**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

### 16.2.2.33 EUSCI\_B\_I2C\_masterSingleReceive

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

**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.

### 16.2.2.34 EUSCI\_B\_I2C\_setMode

Sets the mode of the I2C device.

**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

### 16.2.2.35 EUSCI\_B\_I2C\_setSlaveAddress

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

**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

### 16.2.2.36 EUSCI\_B\_I2C\_slaveDataGet

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

**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.

### 16.2.2.37 EUSCI\_B\_I2C\_slaveDataPut

Transmits a byte from the I2C Module.

**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

### 16.2.2.38 EUSCI\_B\_I2C\_slaveInit

Initializes the I2C Slave block.

**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

## 16.3 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);
```

# 17 Flash Memory Controller

Introduction .....	178
API Functions .....	178
Programming Example .....	184

## 17.1 Introduction

The flash memory is byte, word, and long-word addressable and programmable. The flash memory module has an integrated controller that controls programming and erase operations. The flash main memory is partitioned into 512-byte segments. Single bits, bytes, or words can be written to flash memory, but a segment is the smallest size of the flash memory that can be erased. The flash memory is partitioned into main and information memory sections. There is no difference in the operation of the main and information memory sections. Code and data can be located in either section. The difference between the sections is the segment size. There are four information memory segments, A through D. Each information memory segment contains 128 bytes and can be erased individually. The bootstrap loader (BSL) memory consists of four segments, A through D. Each BSL memory segment contains 512 bytes and can be erased individually. The main memory segment size is 512 byte. See the device-specific data sheet for the start and end addresses of each bank, when available, and for the complete memory map of a device. This library provides the API for flash segment erase, flash writes and flash operation status check.

This driver is contained in `flash.c`, with `flash.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	620
CCS 4.2.1	Size	364
CCS 4.2.1	Speed	366
IAR 5.51.6	None	378
IAR 5.51.6	Size	198
IAR 5.51.6	Speed	260
MSPGCC 4.8.0	None	896
MSPGCC 4.8.0	Size	422
MSPGCC 4.8.0	Speed	392

## 17.2 API Functions

### Functions

- void [FLASH\\_bankErase](#) (uint8\_t \*flash\_ptr)
- bool [FLASH\\_eraseCheck](#) (uint8\_t \*flash\_ptr, uint16\_t numberOfBytes)
- void [FLASH\\_lockInfoA](#) (void)
- void [FLASH\\_memoryFill32](#) (uint32\_t value, uint32\_t \*flash\_ptr, uint16\_t count)
- void [FLASH\\_segmentErase](#) (uint8\_t \*flash\_ptr)
- uint8\_t [FLASH\\_status](#) (uint8\_t mask)
- void [FLASH\\_unlockInfoA](#) (void)
- void [FLASH\\_write16](#) (uint16\_t \*data\_ptr, uint16\_t \*flash\_ptr, uint16\_t count)
- void [FLASH\\_write32](#) (uint32\_t \*data\_ptr, uint32\_t \*flash\_ptr, uint16\_t count)
- void [FLASH\\_write8](#) (uint8\_t \*data\_ptr, uint8\_t \*flash\_ptr, uint16\_t count)

## 17.2.1 Detailed Description

FLASH\_segmentErase helps erase a single segment of the flash memory. A pointer to the flash segment being erased is passed on to this function.

FLASH\_eraseCheck helps check if a specific number of bytes in flash are currently erased. A pointer to the starting location of the erase check and the number of bytes to be checked is passed into this function.

Depending on the kind of writes being performed to the flash, this library provides APIs for flash writes.

FLASH\_write8 facilitates writing into the flash memory in byte format. FLASH\_write16 facilitates writing into the flash memory in word format. FLASH\_write32 facilitates writing into the flash memory in long format, pass by reference. FLASH\_memoryFill32 facilitates writing into the flash memory in long format, pass by value. FLASH\_status checks if the flash is currently busy erasing or programming. FLASH\_lockInfoA locks segment A of information memory. FLASH\_unlockInfoA unlocks segment A of information memory.

The Flash API is broken into 4 groups of functions: those that deal with flash erase, those that write into flash, those that give status of flash, and those that lock/unlock segment A of information memory.

The flash erase operations are managed by

- [FLASH\\_segmentErase\(\)](#)
- [FLASH\\_eraseCheck\(\)](#)
- [FLASH\\_bankErase\(\)](#)

Flash writes are managed by

- [FLASH\\_write8\(\)](#)
- [FLASH\\_write16\(\)](#)
- [FLASH\\_write32\(\)](#)
- [FLASH\\_memoryFill32\(\)](#)

The status is given by

- [FLASH\\_status\(\)](#)
- [FLASH\\_eraseCheck\(\)](#)

The segment A of information memory lock/unlock operations are managed by

- [FLASH\\_lockInfoA\(\)](#)
- [FLASH\\_unlockInfoA\(\)](#)

## 17.2.2 Function Documentation

### 17.2.2.1 FLASH\_bankErase

Erase a single bank of the flash memory.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	50
CCS 4.2.1	Size	42
CCS 4.2.1	Speed	42
IAR 5.51.6	None	42
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	92
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

**Prototype:**

```
void
FLASH_bankErase(uint8_t *flash_ptr)
```

**Parameters:**

**flash\_ptr** is a pointer into the bank to be erased

**Returns:**

None

### 17.2.2.2 bool FLASH\_eraseCheck (uint8\_t \* flash\_ptr, uint16\_t numberOfBytes)

Erase check of the flash memory.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	62
CCS 4.2.1	Size	26
CCS 4.2.1	Speed	26
IAR 5.51.6	None	30
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	70
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	36

**Parameters:**

**flash\_ptr** is the pointer to the starting location of the erase check  
**numberOfBytes** is the number of bytes to be checked

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL

### 17.2.2.3 void FLASH\_lockInfoA (void)

Locks the information flash memory segment A.

This function is typically called after an erase or write operation on the information flash segment is performed by any of the other API functions in order to re-lock the information flash segment.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	40
CCS 4.2.1	Size	32
CCS 4.2.1	Speed	32
IAR 5.51.6	None	34
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	16
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

**Returns:**

None



#### 17.2.2.4 void FLASH\_memoryFill32 (uint32\_t *value*, uint32\_t \* *flash\_ptr*, uint16\_t *count*)

Write data into the flash memory in long format, pass by value Assumes the flash memory is already erased.  
FLASH\_segmentErase can be used to erase a segment.

##### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	90
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	50
IAR 5.51.6	None	50
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	124
MSPGCC 4.8.0	Size	52
MSPGCC 4.8.0	Speed	52

##### Parameters:

***value*** value to fill memory with

***flash\_ptr*** is the pointer into which to write the data

***count*** number of times to write the value

##### Returns:

None

#### 17.2.2.5 void FLASH\_segmentErase (uint8\_t \* *flash\_ptr*)

Erase a single segment of the flash memory.

##### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	44
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	36
IAR 5.51.6	None	36
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	74
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	36

##### Parameters:

***flash\_ptr*** is the pointer into the flash segment to be erased

##### Returns:

None

#### 17.2.2.6 uint8\_t FLASH\_status (uint8\_t *mask*)

Check FLASH status to see if it is currently busy erasing or programming.

##### 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

**Parameters:**

**mask** FLASH status to read Mask value is the logical OR of any of the following:

- FLASH\_READY\_FOR\_NEXT\_WRITE
- FLASH\_ACCESS\_VIOLATION\_INTERRUPT\_FLAG
- FLASH\_PASSWORD\_WRITTEN\_INCORRECTLY
- FLASH\_BUSY

**Returns:**

Logical OR of any of the following:

- FLASH\_READY\_FOR\_NEXT\_WRITE
  - FLASH\_ACCESS\_VIOLATION\_INTERRUPT\_FLAG
  - FLASH\_PASSWORD\_WRITTEN\_INCORRECTLY
  - FLASH\_BUSY
- indicating the status of the FLASH

### 17.2.2.7 void FLASH\_unlockInfoA (void)

Unlocks the information flash memory segment A.

This function must be called before an erase or write operation on the information flash segment is performed by any of the other API functions.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	30
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	52
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

**Returns:**

None

### 17.2.2.8 void FLASH\_write16 (uint16\_t \* data\_ptr, uint16\_t \* flash\_ptr, uint16\_t count)

Write data into the flash memory in word format. Assumes the flash memory is already erased. FLASH\_segmentErase can be used to erase a segment.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	92
CCS 4.2.1	Size	46
CCS 4.2.1	Speed	46
IAR 5.51.6	None	48
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	130
MSPGCC 4.8.0	Size	62
MSPGCC 4.8.0	Speed	50

**Parameters:**

***data\_ptr*** is the pointer to the data to be written

***flash\_ptr*** is the pointer into which to write the data

***count*** number of times to write the value

**Returns:**

None

### 17.2.2.9 void FLASH\_write32 (uint32\_t \* *data\_ptr*, uint32\_t \* *flash\_ptr*, uint16\_t *count*)

Write data into the flash memory in long format, pass by reference Assumes the flash memory is already erased. FLASH\_segmentErase can be used to erase a segment.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	98
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	50
IAR 5.51.6	None	54
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	138
MSPGCC 4.8.0	Size	68
MSPGCC 4.8.0	Speed	56

**Parameters:**

***data\_ptr*** is the pointer to the data to be written

***flash\_ptr*** is the pointer into which to write the data

***count*** number of times to write the value

**Returns:**

None

### 17.2.2.10 void FLASH\_write8 (uint8\_t \* *data\_ptr*, uint8\_t \* *flash\_ptr*, uint16\_t *count*)

Write data into the flash memory in byte format. Assumes the flash memory is already erased. FLASH\_segmentErase can be used to erase a segment.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	92
CCS 4.2.1	Size	46
CCS 4.2.1	Speed	46
IAR 5.51.6	None	48
IAR 5.51.6	Size	24
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	130
MSPGCC 4.8.0	Size	60
MSPGCC 4.8.0	Speed	50

**Parameters:**

***data\_ptr*** is the pointer to the data to be written

***flash\_ptr*** is the pointer into which to write the data

***count*** number of times to write the value

**Returns:**

None

## 17.3 Programming Example

The following example shows some flash operations using the APIs

```
do{
    FLASH_segmentErase(FLASH_BASE,
                       (unsigned char *)INFOD_START
                       );
    status = FLASH_eraseCheck(FLASH_BASE,
                              (unsigned char *)INFOD_START,
                              128
                              );
}while(status == STATUS_FAIL);

//Flash write
FLASH_write32(FLASH_BASE,
              calibration_data,
              (unsigned long *) (INFOD_START),1);
```

# 18 GPIO

Introduction .....	185
API Functions .....	186
Programming Example .....	204

## 18.1 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, as well as, configurable drive strength, full or reduced. 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	2950
CCS 4.2.1	Size	1544
CCS 4.2.1	Speed	1550
IAR 5.51.6	None	2168
IAR 5.51.6	Size	1738
IAR 5.51.6	Speed	3018
MSPGCC 4.8.0	None	5230
MSPGCC 4.8.0	Size	2092
MSPGCC 4.8.0	Speed	2106

## 18.2 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)
- void [GPIO\\_setAsPeripheralModuleFunctionOutputPin](#) (uint8\_t selectedPort, uint16\_t selectedPins)
- void [GPIO\\_setDriveStrength](#) (uint8\_t selectedPort, uint16\_t selectedPins, uint8\_t driveStrength)
- 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)

### 18.2.1 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\\_setDriveStrength\(\)](#)
- [GPIO\\_setAsPeripheralModuleFunctionOutputPin\(\)](#)
- [GPIO\\_setAsPeripheralModuleFunctionInputPin\(\)](#)

The GPIO interrupts are handled with

- [GPIO\\_enableInterrupt\(\)](#)
- [GPIO\\_disableInterrupt\(\)](#)
- [GPIO\\_clearInterruptFlag\(\)](#)
- [GPIO\\_getInterruptStatus\(\)](#)

- `GPIO_interruptEdgeSelect()`

The GPIO pin state is accessed with

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

## 18.2.2 Function Documentation

### 18.2.2.1 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	106
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	50
IAR 5.51.6	None	76
IAR 5.51.6	Size	62
IAR 5.51.6	Speed	114
MSPGCC 4.8.0	None	198
MSPGCC 4.8.0	Size	62
MSPGCC 4.8.0	Speed	94

#### 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

### 18.2.2.2 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	106
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	50
IAR 5.51.6	None	76
IAR 5.51.6	Size	62
IAR 5.51.6	Speed	114
MSPGCC 4.8.0	None	198
MSPGCC 4.8.0	Size	62
MSPGCC 4.8.0	Speed	94

**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



### 18.2.2.3 GPIO\_enableInterrupt

This function enables the port interrupt on the selected pin. Note that only Port 1, 2, A have this capability. Does not clear interrupt flags.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	106
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	50
IAR 5.51.6	None	76
IAR 5.51.6	Size	58
IAR 5.51.6	Speed	114
MSPGCC 4.8.0	None	178
MSPGCC 4.8.0	Size	62
MSPGCC 4.8.0	Speed	94

**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

### 18.2.2.4 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	210
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. 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:**

One of the following:

- GPIO\_INPUT\_PIN\_HIGH
  - GPIO\_INPUT\_PIN\_LOW
- indicating the input value of the pin

### 18.2.2.5 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	110
CCS 4.2.1	Size	48
CCS 4.2.1	Speed	52
IAR 5.51.6	None	94
IAR 5.51.6	Size	60
IAR 5.51.6	Speed	132
MSPGCC 4.8.0	None	160
MSPGCC 4.8.0	Size	68
MSPGCC 4.8.0	Speed	104

#### 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]

### 18.2.2.6 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. Note that only Port 1, 2, A have this capability.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	136
CCS 4.2.1	Size	58
CCS 4.2.1	Speed	58
IAR 5.51.6	None	92
IAR 5.51.6	Size	50
IAR 5.51.6	Speed	112
MSPGCC 4.8.0	None	268
MSPGCC 4.8.0	Size	70
MSPGCC 4.8.0	Speed	104

#### 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

### 18.2.2.7 GPIO\_setAsInputPin

This function configures the selected Pin as input pin.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	258
CCS 4.2.1	Size	132
CCS 4.2.1	Speed	132
IAR 5.51.6	None	190
IAR 5.51.6	Size	170
IAR 5.51.6	Speed	256
MSPGCC 4.8.0	None	524
MSPGCC 4.8.0	Size	174
MSPGCC 4.8.0	Speed	148

**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 and bits of **PxSEL** register.

**Returns:**

None

### 18.2.2.8 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	296
CCS 4.2.1	Size	150
CCS 4.2.1	Speed	150
IAR 5.51.6	None	216
IAR 5.51.6	Size	180
IAR 5.51.6	Speed	298
MSPGCC 4.8.0	None	620
MSPGCC 4.8.0	Size	200
MSPGCC 4.8.0	Speed	174

**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

### 18.2.2.9 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	296
CCS 4.2.1	Size	150
CCS 4.2.1	Speed	150
IAR 5.51.6	None	216
IAR 5.51.6	Size	180
IAR 5.51.6	Speed	298
MSPGCC 4.8.0	None	600
MSPGCC 4.8.0	Size	200
MSPGCC 4.8.0	Speed	174

**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

### 18.2.2.10 GPIO\_setAsOutputPin

This function configures the selected Pin as output pin.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	216
CCS 4.2.1	Size	114
CCS 4.2.1	Speed	114
IAR 5.51.6	None	160
IAR 5.51.6	Size	128
IAR 5.51.6	Speed	218
MSPGCC 4.8.0	None	380
MSPGCC 4.8.0	Size	164
MSPGCC 4.8.0	Speed	138



**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

### 18.2.2.11 GPIO\_setAsPeripheralModuleFunctionInputPin

This function configures the peripheral module function in input direction for the selected pin.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	216
CCS 4.2.1	Size	114
CCS 4.2.1	Speed	114
IAR 5.51.6	None	160
IAR 5.51.6	Size	120
IAR 5.51.6	Speed	220
MSPGCC 4.8.0	None	380
MSPGCC 4.8.0	Size	164
MSPGCC 4.8.0	Speed	138

**Prototype:**

```
void
GPIO_setAsPeripheralModuleFunctionInputPin(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

### 18.2.2.12 GPIO\_setAsPeripheralModuleFunctionOutputPin

This function configures the peripheral module function in output direction for the selected pin.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	216
CCS 4.2.1	Size	114
CCS 4.2.1	Speed	114
IAR 5.51.6	None	160
IAR 5.51.6	Size	128
IAR 5.51.6	Speed	216
MSPGCC 4.8.0	None	360
MSPGCC 4.8.0	Size	164
MSPGCC 4.8.0	Speed	138

#### Prototype:

```
void
GPIO_setAsPeripheralModuleFunctionOutputPin(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

### 18.2.2.13 GPIO\_setDriveStrength

This function sets the drive strength for the selected port pin.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	178
CCS 4.2.1	Size	106
CCS 4.2.1	Speed	106
IAR 5.51.6	None	120
IAR 5.51.6	Size	80
IAR 5.51.6	Speed	142
MSPGCC 4.8.0	None	336
MSPGCC 4.8.0	Size	100
MSPGCC 4.8.0	Speed	128

**Prototype:**

```
void
GPIO_setDriveStrength(uint8_t selectedPort,
                     uint16_t selectedPins,
                     uint8_t driveStrength)
```

**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

**driveStrength** specifies the drive strength of the pin Valid values are:

- GPIO\_REDUCED\_OUTPUT\_DRIVE\_STRENGTH
- GPIO\_FULL\_OUTPUT\_DRIVE\_STRENGTH

**Description:**

Modified bits of **PxDS** register.

**Returns:**

None

## 18.2.2.14 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	192
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

### 18.2.2.15 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	118
IAR 5.51.6	Speed	192
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

## 18.2.2.16 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	190
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

## 18.3 Programming Example

The following example shows how to use the GPIO API.



```
// Set P1.0 to output direction
GPIO_setAsOutputPin(GPIO_PORT_P1,
                    GPIO_PIN0
                    );

// Set P1.4 to input direction
GPIO_setAsInputPin(GPIO_PORT_P1,
                   GPIO_PIN4
                   );

while (1)
{
    // Test P1.4
    if(GPIO_INPUT_PIN_HIGH == GPIO_getInputPinValue(
        GPIO_PORT_P1,
        GPIO_PIN4
        ))
    {
        // if P1.4 set, set P1.0
        GPIO_setOutputHighOnPin(
            GPIO_PORT_P1,
            GPIO_PIN0
            );
    }
    else
    {
        // else reset
        GPIO_setOutputLowOnPin(
            GPIO_PORT_P1,
            GPIO_PIN0
            );
    }
}
```

# 19 LDO-PWR

Introduction .....	206
API Functions .....	206
Programming Example .....	214

## 19.1 Introduction

The features of the LDO-PWR module include:

- Integrated 3.3-V LDO regulator with sufficient output to power the entire MSP430? microcontroller and system circuitry from 5-V external supply
- Current-limiting capability on 3.3-V LDO output with detection flag and interrupt generation
- LDO input voltage detection flag and interrupt generation

The LDO-PWR power system incorporates an integrated 3.3-V LDO regulator that allows the entire MSP430 microcontroller to be powered from nominal 5-V LDO when it is made available from the system. Alternatively, the power system can supply power only to other components within the system, or it can be unused altogether.

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

## 19.2 API Functions

### Functions

- void [LDOPWR\\_clearInterruptStatus](#) (uint32\_t baseAddress, uint16\_t mask)
- void [LDOPWR\\_disable](#) (uint32\_t baseAddress)
- void [LDOPWR\\_disableInterrupt](#) (uint32\_t baseAddress, uint16\_t mask)
- void [LDOPWR\\_disableOverloadAutoOff](#) (uint32\_t baseAddress)
- void [LDOPWR\\_disablePort\\_U\\_inputs](#) (uint32\_t baseAddress)
- void [LDOPWR\\_disablePort\\_U\\_outputs](#) (uint32\_t baseAddress)
- void [LDOPWR\\_enable](#) (uint32\_t baseAddress)
- void [LDOPWR\\_enableInterrupt](#) (uint32\_t baseAddress, uint16\_t mask)
- void [LDOPWR\\_enableOverloadAutoOff](#) (uint32\_t baseAddress)
- void [LDOPWR\\_enablePort\\_U\\_inputs](#) (uint32\_t baseAddress)
- void [LDOPWR\\_enablePort\\_U\\_outputs](#) (uint32\_t baseAddress)
- uint8\_t [LDOPWR\\_getInterruptStatus](#) (uint32\_t baseAddress, uint16\_t mask)
- uint8\_t [LDOPWR\\_getOverloadAutoOffStatus](#) (uint32\_t baseAddress)
- uint8\_t [LDOPWR\\_getPort\\_U0\\_inputData](#) (uint32\_t baseAddress)
- uint8\_t [LDOPWR\\_getPort\\_U0\\_outputData](#) (uint32\_t baseAddress)
- uint8\_t [LDOPWR\\_getPort\\_U1\\_inputData](#) (uint32\_t baseAddress)
- uint8\_t [LDOPWR\\_getPort\\_U1\\_outputData](#) (uint32\_t baseAddress)
- uint8\_t [LDOPWR\\_isLDOInputValid](#) (uint32\_t baseAddress)
- void [LDOPWR\\_lockConfiguration](#) (uint32\_t baseAddress)
- void [LDOPWR\\_setPort\\_U0\\_outputData](#) (uint32\_t baseAddress, uint8\_t value)
- void [LDOPWR\\_setPort\\_U1\\_outputData](#) (uint32\_t baseAddress, uint8\_t value)
- void [LDOPWR\\_togglePort\\_U0\\_outputData](#) (uint32\_t baseAddress)
- void [LDOPWR\\_togglePort\\_U1\\_outputData](#) (uint32\_t baseAddress)
- void [LDOPWR\\_unlockConfiguration](#) (uint32\_t baseAddress)

## 19.2.1 Detailed Description

The LDOPWR configuration is handled by

- LDOPWR\_unlockConfiguration()
- LDOPWR\_lockConfiguration()
- LDOPWR\_enablePort\_U\_inputs()
- LDOPWR\_disablePort\_U\_inputs()
- LDOPWR\_enablePort\_U\_outputs()
- LDOPWR\_disablePort\_U\_outputs()
- LDOPWR\_enable()
- LDOPWR\_disable()
- LDOPWR\_enableOverloadAutoOff()
- LDOPWR\_disableOverloadAutoOff()

Handling the read/write of output data is handled by

- LDOPWR\_getPort\_U1\_inputData()
- LDOPWR\_getPort\_U0\_inputData()
- LDOPWR\_getPort\_U1\_outputData()
- LDOPWR\_getPort\_U0\_outputData()
- LDOPWR\_getOverloadAutoOffStatus()
- LDOPWR\_setPort\_U0\_outputData()
- LDOPWR\_togglePort\_U1\_outputData()
- LDOPWR\_togglePort\_U0\_outputData()
- LDOPWR\_setPort\_U1\_outputData()

The interrupt and status operations are handled by

- LDOPWR\_enableInterrupt()
- LDOPWR\_disableInterrupt()
- LDOPWR\_getInterruptStatus()
- LDOPWR\_clearInterruptStatus()
- LDOPWR\_isLDOInputValid()
- LDOPWR\_getOverloadAutoOffStatus()

## 19.2.2 Function Documentation

### 19.2.2.1 LDOPWR\_clearInterruptStatus

Clears the interrupt status of LDO-PWR module interrupts.

**Prototype:**

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

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**mask** mask of interrupts to clear the status of Mask value is the logical OR of any of the following:

- LDOPWR\_LDO\_VOLTAGE\_GOING\_OFF\_INTERRUPT
- LDOPWR\_LDO\_VOLTAGE\_COMING\_ON\_INTERRUPT
- LDOPWR\_LDO\_OVERLOAD\_INDICATION\_INTERRUPT

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

**Returns:**  
None

### 19.2.2.2 LDOPWR\_disable

Disables LDO-PWR module.

**Prototype:**  

```
void  
LDOPWR_disable(uint32_t baseAddress)
```

**Parameters:**  
**baseAddress** is the base address of the LDOPWR module.

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

**Returns:**  
None

### 19.2.2.3 LDOPWR\_disableInterrupt

Disables LDO-PWR module interrupts.

**Prototype:**  

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

**Parameters:**  
**baseAddress** is the base address of the LDOPWR module.  
**mask** mask of interrupts to disable Mask value is the logical OR of any of the following:  
■ **LDOPWR\_LDO\_VOLTAGE\_GOING\_OFF\_INTERRUPT**  
■ **LDOPWR\_LDO\_VOLTAGE\_COMING\_ON\_INTERRUPT**  
■ **LDOPWR\_LDO\_OVERLOAD\_INDICATION\_INTERRUPT**

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

**Returns:**  
None

### 19.2.2.4 LDOPWR\_disableOverloadAutoOff

Disables the LDO overload auto-off.

**Prototype:**  

```
void  
LDOPWR_disableOverloadAutoOff(uint32_t baseAddress)
```

**Parameters:**  
**baseAddress** is the base address of the LDOPWR module.

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

**Returns:**  
None

### 19.2.2.5 LDOPWR\_disablePort\_U\_inputs

Disables Port U inputs.

**Prototype:**

```
void  
LDOPWR_disablePort_U_inputs(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Description:**

Modified bits of **PUCTL** register.

**Returns:**

None

### 19.2.2.6 LDOPWR\_disablePort\_U\_outputs

Disables Port U inputs.

**Prototype:**

```
void  
LDOPWR_disablePort_U_outputs(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Description:**

Modified bits of **PUCTL** register.

**Returns:**

None

### 19.2.2.7 LDOPWR\_enable

Enables LDO-PWR module.

**Prototype:**

```
void  
LDOPWR_enable(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Description:**

Modified bits of **LDOPWRCTL** register.

**Returns:**

None

### 19.2.2.8 LDOPWR\_enableInterrupt

Enables LDO-PWR module interrupts.

**Prototype:**

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

**Description:**

Does not clear interrupt flags.

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**mask** mask of interrupts to enable Mask value is the logical OR of any of the following:

- LDOPWR\_LDOI\_VOLTAGE\_GOING\_OFF\_INTERRUPT
- LDOPWR\_LDOI\_VOLTAGE\_COMING\_ON\_INTERRUPT
- LDOPWR\_LDO\_OVERLOAD\_INDICATION\_INTERRUPT

Modified bits of **LDOPWRCTL** register.

**Returns:**

None

### 19.2.2.9 LDOPWR\_enableOverloadAutoOff

Enables the LDO overload auto-off.

**Prototype:**

```
void
LDOPWR_enableOverloadAutoOff(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Description:**

Modified bits of **LDOPWRCTL** register.

**Returns:**

None

### 19.2.2.10 LDOPWR\_enablePort\_U\_inputs

Enables Port U inputs.

**Prototype:**

```
void
LDOPWR_enablePort_U_inputs(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Description:**

Modified bits of **PUCTL** register.

**Returns:**

None

### 19.2.2.11 LDOPWR\_enablePort\_U\_outputs

Enables Port U outputs.

**Prototype:**

```
void
LDOPWR_enablePort_U_outputs(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Description:**

Modified bits of **PUCTL** register.

**Returns:**

None

### 19.2.2.12 LDOPWR\_getInterruptStatus

Returns the interrupt status of LDO-PWR module interrupts.

**Prototype:**

```
uint8_t
LDOPWR_getInterruptStatus(uint32_t baseAddress,
                          uint16_t mask)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**mask** mask of interrupts to get the status of Mask value is the logical OR of any of the following:

- LDOPWR\_LDO\_VOLTAGE\_GOING\_OFF\_INTERRUPT
- LDOPWR\_LDO\_VOLTAGE\_COMING\_ON\_INTERRUPT
- LDOPWR\_LDO\_OVERLOAD\_INDICATION\_INTERRUPT

**Returns:**

Logical OR of any of the following:

- LDOPWR\_LDO\_VOLTAGE\_GOING\_OFF\_INTERRUPT
  - LDOPWR\_LDO\_VOLTAGE\_COMING\_ON\_INTERRUPT
  - LDOPWR\_LDO\_OVERLOAD\_INDICATION\_INTERRUPT
- indicating the status of the masked interrupts

### 19.2.2.13 uint8\_t LDOPWR\_getOverloadAutoOffStatus (uint32\_t baseAddress)

Returns if the LDO overloading auto-off is enabled or disabled.

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Returns:**

One of the following:

- LDOPWR\_AUTOOFF\_ENABLED
- LDOPWR\_AUTOOFF\_DISABLED

### 19.2.2.14 uint8\_t LDOPWR\_getPort\_U0\_inputData (uint32\_t baseAddress)

Returns PU.0 input data.

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Returns:**

One of the following:

- LDOPWR\_PORTU\_PIN\_HIGH
- LDOPWR\_PORTU\_PIN\_LOW

#### 19.2.2.15 uint8\_t LDOPWR\_getPort\_U0\_outputData (uint32\_t *baseAddress*)

Returns PU.0 output data.

**Parameters:**

***baseAddress*** is the base address of the LDOPWR module.

**Returns:**

One of the following:

- LDOPWR\_PORTU\_PIN\_HIGH
- LDOPWR\_PORTU\_PIN\_LOW

#### 19.2.2.16 uint8\_t LDOPWR\_getPort\_U1\_inputData (uint32\_t *baseAddress*)

Returns PU.1 input data.

**Parameters:**

***baseAddress*** is the base address of the LDOPWR module.

**Returns:**

One of the following:

- LDOPWR\_PORTU\_PIN\_HIGH
- LDOPWR\_PORTU\_PIN\_LOW

#### 19.2.2.17 uint8\_t LDOPWR\_getPort\_U1\_outputData (uint32\_t *baseAddress*)

Returns PU.1 output data.

**Parameters:**

***baseAddress*** is the base address of the LDOPWR module.

**Returns:**

One of the following:

- LDOPWR\_PORTU\_PIN\_HIGH
- LDOPWR\_PORTU\_PIN\_LOW

#### 19.2.2.18 uint8\_t LDOPWR\_isLDOInputValid (uint32\_t *baseAddress*)

Returns if the the LDOI is valid and within bounds.

**Parameters:**

***baseAddress*** is the base address of the LDOPWR module.

**Returns:**

One of the following:

- LDOPWR\_LDO\_INPUT\_VALID
- LDOPWR\_LDO\_INPUT\_INVALID



### 19.2.2.19 void LDOPWR\_lockConfiguration (uint32\_t *baseAddress*)

Locks the configuration registers and disables write access.

**Parameters:**

***baseAddress*** is the base address of the LDOPWR module.

**Description:**

Modified bits of **LDOKEYPID** register.

**Returns:**

None

### 19.2.2.20 LDOPWR\_setPort\_U0\_outputData

Sets PU.0 output data.

**Prototype:**

```
void  
LDOPWR_setPort_U0_outputData (uint32_t baseAddress,  
                               uint8_t value)
```

**Parameters:**

***baseAddress*** is the base address of the LDOPWR module.

***value*** Valid values are:

- **LDOPWR\_PORTU\_PIN\_HIGH**
- **LDOPWR\_PORTU\_PIN\_LOW**

**Description:**

Modified bits of **PUCTL** register.

**Returns:**

None

### 19.2.2.21 LDOPWR\_setPort\_U1\_outputData

Sets PU.1 output data.

**Prototype:**

```
void  
LDOPWR_setPort_U1_outputData (uint32_t baseAddress,  
                               uint8_t value)
```

**Parameters:**

***baseAddress*** is the base address of the LDOPWR module.

***value*** Valid values are:

- **LDOPWR\_PORTU\_PIN\_HIGH**
- **LDOPWR\_PORTU\_PIN\_LOW**

**Description:**

Modified bits of **PUCTL** register.

**Returns:**

None

### 19.2.2.22 LDOPWR\_togglePort\_U0\_outputData

Toggles PU.0 output data.

**Prototype:**

```
void  
LDOPWR_togglePort_U0_outputData(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Description:**

Modified bits of **PUCTL** register.

**Returns:**

None

### 19.2.2.23 LDOPWR\_togglePort\_U1\_outputData

Toggles PU.1 output data.

**Prototype:**

```
void  
LDOPWR_togglePort_U1_outputData(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Description:**

Modified bits of **PUCTL** register.

**Returns:**

None

### 19.2.2.24 LDOPWR\_unlockConfiguration

Unlocks the configuration registers and enables write access.

**Prototype:**

```
void  
LDOPWR_unlockConfiguration(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the LDOPWR module.

**Description:**

Modified bits of **LDOKEYPID** register.

**Returns:**

None

## 19.3 Programming Example

The following example shows how to use the LDO-PWR API.

```

{
    // Enable access to config registers
    LDOPWR_unlockConfiguration(LDOPWR_BASE);

    // Configure PU.0 as output pins
    LDOPWR_enablePort_U_outputs(LDOPWR_BASE);

    //Set PU.1 = high
    LDOPWR_setPort_U1_outputData(LDOPWR_BASE,
                                LDOPWR_PORTU_PIN_HIGH
                                );

    //Set PU.0 = low
    LDOPWR_setPort_U0_outputData(LDOPWR_BASE,
                                LDOPWR_PORTU_PIN_LOW
                                );

    // Enable LDO overload indication interrupt
    LDOPWR_enableInterrupt(LDOPWR_BASE,
                           LDOPWR_LDO_OVERLOAD_INDICATION_INTERRUPT
                           );

    // Disable access to config registers
    LDOPWR_lockConfiguration(LDOPWR_BASE);

    // continuous loop
    while(1)
    {
        // Delay
        for(i=50000;i>0;i--);

        // Enable access to config registers
        LDOPWR_unlockConfiguration(LDOPWR_BASE);

        // XOR PU.0/1
        LDOPWR_togglePort_U1_outputData(LDOPWR_BASE);
        LDOPWR_togglePort_U0_outputData(LDOPWR_BASE);

        // Disable access to config registers
        LDOPWR_lockConfiguration(LDOPWR_BASE);
    }
}

//*****
//
// This is the LDO_PWR_VECTOR interrupt vector service routine.
//
//*****
__interrupt void LDOInterruptHandler(void)
{
    if (LDOPWR_getInterruptStatus(LDOPWR_BASE,
                                LDOPWR_LDO_OVERLOAD_INDICATION_INTERRUPT
                                ))

    {
        // Enable access to config registers
        LDOPWR_unlockConfiguration(LDOPWR_BASE);

        // Software clear IFG
        LDOPWR_clearInterruptStatus(LDOPWR_BASE,
                                    LDOPWR_LDO_OVERLOAD_INDICATION_INTERRUPT
                                    );

        // Disable access to config registers
        LDOPWR_lockConfiguration(LDOPWR_BASE);
    }
}

```

```
    // Over load indication; take necessary steps in application firmware
    while(1);
  }
}
```

## 20 32-Bit Hardware Multiplier (MPY32)

Introduction .....	217
API Functions .....	217
Programming Example .....	229

### 20.1 Introduction

The 32-Bit Hardware Multiplier (MPY32) API provides a set of functions for using the MSP430Ware MPY32 modules. Functions are provided to setup the MPY32 modules, set the operand registers, and obtain the results.

The MPY32 Modules does not generate any interrupts.

This driver is contained in `mpy32.c`, with `mpy32.h` containing the API definitions for use by applications.

The 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

### 20.2 API Functions

#### Functions

- `uint16_t MPY32_getCarryBitValue (void)`
- `uint16_t MPY32_getResult16Bit (void)`
- `uint32_t MPY32_getResult24Bit (void)`
- `uint32_t MPY32_getResult32Bit (void)`
- `uint64_t 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)

## 20.2.1 Detailed Description

The MPY32 API is broken into three groups of functions: those that control the settings, those that set the operand registers, and those that return the results, sum extension, and carry bit value.

The settings are handled by

- [MPY32\\_setWriteDelay](#)()
- [MPY32\\_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](#)()

## 20.2.2 Function Documentation

### 20.2.2.1 MPY32\_getCarryBitValue

Returns the Carry Bit of 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.

### 20.2.2.2 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.

### 20.2.2.3 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.

### 20.2.2.4 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.

### 20.2.2.5 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

### 20.2.2.6 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.

### 20.2.2.7 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.

### 20.2.2.8 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

### 20.2.2.9 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

### 20.2.2.10 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

### 20.2.2.11 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

### 20.2.2.12 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

### 20.2.2.13 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

### 20.2.2.14 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

### 20.2.2.15 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

### 20.2.2.16 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

### 20.2.2.17 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

### 20.2.2.18 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

### 20.2.2.19 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

### 20.2.2.20 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 unpredictable 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 **MPYDLYWRNEN** of **MPY32CTL0** register.

**Returns:**

None

## 20.3 Programming Example

The following example shows how to initialize and use the MPY32 API to calculate a 16-bit by 16-bit unsigned multiplication operation.

```
WDT_hold(WDT_A_BASE);    // Stop WDT

// Set a 16-bit Operand into the specific Operand 1 register to specify
// unsigned multiplication
MPY32_setOperandOne16Bit(MPY32_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
```

# 21 Port Mapping Controller

Introduction .....	230
API Functions .....	230
Programming Example .....	231

## 21.1 Introduction

The port mapping controller allows the flexible and re-configurable mapping of digital functions to port pins. The port mapping controller features are:

- Configuration protected by write access key.
- Default mapping provided for each port pin (device-dependent, the device pinout in the device-specific data sheet).
- Mapping can be reconfigured during runtime.
- Each output signal can be mapped to several output pins.

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

The 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	118
CCS 4.2.1	Size	66
CCS 4.2.1	Speed	66
IAR 5.51.6	None	78
IAR 5.51.6	Size	74
IAR 5.51.6	Speed	82
MSPGCC 4.8.0	None	184
MSPGCC 4.8.0	Size	70
MSPGCC 4.8.0	Speed	226

## 21.2 API Functions

### Functions

- void [PMAP\\_configurePorts](#) (uint32\_t baseAddress, const uint8\_t \*portMapping, uint8\_t \*PxMAPy, uint8\_t numberOfPorts, uint8\_t portMapReconfigure)

### 21.2.1 Detailed Description

The MSP430ware API that configures Port Mapping is [PMAP\\_configurePorts\(\)](#)

It needs the following data to configure port mapping. portMapping - pointer to init Data PxMAPy - pointer start of first Port Mapper to initialize numberOfPorts - number of Ports to initialize portMapReconfigure - to enable/disable reconfiguration

## 21.2.2 Function Documentation

### 21.2.2.1 PMAP\_configurePorts

This function configures the MSP430 Port Mapper.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	118
CCS 4.2.1	Size	66
CCS 4.2.1	Speed	66
IAR 5.51.6	None	78
IAR 5.51.6	Size	74
IAR 5.51.6	Speed	82
MSPGCC 4.8.0	None	184
MSPGCC 4.8.0	Size	70
MSPGCC 4.8.0	Speed	226

**Prototype:**

```
void
PMAP_configurePorts(uint32_t baseAddress,
                   const uint8_t *portMapping,
                   uint8_t *PxMAPy,
                   uint8_t numberOfPorts,
                   uint8_t portMapReconfigure)
```

**Description:**

This function port maps a set of pins to a new set.

**Parameters:**

**baseAddress** is the base address of the PMAP control module.

**portMapping** is the pointer to init Data

**PxMAPy** is the pointer start of first PMAP to initialize

**numberOfPorts** is the number of Ports to initialize

**portMapReconfigure** is used to enable/disable reconfiguration Valid values are:

- PMAP\_ENABLE\_RECONFIGURATION
- PMAP\_DISABLE\_RECONFIGURATION [Default]

Modified bits of **PMAPKETID** register and bits of **PMAPCTL** register.

**Returns:**

None

## 21.3 Programming Example

The following example shows some Port Mapping Controller operations using the APIs

```
const unsigned char port_mapping[] = {
    //Port P4:
    PM_TB0CCR0A,
    PM_TB0CCR1A,
    PM_TB0CCR2A,
    PM_TB0CCR3A,
    PM_TB0CCR4A,
    PM_TB0CCR5A,
    PM_TB0CCR6A,
```

```
        PM_NONE
    };

    //CONFIGURE PORTS- pass the port_mapping array, start @ P4MAP01, initialize
    //a single port, do not allow run-time reconfiguration of port mapping

    PMAP_configurePorts(P4MAP_BASE,
        (const unsigned char *)port_mapping,
        (unsigned char *)&P4MAP01,
        1,
        PMAP_DISABLE_RECONFIGURATION
    );
```

## 22 Power Management Module (PMM)

Introduction .....	233
API Functions .....	234
Programming Example .....	245

### 22.1 Introduction

The PMM manages the following internal circuitry:

- An integrated low-dropout voltage regulator (LDO) that produces a secondary core voltage (VCORE) from the primary voltage that is applied to the device (DVCC)
- Supply voltage supervisors (SVS) and supply voltage monitors (SVM) for the primary voltage (DVCC) and the secondary voltage (VCORE). The SVS and SVM include programmable threshold levels and power-fail indicators. Therefore, the PMM plays a crucial role in defining the maximum performance, valid voltage conditions, and current consumption for an application running on an MSP430x5xx or MSP430x6xx device. The secondary voltage that is generated by the integrated LDO, VCORE, is programmable to one of four core voltage levels, shown as 0, 1, 2, and 3. Each increase in VCORE allows the CPU to operate at a higher maximum frequency. The values of these frequencies are specified in the device-specific data sheet. This feature allows the user the flexibility to trade power consumption in active and low-power modes for different degrees of maximum performance and minimum supply voltage.

NOTE: To align with the nomenclature in the MSP430x5xx/MSP430x6xx Family User's Guide, the primary voltage domain (DVCC) is referred to as the high-side voltage (SvsH/SVMH) and the secondary voltage domain (VCORE) is referred to as the low-side voltage (SvsL/SvmL).

Moving between the different VCORE voltages requires a specific sequence of events and can be done only one level at a time; for example, to change from level 0 to level 3, the application code must step through level 1 and level 2.

VCORE increase: 1. SvmL monitor level is incremented. 2. VCORE level is incremented. 3. The SvmL Level Reached Interrupt Flag (SVSMLVLRIIFG) in the PMMIFG register is polled. When asserted, SVSMLVLRIIFG indicates that the VCORE voltage has reached its next level. 4. SvsL is increased. SvsL is changed last, because if SVSL were incremented prior to VCORE, it would potentially cause a reset.

VCORE decrease: 5. Decrement SvmL and SVSL levels. 6. Decrement VCORE. The [PMM\\_setVCore\(\)](#) function appropriately handles an increase or decrease of the core voltage. NOTE: The procedure recommended above provides a workaround for the erratum FLASH37. See the device-specific erratasheet to determine if a device is affected by FLASH37. The workaround is also highlighted in the source code for the PMM library

**Recommended SVS and SVM Settings** The SVS and SVM on both the high side and the low side are enabled in normal performance mode following a brown-out reset condition. The device is held in reset until the SVS and SVM verify that the external and core voltages meet the minimum requirements of the default core voltage, which is level zero. The SVS and SVM remain enabled unless disabled by the firmware. The low-side SVS and SVM are useful for verifying the startup conditions and for verifying any modification to the core voltage. However, in their default mode, they prevent the CPU from executing code on wake-up from low-power modes 2, 3, and 4 for a full 150  $\mu$ s, not 5  $\mu$ s. This is because, in their default states, the SVSL and SvmL are powered down in the low-power mode of the PMM and need time for their comparators to wake and stabilize before they can verify the voltage condition and release the CPU for execution. Note that the high-side SVS and SVM do not influence the wake time from low-power modes. If the wake-up from low-power modes needs to be shortened to 5  $\mu$ s, the SVSL and SvmL should be disabled after the initialization of the core voltage at the beginning of the application. Disabling SVSL and SvmL prevents them from gating the CPU on wake-up from LPM2, LPM3, and LPM4. The application is still protected on the high side with SvsH and SVMH. The [PMM\\_setVCore\(\)](#) function automatically enables and disables the SVS and SVM as necessary if a non-zero core voltage level is required. If the application does not require a change in the core voltage (that is, when the target MCLK is less than 8 MHz), the [PMM\\_disableSVLSvmL\(\)](#) and [PMM\\_enableSvsHReset\(\)](#) macros can be used to disable the low-side SVS and SVM circuitry and enable only the high-side SVS POR reset, respectively.

**Setting SVS/SVM Threshold Levels** The voltage thresholds for the SVS and SVM modules are programmable. On the high side, there are two bit fields that control these threshold levels: the SvsHRVL and SVSMHRRRL. The SvsHRVL field defines the voltage threshold at which the SvsH triggers a reset (also known as the SvsH ON voltage level). The SVSMHRRRL field defines the voltage threshold at which the SvsH releases the device from a reset (also known as SvsH OFF voltage level). The MSP430x5xx/MSP430x6xx Family User's Guide (SLAU208) [1] recommends the settings shown in Table 1 when setting these bits. The [PMM\\_setVCore\(\)](#) function follows these recommendations and ensures that the SVS levels match the core voltage levels that are used.

Advanced SVS Controls and Trade-offs In addition to the default SVS settings that are provided with the [PMM\\_setVCore\(\)](#) function, the SVS/SVM modules can be optimized for wake-up speed, response time (propagation delay), and current consumption, as needed. The following controls can be optimized for the SVS/SVM modules:

- Protection in low power modes - LPM2, LPM3, and LPM4
- Wake-up time from LPM2, LPM3, and LPM4
- Response time to react to an SVS event Selecting the LPM option, wake-up time, and response time that is best suited for the application is left to the user. A few typical examples illustrate the trade-offs: Case A: The most robust protection that stays on in LPMs and has the fastest response and wake-up time consumes the most power. Case B: With SVS high side active only in AM, no protection in LPMs, slow wake-up, and slow response time has SVS protection with the least current consumption. Case C: An optimized case is described - turn off the low-side monitor and supervisor, thereby saving power while keeping response time fast on the high side to help with timing critical applications. The user can call the [PMM\\_setVCore\(\)](#) function, which configures SVS/SVM high side and low side with the recommended or default configurations, or can call the APIs provided to control the parameters as the application demands.

Any writes to the SVSMLCTL and SVSMHCTL registers require a delay time for these registers to settle before the new settings take effect. This delay time is dependent on whether the SVS and SVM modules are configured for normal or full performance. See device-specific data sheet for exact delay times.

## 22.2 API Functions

### Functions

- void [PMM\\_clearPMMIFGS](#) (void)
- void [PMM\\_disableSvmH](#) (void)
- void [PMM\\_disableSvmHInterrupt](#) (void)
- void [PMM\\_disableSvmL](#) (void)
- void [PMM\\_disableSvmLInterrupt](#) (void)
- void [PMM\\_disableSvsH](#) (void)
- void [PMM\\_disableSvsHReset](#) (void)
- void [PMM\\_disableSvsHSvmH](#) (void)
- void [PMM\\_disableSvsL](#) (void)
- void [PMM\\_disableSvsLReset](#) (void)
- void [PMM\\_disableSvsLSvmL](#) (void)
- void [PMM\\_enableSvmH](#) (void)
- void [PMM\\_enableSvmHInterrupt](#) (void)
- void [PMM\\_enableSvmL](#) (void)
- void [PMM\\_enableSvmLInterrupt](#) (void)
- void [PMM\\_enableSvsH](#) (void)
- void [PMM\\_enableSvsHReset](#) (void)
- void [PMM\\_enableSvsHSvmH](#) (void)
- void [PMM\\_enableSvsL](#) (void)
- void [PMM\\_enableSvsLReset](#) (void)
- void [PMM\\_enableSvsLSvmL](#) (void)
- uint16\_t [PMM\\_getInterruptStatus](#) (uint16\_t mask)
- bool [PMM\\_setVCore](#) (uint8\_t level)
- uint16\_t [PMM\\_setVCoreDown](#) (uint8\_t level)
- uint16\_t [PMM\\_setVCoreUp](#) (uint8\_t level)
- void [PMM\\_SvsHDisabledInLPMFullPerf](#) (void)
- void [PMM\\_SvsHDisabledInLPMNormPerf](#) (void)
- void [PMM\\_SvsHEnabledInLPMFullPerf](#) (void)
- void [PMM\\_SvsHEnabledInLPMNormPerf](#) (void)
- void [PMM\\_SvsHOptimizedInLPMFullPerf](#) (void)
- void [PMM\\_SvsLDisabledInLPMFastWake](#) (void)

- void [PMM\\_SvsLDisabledInLPMSlowWake](#) (void)
- void [PMM\\_SvsLEnabledInLPMFastWake](#) (void)
- void [PMM\\_SvsLEnabledInLPMSlowWake](#) (void)
- void [PMM\\_SvsLOptimizedInLPMFastWake](#) (void)

## 22.2.1 Detailed Description

[PMM\\_enableSvsL\(\)](#) / [PMM\\_disableSvsL\(\)](#) Enables or disables the low-side SVS circuitry

[PMM\\_enableSvmL\(\)](#) / [PMM\\_disableSvmL\(\)](#) Enables or disables the low-side SVM circuitry

[PMM\\_enableSvsH\(\)](#) / [PMM\\_disableSvsH\(\)](#) Enables or disables the high-side SVS circuitry

[PMM\\_enableSVMH\(\)](#) / [PMM\\_disableSVMH\(\)](#) Enables or disables the high-side SVM circuitry

[PMM\\_enableSvsLSvmL\(\)](#) / [PMM\\_disableSvsLSvmL\(\)](#) Enables or disables the low-side SVS and SVM circuitry

[PMM\\_enableSvsHSvmH\(\)](#) / [PMM\\_disableSvsHSvmH\(\)](#) Enables or disables the high-side SVS and SVM circuitry

[PMM\\_enableSvsLReset\(\)](#) / [PMM\\_disableSvsLReset\(\)](#) Enables or disables the POR signal generation when a low-voltage event is registered by the low-side SVS

[PMM\\_enableSvmLInterrupt\(\)](#) / [PMM\\_disableSvmLInterrupt\(\)](#) Enables or disables the interrupt generation when a low-voltage event is registered by the low-side SVM

[PMM\\_enableSvsHReset\(\)](#) / [PMM\\_disableSvsHReset\(\)](#) Enables or disables the POR signal generation when a low-voltage event is registered by the high-side SVS

[PMM\\_enableSVMHInterrupt\(\)](#) / [PMM\\_disableSVMHInterrupt\(\)](#) Enables or disables the interrupt generation when a low-voltage event is registered by the high-side SVM

[PMM\\_clearPMMIFGS\(\)](#) Clear all interrupt flags for the PMM

[PMM\\_SvsLEnabledInLPMFastWake\(\)](#) Enables supervisor low side in LPM with twake-up-fast from LPM2, LPM3, and LPM4

[PMM\\_SvsLEnabledInLPMSlowWake\(\)](#) Enables supervisor low side in LPM with twake-up-slow from LPM2, LPM3, and LPM4

[PMM\\_SvsLDisabledInLPMFastWake\(\)](#) Disables supervisor low side in LPM with twake-up-fast from LPM2, LPM3, and LPM4

[PMM\\_SvsLDisabledInLPMSlowWake\(\)](#) Disables supervisor low side in LPM with twake-up-slow from LPM2, LPM3, and LPM4

[PMM\\_SvsHEnabledInLPMNormPerf\(\)](#) Enables supervisor high side in LPM with tpd = 20 ?s(1)

[PMM\\_SvsHEnabledInLPMFullPerf\(\)](#) Enables supervisor high side in LPM with tpd = 2.5 ?s(1)

[PMM\\_SvsHDisabledInLPMNormPerf\(\)](#) Disables supervisor high side in LPM with tpd = 20 ?s(1)

[PMM\\_SvsHDisabledInLPMFullPerf\(\)](#) Disables supervisor high side in LPM with tpd = 2.5 ?s(1)

[PMM\\_SvsLOptimizedInLPMFastWake\(\)](#) Optimized to provide twake-up-fast from LPM2, LPM3, and LPM4 with least power

[PMM\\_SvsHOptimizedInLPMFullPerf\(\)](#) Optimized to provide tpd = 2.5 ?s(1) in LPM with least power

[PMM\\_getInterruptStatus\(\)](#) Returns interrupt status of the PMM module

[PMM\\_setVCore\(\)](#) Sets the appropriate VCore level. Calls the [PMM\\_setVCoreUp\(\)](#) or [PMM\\_setVCoreDown\(\)](#) function the required number of times depending on the current VCore level, because the levels must be stepped through individually. A status indicator equal to STATUS\_SUCCESS or STATUS\_FAIL that indicates a valid or invalid VCore transition, respectively. An invalid VCore transition exists if DVCC is less than the minimum required voltage for the target VCore voltage.

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

## 22.2.2 Function Documentation

### 22.2.2.1 PMM\_clearPMMIFGS

Clear all interrupt flags for the PMM.

**Prototype:**

```
void  
PMM_clearPMMIFGS(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **PMMIFG** register.

**Returns:**

None

### 22.2.2.2 PMM\_disableSvmH

Disables the high-side SVM circuitry.

**Prototype:**

```
void  
PMM_disableSvmH(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

### 22.2.2.3 PMM\_disableSvmHInterrupt

Disables the interrupt generation when a low-voltage event is registered by the high-side SVM.

**Prototype:**

```
void  
PMM_disableSvmHInterrupt(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **PMMIE** register.

**Returns:**

None

### 22.2.2.4 PMM\_disableSvmL

Disables the low-side SVM circuitry.

**Prototype:**

```
void  
PMM_disableSvmL(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None



### 22.2.2.5 PMM\_disableSvmLInterrupt

Disables the interrupt generation when a low-voltage event is registered by the low-side SVM.

**Prototype:**

```
void  
PMM_disableSvmLInterrupt(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **PMMIE** register.

**Returns:**

None

### 22.2.2.6 PMM\_disableSvsH

Disables the high-side SVS circuitry.

**Prototype:**

```
void  
PMM_disableSvsH(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

### 22.2.2.7 PMM\_disableSvsHReset

Disables the POR signal generation when a low-voltage event is registered by the high-side SVS.

**Prototype:**

```
void  
PMM_disableSvsHReset(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **PMMIE** register.

**Returns:**

None

### 22.2.2.8 PMM\_disableSvsHSvmH

Disables the high-side SVS and SVM circuitry.

**Prototype:**

```
void  
PMM_disableSvsHSvmH(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

#### 22.2.2.9 PMM\_disableSvsL

Disables the low-side SVS circuitry.

**Prototype:**

```
void  
PMM_disableSvsL(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

#### 22.2.2.10 PMM\_disableSvsLReset

Disables the POR signal generation when a low-voltage event is registered by the low-side SVS.

**Prototype:**

```
void  
PMM_disableSvsLReset(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **PMMIE** register.

**Returns:**

None

#### 22.2.2.11 PMM\_disableSvsLSvmL

Disables the low-side SVS and SVM circuitry.

**Prototype:**

```
void  
PMM_disableSvsLSvmL(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

#### 22.2.2.12 PMM\_enableSvmH

Enables the high-side SVM circuitry.

**Prototype:**

```
void  
PMM_enableSvmH(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

### 22.2.2.13 PMM\_enableSvmHInterrupt

Enables the interrupt generation when a low-voltage event is registered by the high-side SVM.

**Prototype:**

```
void  
PMM_enableSvmHInterrupt(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **PMMIE** register.

**Returns:**

None

### 22.2.2.14 PMM\_enableSvmL

Enables the low-side SVM circuitry.

**Prototype:**

```
void  
PMM_enableSvmL(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

### 22.2.2.15 PMM\_enableSvmLInterrupt

Enables the interrupt generation when a low-voltage event is registered by the low-side SVM.

**Prototype:**

```
void  
PMM_enableSvmLInterrupt(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **PMMIE** register.

**Returns:**

None

### 22.2.2.16 PMM\_enableSvsH

Enables the high-side SVS circuitry.

**Prototype:**

```
void  
PMM_enableSvsH(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

### 22.2.2.17 PMM\_enableSvsHReset

Enables the POR signal generation when a low-voltage event is registered by the high-side SVS.

**Prototype:**

```
void  
PMM_enableSvsHReset(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **PMMIE** register.

**Returns:**

None

### 22.2.2.18 PMM\_enableSvsHSvmH

Enables the high-side SVS and SVM circuitry.

**Prototype:**

```
void  
PMM_enableSvsHSvmH(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

### 22.2.2.19 PMM\_enableSvsL

Enables the low-side SVS circuitry.

**Prototype:**

```
void  
PMM_enableSvsL(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

### 22.2.2.20 PMM\_enableSvsLReset

Enables the POR signal generation when a low-voltage event is registered by the low-side SVS.

**Prototype:**

```
void  
PMM_enableSvsLReset(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **PMMIE** register.

**Returns:**

None

### 22.2.2.21 PMM\_enableSvsLSvmL

Enables the low-side SVS and SVM circuitry.

**Prototype:**

```
void
PMM_enableSvsLSvmL(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

### 22.2.2.22 PMM\_getInterruptStatus

Returns interrupt status.

**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\_SVSMLDLYIFG
- PMM\_SVMLIFG
- PMM\_SVMLVLRIFG
- PMM\_SVSMHDLYIFG
- PMM\_SVMHIFG
- PMM\_SVMHVLRFIFG
- PMM\_PMMBORIFG
- PMM\_PMMRSTIFG
- PMM\_PMPORIFG
- PMM\_SVSHIFG
- PMM\_SVSLIFG
- PMM\_PMMLPM5IFG

**Returns:**

Logical OR of any of the following:

- PMM\_SVSMLDLYIFG
- PMM\_SVMLIFG
- PMM\_SVMLVLRIFG
- PMM\_SVSMHDLYIFG
- PMM\_SVMHIFG
- PMM\_SVMHVLRFIFG
- PMM\_PMMBORIFG
- PMM\_PMMRSTIFG
- PMM\_PMPORIFG
- PMM\_SVSHIFG
- PMM\_SVSLIFG
- PMM\_PMMLPM5IFG

indicating the status of the masked interrupts

### 22.2.2.23 bool PMM\_setVCore (uint8\_t level)

Set Vcore to expected level.

**Parameters:**

**level** level to which Vcore needs to be decreased/increased Valid values are:

- PMM\_CORE\_LEVEL\_0 [Default]
- PMM\_CORE\_LEVEL\_1
- PMM\_CORE\_LEVEL\_2
- PMM\_CORE\_LEVEL\_3

**Description:**

Modified bits of **PMMCTL0** register, bits of **PMMIFG** register, bits of **PMMRIE** register, bits of **SVSMHCTL** register and bits of **SVSMLCTL** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL

### 22.2.2.24 PMM\_setVCoreDown

Decrease Vcore by one level.

**Prototype:**

```
uint16_t
PMM_setVCoreDown (uint8_t level)
```

**Parameters:**

**level** level to which Vcore needs to be decreased Valid values are:

- PMM\_CORE\_LEVEL\_0 [Default]
- PMM\_CORE\_LEVEL\_1
- PMM\_CORE\_LEVEL\_2
- PMM\_CORE\_LEVEL\_3

**Description:**

Modified bits of **PMMCTL0** register, bits of **PMMIFG** register, bits of **PMMRIE** register, bits of **SVSMHCTL** register and bits of **SVSMLCTL** register.

**Returns:**

STATUS\_SUCCESS

### 22.2.2.25 PMM\_setVCoreUp

Increase Vcore by one level.

**Prototype:**

```
uint16_t
PMM_setVCoreUp (uint8_t level)
```

**Parameters:**

**level** level to which Vcore needs to be increased Valid values are:

- PMM\_CORE\_LEVEL\_0 [Default]
- PMM\_CORE\_LEVEL\_1
- PMM\_CORE\_LEVEL\_2
- PMM\_CORE\_LEVEL\_3

**Description:**

Modified bits of **PMMCTL0** register, bits of **PMMIFG** register, bits of **PMMRIE** register, bits of **SVSMHCTL** register and bits of **SVSMLCTL** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL

#### 22.2.2.26 PMM\_SvsHDisabledInLPMFullPerf

Disables supervisor high side in LPM with  $t_{pd} = 2.5 \mu s(1)$ .

**Prototype:**

```
void  
PMM_SvsHDisabledInLPMFullPerf(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

#### 22.2.2.27 PMM\_SvsHDisabledInLPMNormPerf

Disables supervisor high side in LPM with  $t_{pd} = 20 \mu s(1)$ .

**Prototype:**

```
void  
PMM_SvsHDisabledInLPMNormPerf(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

#### 22.2.2.28 PMM\_SvsHEnabledInLPMFullPerf

Enables supervisor high side in LPM with  $t_{pd} = 2.5 \mu s(1)$ .

**Prototype:**

```
void  
PMM_SvsHEnabledInLPMFullPerf(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

#### 22.2.2.29 PMM\_SvsHEnabledInLPMNormPerf

Enables supervisor high side in LPM with  $t_{pd} = 20 \mu s(1)$ .

**Prototype:**

```
void  
PMM_SvsHEnabledInLPMNormPerf(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMHCTL** register.

**Returns:**

None

### 22.2.2.30 PMM\_SvsHOptimizedInLPMFullPerf

Optimized to provide  $t_{pd} = 2.5 \mu s(1)$  in LPM with least power.

**Prototype:**

```
void  
PMM_SvsHOptimizedInLPMFullPerf(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

### 22.2.2.31 PMM\_SvsLDisabledInLPMFastWake

Disables supervisor low side in LPM with twake-up-fast from LPM2, LPM3, and LPM4.

**Prototype:**

```
void  
PMM_SvsLDisabledInLPMFastWake(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

### 22.2.2.32 PMM\_SvsLDisabledInLPMSlowWake

Disables supervisor low side in LPM with twake-up-slow from LPM2, LPM3, and LPM4.

**Prototype:**

```
void  
PMM_SvsLDisabledInLPMSlowWake(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

### 22.2.2.33 PMM\_SvsLEnabledInLPMFastWake

Enables supervisor low side in LPM with twake-up-fast from LPM2, LPM3, and LPM4.

**Prototype:**

```
void  
PMM_SvsLEnabledInLPMFastWake(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None



### 22.2.2.34 PMM\_SvsLEnabledInLPMSlowWake

Enables supervisor low side in LPM with twake-up-slow from LPM2, LPM3, and LPM4.

**Prototype:**

```
void
PMM_SvsLEnabledInLPMSlowWake(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

### 22.2.2.35 PMM\_SvsLOptimizedInLPMFastWake

Optimized to provide twake-up-fast from LPM2, LPM3, and LPM4 with least power.

**Prototype:**

```
void
PMM_SvsLOptimizedInLPMFastWake(void)
```

**Description:**

Modified bits of **PMMCTL0** register and bits of **SVSMLCTL** register.

**Returns:**

None

## 22.3 Programming Example

The following example shows some pmm operations using the APIs

```
//Use the line below to bring the level back to 0
status = PMM_setVCore(PMM_BASE,
    PMMCOREV_0
);

//Set P1.0 to output direction
GPIO_setAsOutputPin(
    GPIO_PORT_P1,
    GPIO_PIN0
);

//continuous loop
while (1)
{
    //Toggle P1.0
    GPIO_toggleOutputOnPin(
        GPIO_PORT_P1,
        GPIO_PIN0
    );
    //Delay
    __delay_cycles(20000);
}
```

## 23 RAM Controller

Introduction .....	246
API Functions .....	246
Programming Example .....	248

### 23.1 Introduction

The RAMCTL provides access to the different power modes of the RAM. The RAMCTL allows the ability to reduce the leakage current while the CPU is off. The RAM can also be switched off. In retention mode, the RAM content is saved while the RAM content is lost in off mode. The RAM is partitioned in sectors, typically of 4KB (sector) size. See the device-specific data sheet for actual block allocation and size. Each sector is controlled by the RAM controller RAM Sector Off control bit (RCRSyOFF) of the RAMCTL Control 0 register (RCCTL0). The RCCTL0 register is protected with a key. Only if the correct key is written during a word write, the RCCTL0 register content can be modified. Byte write accesses or write accesses with a wrong key are ignored.

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

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	36
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	22
IAR 5.51.6	None	18
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	20
MSPGCC 4.8.0	Speed	20

### 23.2 API Functions

#### Functions

- `uint8_t RAM_getSectorState (uint8_t sector)`
- `void RAM_setSectorOff (uint8_t sector)`

#### 23.2.1 Detailed Description

The MSP430ware API that configure the RAM controller are:

`RAM_setSectorOff()` - Set specified RAM sector off `RAM_getSectorState()` - Get RAM sector ON/OFF status

## 23.2.2 Function Documentation

### 23.2.2.1 RAM\_getSectorState

Get RAM sector ON/OFF status.

**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
RAM_getSectorState(uint8_t sector)
```

**Parameters:**

**sector** is specified sector Mask value is the logical OR of any of the following:

- RAM\_SECTOR0
- RAM\_SECTOR1
- RAM\_SECTOR2
- RAM\_SECTOR3
- RAM\_SECTOR4
- RAM\_SECTOR5
- RAM\_SECTOR6
- RAM\_SECTOR7

**Description:**

Modified bits of **RCCTL0** register.

**Returns:**

Logical OR of any of the following:

- RAM\_SECTOR0
  - RAM\_SECTOR1
  - RAM\_SECTOR2
  - RAM\_SECTOR3
  - RAM\_SECTOR4
  - RAM\_SECTOR5
  - RAM\_SECTOR6
  - RAM\_SECTOR7
- indicating the status of the masked sectors

### 23.2.2.2 RAM\_setSectorOff

Set specified RAM sector off.

**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	26
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

**Prototype:**

```
void
RAM_setSectorOff(uint8_t sector)
```

**Parameters:**

**sector** is specified sector to be set off. Mask value is the logical OR of any of the following:

- **RAM\_SECTOR0**
- **RAM\_SECTOR1**
- **RAM\_SECTOR2**
- **RAM\_SECTOR3**
- **RAM\_SECTOR4**
- **RAM\_SECTOR5**
- **RAM\_SECTOR6**
- **RAM\_SECTOR7**

**Description:**

Modified bits of **RCCTL0** register.

**Returns:**

None

## 23.3 Programming Example

The following example shows some RAM Controller operations using the APIs

```
//Start timer
Timer_startUpMode(    TIMER_B0_BASE,
    TIMER_CLOCKSOURCE_ACLK,
    TIMER_CLOCKSOURCE_DIVIDER_1,
    25000,
    TIMER_TAIE_INTERRUPT_DISABLE,
    TIMER_CAPTURECOMPARE_INTERRUPT_ENABLE,
    TIMER_DO_CLEAR
);

//RAM controller sector off
RAM_setSectorOff(RAM_BASE,
    RAMCONTROL_SECTOR2
);

//Enter LPM0, enable interrupts
__bis_SR_register(LPM3_bits + GIE);

//For debugger
__no_operation();
}
```

```
//*****  
//  
//This is the Timer B0 interrupt vector service routine.  
//  
//*****  
#pragma vector=TIMERB0_VECTOR  
__interrupt void TIMERB0_ISR (void)  
{  
    returnValue = RAM_getSectorState(RAM_BASE,  
        RAM_SECTOR0 +  
        RAM_SECTOR1 +  
        RAM_SECTOR2 +  
        RAM_SECTOR3);  
}
```

## 24 Internal Reference (REF)

Introduction .....	250
API Functions .....	250
Programming Example .....	258

### 24.1 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 necessarily limited to, the ADC10\_A, ADC12\_A, DAC12\_A, LCD\_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 with 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	262
CCS 4.2.1	Size	100
CCS 4.2.1	Speed	100
IAR 5.51.6	None	116
IAR 5.51.6	Size	78
IAR 5.51.6	Speed	78
MSPGCC 4.8.0	None	400
MSPGCC 4.8.0	Size	86
MSPGCC 4.8.0	Speed	86

### 24.2 API Functions

#### Functions

- void [REF\\_disableReferenceVoltage](#) (uint32\_t baseAddress)
- void [REF\\_disableReferenceVoltageOutput](#) (uint32\_t baseAddress)
- void [REF\\_disableTempSensor](#) (uint32\_t baseAddress)
- void [REF\\_enableReferenceVoltage](#) (uint32\_t baseAddress)
- void [REF\\_enableReferenceVoltageOutput](#) (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)

## 24.2.1 Detailed Description

The DMA 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\\_enableReferenceVoltageOutput\(\)](#)
- [REF\\_disableReferenceVoltageOutput\(\)](#)
- [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\(\)](#)

## 24.2.2 Function Documentation

### 24.2.2.1 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 [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

### 24.2.2.2 REF\_disableReferenceVoltageOutput

Disables the reference voltage as an output to a pin.

**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_disableReferenceVoltageOutput(uint32_t baseAddress)
```

**Description:**

This function is used to disables the reference voltage being generated to be given to an output pin. 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.

**Returns:**

None

### 24.2.2.3 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 **RFETCOFF** of **REFCTL0** register.

**Returns:**

None

#### 24.2.2.4 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

#### 24.2.2.5 REF\_enableReferenceVoltageOutput

Outputs the reference voltage to an output pin.

**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_enableReferenceVoltageOutput(uint32_t baseAddress)
```

**Description:**

This function is used to output the reference voltage being generated to an output pin. Please note, the output pin is device specific. Please note, that giving ADC12 module control over the REF module, the state of the reference voltage as an output to a pin is dependent on the controls of the ADC12 module. Please note, if the [REF\\_isRefGenBusy\(\)](#) returns REF\_BUSY, this function will have no effect.

**Parameters:**

**baseAddress** is the base address of the REF module.

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

**Returns:**

None

### 24.2.2.6 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

### 24.2.2.7 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 indicating the REF bandgap mode

## 24.2.2.8 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	16
CCS 4.2.1	Speed	16
IAR 5.51.6	None	20
IAR 5.51.6	Size	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

**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\_INACTIVE**
- **REF\_ACTIVE**  
indicating the bandgap active status of the REF module

### 24.2.2.9 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	16
CCS 4.2.1	Speed	16
IAR 5.51.6	None	20
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

**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\_INACTIVE**
- **REF\_ACTIVE**  
indicating the REF generator status

### 24.2.2.10 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 any changes to be made to the REF module controls indicating the REF generator busy status

### 24.2.2.11 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

## 24.3 Programming Example

The following example shows how to initialize and use the REF API with the ADC12\_A 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

// Initialize the ADC12_A Module
/*
Base address of ADC12_A Module
Use internal ADC12_A bit as sample/hold signal to start conversion
USE MODOSC 5MHZ Digital Oscillator as clock source
Use default clock divider of 1
*/
ADC12_A_init(ADC12_A_BASE,
             ADC12_A_SAMPLEHOLDSOURCE_SC,
             ADC12_A_CLOCKSOURCE_ADC12OSC,
             ADC12_A_CLOCKDIVIDEBY_1);

/*
Base address of ADC12 Module
For memory buffers 0-7 sample/hold for 64 clock cycles
For memory buffers 8-15 sample/hold for 4 clock cycles (default)
Disable Multiple Sampling
*/
ADC12_A_setupSamplingTimer(ADC12_A_BASE,
                           ADC12_A_CYCLEHOLD_64_CYCLES,
                           ADC12_A_CYCLEHOLD_4_CYCLES,
                           ADC12_A_MULTIPLESAMPLESENABLE);

// Configure Memory Buffer
/*
Base address of the ADC12 Module
Configure memory buffer 0
Map input A0 to memory buffer 0
Vref+ = Vref+ (INT)
Vref- = AVss
*/
ADC12_A_memoryConfigure(ADC12_A_BASE,
                        ADC12_A_MEMORY_0,
                        ADC12_A_INPUT_A0,
                        ADC12_A_VREFPOS_INT,
                        ADC12_A_VREFNEG_AVSS,
                        ADC12_A_NOTENDOFSEQUENCE);

while (1)
{
    // Enable/Start sampling and conversion
    /*
Base address of ADC12 Module
Start the conversion into memory buffer 0
Use the single-channel, single-conversion mode
*/
    ADC12_A_startConversion(ADC12_A_BASE,
                           ADC12_A_MEMORY_0,
                           ADC12_A_SINGLECHANNEL);
}
```

```
// Poll for interrupt on memory buffer 0
while(!ADC12_A_interruptStatus(ADC12_A_BASE, ADC12IFG0));

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

## 25 Real-Time Clock (RTC)

Introduction .....	260
API Functions .....	260
Programming Example .....	276

### 25.1 Introduction

The Real Time Clock (RTC) 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\_A\_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\_A\_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_a.c`, with `rtc_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 with 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	1384
CCS 4.2.1	Size	640
CCS 4.2.1	Speed	632
IAR 5.51.6	None	988
IAR 5.51.6	Size	804
IAR 5.51.6	Speed	804
MSPGCC 4.8.0	None	2446
MSPGCC 4.8.0	Size	848
MSPGCC 4.8.0	Speed	854

### 25.2 API Functions

#### Functions

- void [RTC\\_A\\_calendarInit](#) (uint32\_t baseAddress, Calendar CalendarTime, uint16\_t formatSelect)
- void [RTC\\_A\\_clearInterrupt](#) (uint32\_t baseAddress, uint8\_t interruptFlagMask)
- void [RTC\\_A\\_counterPrescaleHold](#) (uint32\_t baseAddress, uint8\_t prescaleSelect)
- void [RTC\\_A\\_counterPrescaleInit](#) (uint32\_t baseAddress, uint8\_t prescaleSelect, uint16\_t prescaleClockSelect, uint16\_t prescaleDivider)
- void [RTC\\_A\\_counterPrescaleStart](#) (uint32\_t baseAddress, uint8\_t prescaleSelect)
- void [RTC\\_A\\_definePrescaleEvent](#) (uint32\_t baseAddress, uint8\_t prescaleSelect, uint8\_t prescaleEventDivider)
- void [RTC\\_A\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t interruptMask)
- void [RTC\\_A\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t interruptMask)
- Calendar [RTC\\_A\\_getCalendarTime](#) (uint32\_t baseAddress)
- uint32\_t [RTC\\_A\\_getCounterValue](#) (uint32\_t baseAddress)
- uint8\_t [RTC\\_A\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t interruptFlagMask)



- uint8\_t [RTC\\_A\\_getPrescaleValue](#) (uint32\_t baseAddress, uint8\_t prescaleSelect)
- void [RTC\\_A\\_holdClock](#) (uint32\_t baseAddress)
- void [RTC\\_A\\_initCalendar](#) (uint32\_t baseAddress, Calendar \*CalendarTime, uint16\_t formatSelect)
- void [RTC\\_A\\_initCounter](#) (uint32\_t baseAddress, uint16\_t clockSelect, uint16\_t counterSizeSelect)
- void [RTC\\_A\\_setCalendarAlarm](#) (uint32\_t baseAddress, uint8\_t minutesAlarm, uint8\_t hoursAlarm, uint8\_t dayOfWeekAlarm, uint8\_t dayOfMonthAlarm)
- void [RTC\\_A\\_setCalendarEvent](#) (uint32\_t baseAddress, uint16\_t eventSelect)
- void [RTC\\_A\\_setCalibrationData](#) (uint32\_t baseAddress, uint8\_t offsetDirection, uint8\_t offsetValue)
- void [RTC\\_A\\_setCalibrationFrequency](#) (uint32\_t baseAddress, uint16\_t frequencySelect)
- void [RTC\\_A\\_setCounterValue](#) (uint32\_t baseAddress, uint32\_t counterValue)
- void [RTC\\_A\\_setPrescaleCounterValue](#) (uint32\_t baseAddress, uint8\_t prescaleSelect, uint8\_t prescaleCounterValue)
- void [RTC\\_A\\_startClock](#) (uint32\_t baseAddress)

## 25.2.1 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\\_A\\_startClock\(\)](#)
- [RTC\\_A\\_holdClock\(\)](#)
- [RTC\\_A\\_setCalibrationFrequency\(\)](#)
- [RTC\\_A\\_setCalibrationData\(\)](#)

The RTC Calender Mode is initialized and setup by

- [RTC\\_A\\_calenderInit\(\)](#)
- [RTC\\_A\\_getCalenderTime\(\)](#)
- [RTC\\_A\\_getPrescaleValue\(\)](#)
- [RTC\\_A\\_setPrescaleValue\(\)](#)

The RTC Counter Mode is initialized and setup by

- [RTC\\_A\\_counterInit\(\)](#)
- [RTC\\_A\\_getCounterValue\(\)](#)
- [RTC\\_A\\_setCounterValue\(\)](#)
- [RTC\\_A\\_counterPrescaleInit\(\)](#)
- [RTC\\_A\\_counterPrescaleHold\(\)](#)
- [RTC\\_A\\_counterPrescaleStart\(\)](#)
- [RTC\\_A\\_getPrescaleValue\(\)](#)
- [RTC\\_A\\_setPrescaleValue\(\)](#)

The RTC interrupts are handled by

- [RTC\\_A\\_setCalenderAlarm\(\)](#)
- [RTC\\_A\\_setCalenderEvent\(\)](#)
- [RTC\\_A\\_definePrescaleEvent\(\)](#)
- [RTC\\_A\\_enableInterrupt\(\)](#)
- [RTC\\_A\\_disableInterrupt\(\)](#)
- [RTC\\_A\\_getInterruptStatus\(\)](#)
- [RTC\\_A\\_clearInterrupt\(\)](#)

## 25.2.2 Function Documentation

### 25.2.2.1 RTC\_A\_calendarInit

Deprecated - Initializes the settings to operate the RTC in calendar mode.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	130
CCS 4.2.1	Size	84
CCS 4.2.1	Speed	84
IAR 5.51.6	None	108
IAR 5.51.6	Size	104
IAR 5.51.6	Speed	104
MSPGCC 4.8.0	None	232
MSPGCC 4.8.0	Size	150
MSPGCC 4.8.0	Speed	150

#### Prototype:

```
void
RTC_A_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\_A 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\_A\_FORMAT\_BINARY** [Default]
- **RTC\_A\_FORMAT\_BCD**  
Modified bits are **RTCB**CD of **RTCCTL1** register.

#### Returns:

None

### 25.2.2.2 RTC\_A\_clearInterrupt

Clears selected RTC interrupt flags.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	78
CCS 4.2.1	Size	40
CCS 4.2.1	Speed	40
IAR 5.51.6	None	48
IAR 5.51.6	Size	46
IAR 5.51.6	Speed	46
MSPGCC 4.8.0	None	176
MSPGCC 4.8.0	Size	40
MSPGCC 4.8.0	Speed	40

**Prototype:**

```
void
RTC_A_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\_A 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\_A\_TIME\_EVENT\_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC\_A\_CLOCK\_ALARM\_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC\_A\_CLOCK\_READ\_READY\_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC\_A\_PRESCALE\_TIMER0\_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC\_A\_PRESCALE\_TIMER1\_INTERRUPT** - asserts when Prescaler 1 event condition is met.

**Returns:**

None

### 25.2.2.3 RTC\_A\_counterPrescaleHold

Holds the selected Prescaler.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	10
CCS 4.2.1	Speed	10
IAR 5.51.6	None	16
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

**Prototype:**

```
void
RTC_A_counterPrescaleHold(uint32_t baseAddress,
                        uint8_t prescaleSelect)
```

**Description:**

This function holds the prescale counter from continuing. This will only work in counter mode, in Calendar mode, the [RTC\\_A\\_holdClock\(\)](#) must be used. In counter mode, if using both prescalers in conjunction with the main RTC counter, then stopping RT0PS will stop RT1PS, but stopping RT1PS will not stop RT0PS.

**Parameters:**

**baseAddress** is the base address of the RTC\_A module.

**prescaleSelect** is the prescaler to hold. Valid values are:

- **RTC\_A\_PRESCALE\_0**
- **RTC\_A\_PRESCALE\_1**

**Returns:**

None

### 25.2.2.4 RTC\_A\_counterPrescaleInit

Initializes the Prescaler for Counter mode.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	40
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	14
IAR 5.51.6	None	26
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

#### Prototype:

```
void
RTC_A_counterPrescaleInit(uint32_t baseAddress,
                          uint8_t prescaleSelect,
                          uint16_t prescaleClockSelect,
                          uint16_t prescaleDivider)
```

#### Description:

This function initializes the selected prescaler for the counter mode in the RTC\_A module. If the RTC is initialized in Calendar mode, then these are automatically initialized. The Prescalers can be used to divide a clock source additionally before it gets to the main RTC clock.

#### Parameters:

**baseAddress** is the base address of the RTC\_A module.

**prescaleSelect** is the prescaler to initialize. Valid values are:

- RTC\_A\_PRESCALE\_0
- RTC\_A\_PRESCALE\_1

**prescaleClockSelect** is the clock to drive the selected prescaler. Valid values are:

- RTC\_A\_PSCLOCKSELECT\_ACLK
- RTC\_A\_PSCLOCKSELECT\_SMCLK
- RTC\_A\_PSCLOCKSELECT\_RT0PS - use Prescaler 0 as source to Prescaler 1 (May only be used if prescaleSelect is RTC\_A\_PRESCALE\_1)  
Modified bits are **RTxSSEL** of **RTCPSxCTL** register.

**prescaleDivider** is the divider for the selected clock source. Valid values are:

- RTC\_A\_PSDIVIDER\_2 [Default]
- RTC\_A\_PSDIVIDER\_4
- RTC\_A\_PSDIVIDER\_8
- RTC\_A\_PSDIVIDER\_16
- RTC\_A\_PSDIVIDER\_32
- RTC\_A\_PSDIVIDER\_64
- RTC\_A\_PSDIVIDER\_128
- RTC\_A\_PSDIVIDER\_256  
Modified bits are **RTxPSDIV** of **RTCPSxCTL** register.

#### Returns:

None

### 25.2.2.5 RTC\_A\_counterPrescaleStart

Starts the selected Prescaler.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	10
CCS 4.2.1	Speed	10
IAR 5.51.6	None	16
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	68
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

**Prototype:**

```
void
RTC_A_counterPrescaleStart(uint32_t baseAddress,
                           uint8_t prescaleSelect)
```

**Description:**

This function starts the selected prescale counter. This function will only work if the RTC is in counter mode.

**Parameters:**

**baseAddress** is the base address of the RTC\_A module.

**prescaleSelect** is the prescaler to start. Valid values are:

- RTC\_A\_PRESCALE\_0
- RTC\_A\_PRESCALE\_1

**Returns:**

None

## 25.2.2.6 RTC\_A\_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_A_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\_A module.

**prescaleSelect** is the prescaler to define an interrupt for. Valid values are:

- **RTC\_A\_PRESCALE\_0**
- **RTC\_A\_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\_A\_PSEVENTDIVIDER\_2** [Default]
  - **RTC\_A\_PSEVENTDIVIDER\_4**
  - **RTC\_A\_PSEVENTDIVIDER\_8**
  - **RTC\_A\_PSEVENTDIVIDER\_16**
  - **RTC\_A\_PSEVENTDIVIDER\_32**
  - **RTC\_A\_PSEVENTDIVIDER\_64**
  - **RTC\_A\_PSEVENTDIVIDER\_128**
  - **RTC\_A\_PSEVENTDIVIDER\_256**
- Modified bits are **RTxIP** of **RTCPSxCTL** register.

**Returns:**

None

### 25.2.2.7 RTC\_A\_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_A_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\_A module.

**interruptMask** is a bit mask of the interrupts to disable. Mask value is the logical OR of any of the following:

- **RTC\_A\_TIME\_EVENT\_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC\_A\_CLOCK\_ALARM\_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC\_A\_CLOCK\_READ\_READY\_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC\_A\_PRESCALE\_TIMER0\_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC\_A\_PRESCALE\_TIMER1\_INTERRUPT** - asserts when Prescaler 1 event condition is met.

**Returns:**

None

### 25.2.2.8 RTC\_A\_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_A_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\_A module.

**interruptMask** is a bit mask of the interrupts to enable. Mask value is the logical OR of any of the following:

- **RTC\_A\_TIME\_EVENT\_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC\_A\_CLOCK\_ALARM\_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC\_A\_CLOCK\_READ\_READY\_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC\_A\_PRESCALE\_TIMER0\_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC\_A\_PRESCALE\_TIMER1\_INTERRUPT** - asserts when Prescaler 1 event condition is met.

#### Returns:

None

### 25.2.2.9 RTC\_A\_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_A_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\_A module.

**Returns:**

A Calendar structure containing the current time.

### 25.2.2.10 RTC\_A\_getCounterValue

Returns the value of the Counter register.

**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	64
IAR 5.51.6	Size	46
IAR 5.51.6	Speed	46
MSPGCC 4.8.0	None	134
MSPGCC 4.8.0	Size	62
MSPGCC 4.8.0	Speed	62

**Prototype:**

```
uint32_t
RTC_A_getCounterValue(uint32_t baseAddress)
```

**Description:**

This function returns the value of the counter register for the RTC\_A module. It will return the 32-bit value no matter the size set during initialization. The RTC should be held before trying to use this function.

**Parameters:**

**baseAddress** is the base address of the RTC\_A module.

**Returns:**

The raw value of the full 32-bit Counter Register.

### 25.2.2.11 RTC\_A\_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	44
CCS 4.2.1	Speed	44
IAR 5.51.6	None	64
IAR 5.51.6	Size	52
IAR 5.51.6	Speed	52
MSPGCC 4.8.0	None	160
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	54



**Prototype:**

```
uint8_t
RTC_A_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\_A 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\_A\_TIME\_EVENT\_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC\_A\_CLOCK\_ALARM\_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC\_A\_CLOCK\_READ\_READY\_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC\_A\_PRESCALE\_TIMER0\_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC\_A\_PRESCALE\_TIMER1\_INTERRUPT** - asserts when Prescaler 1 event condition is met.

**Returns:**

Logical OR of any of the following:

- **RTC\_A\_TIME\_EVENT\_INTERRUPT** asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
  - **RTC\_A\_CLOCK\_ALARM\_INTERRUPT** asserts when alarm condition in Calendar mode is met.
  - **RTC\_A\_CLOCK\_READ\_READY\_INTERRUPT** asserts when Calendar registers are settled.
  - **RTC\_A\_PRESCALE\_TIMER0\_INTERRUPT** asserts when Prescaler 0 event condition is met.
  - **RTC\_A\_PRESCALE\_TIMER1\_INTERRUPT** asserts when Prescaler 1 event condition is met.
- indicating the status of the masked interrupts

## 25.2.2.12 RTC\_A\_getPrescaleValue

Returns the selected prescaler value.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	68
CCS 4.2.1	Size	34
CCS 4.2.1	Speed	34
IAR 5.51.6	None	56
IAR 5.51.6	Size	44
IAR 5.51.6	Speed	44
MSPGCC 4.8.0	None	106
MSPGCC 4.8.0	Size	44
MSPGCC 4.8.0	Speed	42

**Prototype:**

```
uint8_t
RTC_A_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\_A module.

**prescaleSelect** is the prescaler to obtain the value of. Valid values are:

- RTC\_A\_PRESCALE\_0
- RTC\_A\_PRESCALE\_1

**Returns:**

The value of the specified prescaler count register

### 25.2.2.13 RTC\_A\_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_A_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\_A module.

**Returns:**

None

### 25.2.2.14 RTC\_A\_initCalendar

Initializes the settings to operate the RTC in calendar mode.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	130
CCS 4.2.1	Size	58
CCS 4.2.1	Speed	58
IAR 5.51.6	None	110
IAR 5.51.6	Size	106
IAR 5.51.6	Speed	106
MSPGCC 4.8.0	None	230
MSPGCC 4.8.0	Size	92
MSPGCC 4.8.0	Speed	92

**Prototype:**

```
void
RTC_A_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\_A 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\_A\_FORMAT\_BINARY** [Default]
- **RTC\_A\_FORMAT\_BCD**  
Modified bits are **RTCBCD** of **RTCCTL1** register.

**Returns:**

None

### 25.2.2.15 RTC\_A\_initCounter

Initializes the settings to operate the RTC in Counter mode.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	62
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	42
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	114
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

**Prototype:**

```
void
RTC_A_initCounter(uint32_t baseAddress,
                  uint16_t clockSelect,
                  uint16_t counterSizeSelect)
```

**Description:**

This function initializes the Counter mode of the RTC\_A. Setting the clock source and counter size will allow an interrupt from the RTCTEVIFG once an overflow to the counter register occurs.

**Parameters:**

**baseAddress** is the base address of the RTC\_A module.

**clockSelect** is the selected clock for the counter mode to use. Valid values are:

- **RTC\_A\_CLOCKSELECT\_ACLK** [Default]
- **RTC\_A\_CLOCKSELECT\_SMCLK**
- **RTC\_A\_CLOCKSELECT\_RT1PS** - use Prescaler 1 as source to RTC  
Modified bits are **RTCSEL** of **RTCCTL1** register.

**counterSizeSelect** is the size of the counter. Valid values are:

- **RTC\_A\_COUNTERSIZE\_8BIT** [Default]
- **RTC\_A\_COUNTERSIZE\_16BIT**
- **RTC\_A\_COUNTERSIZE\_24BIT**
- **RTC\_A\_COUNTERSIZE\_32BIT**  
Modified bits are **RTCTEV** of **RTCCTL1** register.

**Returns:**

None

### 25.2.2.16 RTC\_A\_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_A_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\_A\_ALARM\_OFF for any alarm settings that should not be apart of the alarm condition.

#### Parameters:

**baseAddress** is the base address of the RTC\_A module.

**minutesAlarm** is the alarm condition for the minutes. Valid values are:

- RTC\_A\_ALARMCONDITION\_OFF [Default]
- An integer between 0-59

**hoursAlarm** is the alarm condition for the hours. Valid values are:

- RTC\_A\_ALARMCONDITION\_OFF [Default]
- An integer between 0-24

**dayOfWeekAlarm** is the alarm condition for the day of week. Valid values are:

- RTC\_A\_ALARMCONDITION\_OFF [Default]
- An integer between 0-6

**dayOfMonthAlarm** is the alarm condition for the day of the month. Valid values are:

- RTC\_A\_ALARMCONDITION\_OFF [Default]
- An integer between 0-31

#### Returns:

None

### 25.2.2.17 RTC\_A\_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_A_setCalendarEvent(uint32_t baseAddress,
                      uint16_t eventSelect)
```

**Description:**

This function sets a specified event to assert the RTCTEVIHG interrupt. This interrupt is independent from the Calendar alarm interrupt.

**Parameters:**

**baseAddress** is the base address of the RTC\_A module.

**eventSelect** is the condition selected. Valid values are:

- **RTC\_A\_CALENDAREVENT\_MINUTECHANGE** - assert interrupt on every minute
  - **RTC\_A\_CALENDAREVENT\_HOURLCHANGE** - assert interrupt on every hour
  - **RTC\_A\_CALENDAREVENT\_NOON** - assert interrupt when hour is 12
  - **RTC\_A\_CALENDAREVENT\_MIDNIGHT** - assert interrupt when hour is 0
- Modified bits are **RTCTEV** of **RTCCTL** register.

**Returns:**

None

### 25.2.2.18 RTC\_A\_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_A_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\_A module.

**offsetDirection** is the direction that the calibration offset will go. Valid values are:

- **RTC\_A\_CALIBRATION\_DOWN2PPM** - calibrate at steps of -2
  - **RTC\_A\_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

### 25.2.2.19 RTC\_A\_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_A_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\_A module.

**frequencySelect** is the frequency output to RTCCLK. Valid values are:

- **RTC\_A\_CALIBRATIONFREQ\_OFF** [Default] - turn off calibration output
  - **RTC\_A\_CALIBRATIONFREQ\_512HZ** - output signal at 512Hz for calibration
  - **RTC\_A\_CALIBRATIONFREQ\_256HZ** - output signal at 256Hz for calibration
  - **RTC\_A\_CALIBRATIONFREQ\_1HZ** - output signal at 1Hz for calibration
- Modified bits are **RTCCALF** of **RTCCTL3** register.

**Returns:**

None

### 25.2.2.20 RTC\_A\_setCounterValue

Sets the value of the Counter register.

**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	26
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	38
MSPGCC 4.8.0	Speed	38

**Prototype:**

```
void
RTC_A_setCounterValue(uint32_t baseAddress,
                     uint32_t counterValue)
```

**Description:**

This function sets the counter register of the RTC\_A module.

**Parameters:**

**baseAddress** is the base address of the RTC\_A module.

**counterValue** is the value to set the Counter register to; a valid value may be any 32-bit integer.

**Returns:**

None

### 25.2.2.21 RTC\_A\_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_A_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\_A module.

**prescaleSelect** is the prescaler to set the value for. Valid values are:

- RTC\_A\_PRESCALE\_0
- RTC\_A\_PRESCALE\_1

**prescaleCounterValue** is the specified value to set the prescaler to. Valid values are any integer between 0-255  
Modified bits are **RTxPS** of **RTxPS** register.

**Returns:**  
None

### 25.2.2.22 RTC\_A\_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_A_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\_A module.

**Returns:**

None

## 25.3 Programming Example

The following example shows how to initialize and use the RTC API to setup Calendar Mode with the current time and various interrupts.

```
//Initialize Calendar Mode of RTC
/*
Base Address of the RTC_A
Pass in current time, initialized above
Use BCD as Calendar Register Format
*/
RTC_A_calendarInit(RTC_A_BASE,
    currentTime,
    RTC_A_FORMAT_BCD);

//Setup Calendar Alarm for 5:00pm on the 5th day of the week.
//Note: Does not specify day of the week.
RTC_A_setCalendarAlarm(RTC_A_BASE,
    0x00,
    0x17,
    RTC_A_ALARMCONDITION_OFF,
    0x05);
```



```
//Specify an interrupt to assert every minute
RTC_A_setCalendarEvent(RTC_A_BASE,
    RTC_A_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_A_enableInterrupt(RTC_A_BASE,
    RTCRDYIE + RTCTEVIE + RTCAIE);

//Start RTC Clock
RTC_A_startClock(RTC_A_BASE);

//Enter LPM3 mode with interrupts enabled
__bis_SR_register(LPM3_bits + GIE);
__no_operation();
```

## 26 Real-Time Clock (RTC\_B)

Introduction .....	278
API Functions .....	278
Programming Example .....	287

### 26.1 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.

### 26.2 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)

#### 26.2.1 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 Calendar 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\(\)](#)

## 26.2.2 Function Documentation

### 26.2.2.1 RTC\_B\_calendarInit

Deprecated - Initializes the settings to operate the RTC in calendar mode.

**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

### 26.2.2.2 RTC\_B\_clearInterrupt

Clears selected RTC interrupt flags.

**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

### 26.2.2.3 RTC\_B\_convertBCDToBinary

Returns the given BCD value in Binary Format.

**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.

### 26.2.2.4 RTC\_B\_convertBinaryToBCD

Returns the given Binary value in BCD Format.

**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.

### 26.2.2.5 RTC\_B\_definePrescaleEvent

Sets up an interrupt condition for the selected Prescaler.

**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

### 26.2.2.6 RTC\_B\_disableInterrupt

Disables selected RTC interrupt sources.

**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

### 26.2.2.7 RTC\_B\_enableInterrupt

Enables selected RTC interrupt sources.

**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

### 26.2.2.8 RTC\_B\_getCalendarTime

Returns the Calendar Time stored in the Calendar registers of the RTC.

**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.

### 26.2.2.9 RTC\_B\_getInterruptStatus

Returns the status of the selected interrupts flags.

**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

### 26.2.2.10 RTC\_B\_getPrescaleValue

Returns the selected prescaler value.

**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

### 26.2.2.11 RTC\_B\_holdClock

Holds the RTC.

**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

### 26.2.2.12 RTC\_B\_initCalendar

Initializes the settings to operate the RTC in calendar mode.

**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 **RTCB**CD of **RTCCTL1** register.

**Returns:**

None

### 26.2.2.13 RTC\_B\_setCalendarAlarm

Sets and Enables the desired Calendar Alarm settings.

**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

### 26.2.2.14 RTC\_B\_setCalendarEvent

Sets a single specified Calendar interrupt condition.

**Prototype:**

```
void
RTC_B_setCalendarEvent(uint32_t baseAddress,
                      uint16_t eventSelect)
```

**Description:**

This function sets a specified event to assert the RTCTEVIIFG 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\_HOURLCHANGE** - assert interrupt on every hour
  - **RTC\_B\_CALENDAREVENT\_NOON** - assert interrupt when hour is 12
  - **RTC\_B\_CALENDAREVENT\_MIDNIGHT** - assert interrupt when hour is 0
- Modified bits are **RTCTEV** of **RTCCTL** register.

**Returns:**

None

### 26.2.2.15 RTC\_B\_setCalibrationData

Sets the specified calibration for the RTC.

**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

## 26.2.2.16 RTC\_B\_setCalibrationFrequency

Allows and Sets the frequency output to RTCCLK pin for calibration measurement.

**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

## 26.2.2.17 RTC\_B\_setPrescaleCounterValue

Sets the selected prescaler value.

**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

### 26.2.2.18 RTC\_B\_startClock

Starts the RTC.

**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

## 26.3 Programming Example

The following example shows how to initialize and use the RTC API to setup Calendar Mode with the current time and various interrupts.

```
//Initialize Calendar 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();
```

## 27 Real-Time Clock (RTC\_C)

Introduction .....	288
API Functions .....	288
Programming Example .....	306

### 27.1 Introduction

The Real Time Clock (RTC\_C) API provides a set of functions for using the MSP430Ware RTC\_C modules. Functions are provided to calibrate the clock, initialize the RTC\_C modules in Calendar mode, and setup conditions for, and enable, interrupts for the RTC\_C modules.

The RTC\_C module provides the ability to keep track of the current time and date in calendar mode.

The RTC\_C module generates multiple interrupts. There are 2 interrupts that can be defined in calendar mode, and 1 interrupt in counter mode for counter overflow, as well as an interrupt for each prescaler.

If the device header file defines the baseaddress as RTC\_C\_BASE, pass in RTC\_C\_BASE as the baseaddress parameter. If the device header file defines the baseaddress as RTC\_CE\_BASE, pass in RTC\_CE\_BASE as the baseaddress parameter.

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

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	1786
CCS 4.2.1	Size	914
CCS 4.2.1	Speed	908
IAR 5.51.6	None	1366
IAR 5.51.6	Size	1008
IAR 5.51.6	Speed	1076
MSPGCC 4.8.0	None	3006
MSPGCC 4.8.0	Size	1144
MSPGCC 4.8.0	Speed	1240

### 27.2 API Functions

#### Functions

- void [RTC\\_C\\_calendarInit](#) (uint32\_t baseAddress, Calendar CalendarTime, uint16\_t formatSelect)
- void [RTC\\_C\\_clearInterrupt](#) (uint32\_t baseAddress, uint8\_t interruptFlagMask)
- uint16\_t [RTC\\_C\\_convertBCDToBinary](#) (uint32\_t baseAddress, uint16\_t valueToConvert)
- uint16\_t [RTC\\_C\\_convertBinaryToBCD](#) (uint32\_t baseAddress, uint16\_t valueToConvert)
- void [RTC\\_C\\_counterPrescaleHold](#) (uint32\_t baseAddress, uint8\_t prescaleSelect)
- void [RTC\\_C\\_counterPrescaleInit](#) (uint32\_t baseAddress, uint8\_t prescaleSelect, uint16\_t prescaleClockSelect, uint16\_t prescaleDivider)
- void [RTC\\_C\\_counterPrescaleStart](#) (uint32\_t baseAddress, uint8\_t prescaleSelect)
- void [RTC\\_C\\_definePrescaleEvent](#) (uint32\_t baseAddress, uint8\_t prescaleSelect, uint8\_t prescaleEventDivider)
- void [RTC\\_C\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t interruptMask)
- void [RTC\\_C\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t interruptMask)

- Calendar [RTC\\_C\\_getCalendarTime](#) (uint32\_t baseAddress)
- uint32\_t [RTC\\_C\\_getCounterValue](#) (uint32\_t baseAddress)
- uint8\_t [RTC\\_C\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t interruptFlagMask)
- uint8\_t [RTC\\_C\\_getPrescaleValue](#) (uint32\_t baseAddress, uint8\_t prescaleSelect)
- void [RTC\\_C\\_holdClock](#) (uint32\_t baseAddress)
- void [RTC\\_C\\_initCalendar](#) (uint32\_t baseAddress, Calendar \*CalendarTime, uint16\_t formatSelect)
- void [RTC\\_C\\_initCounter](#) (uint32\_t baseAddress, uint16\_t clockSelect, uint16\_t counterSizeSelect)
- void [RTC\\_C\\_setCalendarAlarm](#) (uint32\_t baseAddress, uint8\_t minutesAlarm, uint8\_t hoursAlarm, uint8\_t dayOfWeekAlarm, uint8\_t dayOfMonthAlarm)
- void [RTC\\_C\\_setCalendarEvent](#) (uint32\_t baseAddress, uint16\_t eventSelect)
- void [RTC\\_C\\_setCalibrationData](#) (uint32\_t baseAddress, uint8\_t offsetDirection, uint8\_t offsetValue)
- void [RTC\\_C\\_setCalibrationFrequency](#) (uint32\_t baseAddress, uint16\_t frequencySelect)
- void [RTC\\_C\\_setCounterValue](#) (uint32\_t baseAddress, uint32\_t counterValue)
- void [RTC\\_C\\_setPrescaleValue](#) (uint32\_t baseAddress, uint8\_t prescaleSelect, uint8\_t prescaleCounterValue)
- bool [RTC\\_C\\_setTemperatureCompensation](#) (uint32\_t baseAddress, uint8\_t offsetDirection, uint8\_t offsetValue)
- void [RTC\\_C\\_startClock](#) (uint32\_t baseAddress)

## 27.2.1 Detailed Description

The RTC\_C API is broken into 5 groups of functions: clock settings, calender mode, counter mode, interrupt condition setup and enable functions and data conversion.

The RTC\_C clock settings are handled by

- [RTC\\_C\\_startClock\(\)](#)
- [RTC\\_C\\_holdClock\(\)](#)
- [RTC\\_C\\_setCalibrationFrequency\(\)](#)
- [RTC\\_C\\_setCalibrationData\(\)](#)
- [RTC\\_C\\_setTemperatureCompensation\(\)](#)

The RTC\_C Calender Mode is initialized and setup by

- [RTC\\_C\\_calenderInit\(\)](#)
- [RTC\\_C\\_getCalendarTime\(\)](#)
- [RTC\\_C\\_getPrescaleValue\(\)](#)
- [RTC\\_C\\_setPrescaleValue\(\)](#)

The RTC\_C Counter Mode is initialized and handled by

- [RTC\\_C\\_counterInit\(\)](#)
- [RTC\\_C\\_counterPrescaleInit\(\)](#)
- [RTC\\_C\\_setCounterValue\(\)](#)
- [RTC\\_C\\_getCounterValue\(\)](#)
- [RTC\\_C\\_counterPrescaleHold\(\)](#)
- [RTC\\_C\\_counterPrescaleStart\(\)](#)

The RTC\_C interrupts are handled by

- [RTC\\_C\\_setCalenderAlarm\(\)](#)
- [RTC\\_C\\_setCalenderEvent\(\)](#)
- [RTC\\_C\\_definePrescaleEvent\(\)](#)
- [RTC\\_C\\_enableInterrupt\(\)](#)
- [RTC\\_C\\_disableInterrupt\(\)](#)
- [RTC\\_C\\_getInterruptStatus\(\)](#)

- [RTC\\_C\\_clearInterrupt\(\)](#)

The RTC\_C data conversion is handled by

- [RTC\\_C\\_convertBCDToBinary\(\)](#)
- [RTC\\_C\\_convertBinaryToBCD\(\)](#)

## 27.2.2 Function Documentation

### 27.2.2.1 RTC\_C\_calendarInit

Deprecated - Initializes the settings to operate the RTC in calendar mode.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	148
CCS 4.2.1	Size	102
CCS 4.2.1	Speed	102
IAR 5.51.6	None	132
IAR 5.51.6	Size	116
IAR 5.51.6	Speed	122
MSPGCC 4.8.0	None	256
MSPGCC 4.8.0	Size	166
MSPGCC 4.8.0	Speed	166

**Prototype:**

```
void
RTC_C_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\_C module.

**CalendarTime** is the structure containing the values for the Calendar to be initialized to. Valid values should be of type **Calendar** and should contain the following members and corresponding values: **Seconds** between 0-59 **Minutes** between 0-59 **Hours** between 0-24 **DayOfWeek** between 0-6 **DayOfMonth** between 0-31 **Year** between 0-4095 NOTE: Values beyond the ones specified may result in erratic behavior.

**formatSelect** is the format for the Calendar registers to use. Valid values are:

- **RTC\_C\_FORMAT\_BINARY** [Default]
- **RTC\_C\_FORMAT\_BCD**  
Modified bits are **RTCB**CD of **RTCCTL1** register.

**Returns:**

None

### 27.2.2.2 RTC\_C\_clearInterrupt

Clears selected RTC interrupt flags.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	92
CCS 4.2.1	Size	48
CCS 4.2.1	Speed	48
IAR 5.51.6	None	62
IAR 5.51.6	Size	54
IAR 5.51.6	Speed	60
MSPGCC 4.8.0	None	188
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	50

**Prototype:**

```
void
RTC_C_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\_C module.

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

- **RTC\_C\_TIME\_EVENT\_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC\_C\_CLOCK\_ALARM\_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC\_C\_CLOCK\_READ\_READY\_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC\_C\_PRESCALE\_TIMER0\_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC\_C\_PRESCALE\_TIMER1\_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC\_C\_OSCILLATOR\_FAULT\_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

**Returns:**

None

### 27.2.2.3 RTC\_C\_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_C_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\_C module.

**valueToConvert** is the raw value in BCD format to convert to Binary.

Modified bits are **BCD2BIN** of **BCD2BIN** register.

**Returns:**

The Binary version of the valueToConvert parameter.

### 27.2.2.4 RTC\_C\_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_C_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\_C module.

**valueToConvert** is the raw value in Binary format to convert to BCD.

Modified bits are **BIN2BCD** of **BIN2BCD** register.

**Returns:**

The BCD version of the valueToConvert parameter.

### 27.2.2.5 RTC\_C\_counterPrescaleHold

Holds the selected Prescaler.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	10
CCS 4.2.1	Speed	10
IAR 5.51.6	None	16
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16



**Prototype:**

```
void
RTC_C_counterPrescaleHold(uint32_t baseAddress,
                          uint8_t prescaleSelect)
```

**Description:**

This function holds the prescale counter from continuing. This will only work in counter mode, in Calendar mode, the [RTC\\_C\\_holdClock\(\)](#) must be used. In counter mode, if using both prescalers in conjunction with the main RTC counter, then stopping RT0PS will stop RT1PS, but stopping RT1PS will not stop RT0PS.

**Parameters:**

**baseAddress** is the base address of the RTC\_C module.

**prescaleSelect** is the prescaler to hold. Valid values are:

- RTC\_C\_PRESCALE\_0
- RTC\_C\_PRESCALE\_1

**Returns:**

None

### 27.2.2.6 RTC\_C\_counterPrescaleInit

Initializes the Prescaler for Counter mode.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	40
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	14
IAR 5.51.6	None	26
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	44
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

**Prototype:**

```
void
RTC_C_counterPrescaleInit(uint32_t baseAddress,
                          uint8_t prescaleSelect,
                          uint16_t prescaleClockSelect,
                          uint16_t prescaleDivider)
```

**Description:**

This function initializes the selected prescaler for the counter mode in the RTC\_C module. If the RTC is initialized in Calendar mode, then these are automatically initialized. The Prescalers can be used to divide a clock source additionally before it gets to the main RTC clock.

**Parameters:**

**baseAddress** is the base address of the RTC\_C module.

**prescaleSelect** is the prescaler to initialize. Valid values are:

- RTC\_C\_PRESCALE\_0
- RTC\_C\_PRESCALE\_1

**prescaleClockSelect** is the clock to drive the selected prescaler. Valid values are:

- RTC\_C\_PSCLOCKSELECT\_ACLK
- RTC\_C\_PSCLOCKSELECT\_SMCLK
- RTC\_C\_PSCLOCKSELECT\_RT0PS - use Prescaler 0 as source to Prescaler 1 (May only be used if prescaleSelect is RTC\_C\_PRESCALE\_1)  
Modified bits are RTxSSEL of RTCPSxCTL register.

**prescaleDivider** is the divider for the selected clock source. Valid values are:

- RTC\_C\_PSDIVIDER\_2 [Default]
- RTC\_C\_PSDIVIDER\_4
- RTC\_C\_PSDIVIDER\_8
- RTC\_C\_PSDIVIDER\_16
- RTC\_C\_PSDIVIDER\_32
- RTC\_C\_PSDIVIDER\_64
- RTC\_C\_PSDIVIDER\_128
- RTC\_C\_PSDIVIDER\_256

Modified bits are **RTxPSDIV** of **RTCPSxCTL** register.

**Returns:**

None

### 27.2.2.7 RTC\_C\_counterPrescaleStart

Starts the selected Prescaler.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	36
CCS 4.2.1	Size	10
CCS 4.2.1	Speed	10
IAR 5.51.6	None	16
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	68
MSPGCC 4.8.0	Size	16
MSPGCC 4.8.0	Speed	16

**Prototype:**

```
void
RTC_C_counterPrescaleStart(uint32_t baseAddress,
                           uint8_t prescaleSelect)
```

**Description:**

This function starts the selected prescale counter. This function will only work if the RTC is in counter mode.

**Parameters:**

**baseAddress** is the base address of the RTC\_C module.

**prescaleSelect** is the prescaler to start. Valid values are:

- RTC\_C\_PRESCALE\_0
- RTC\_C\_PRESCALE\_1

**Returns:**

None

### 27.2.2.8 RTC\_C\_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_C_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\_C module.

**prescaleSelect** is the prescaler to define an interrupt for. Valid values are:

- RTC\_C\_PRESCALE\_0
- RTC\_C\_PRESCALE\_1

**prescaleEventDivider** is a divider to specify when an interrupt can occur based on the clock source of the selected prescaler. (Does not affect timer of the selected prescaler). Valid values are:

- RTC\_C\_PSEVENTDIVIDER\_2 [Default]
- RTC\_C\_PSEVENTDIVIDER\_4
- RTC\_C\_PSEVENTDIVIDER\_8
- RTC\_C\_PSEVENTDIVIDER\_16
- RTC\_C\_PSEVENTDIVIDER\_32
- RTC\_C\_PSEVENTDIVIDER\_64
- RTC\_C\_PSEVENTDIVIDER\_128
- RTC\_C\_PSEVENTDIVIDER\_256

Modified bits are RTxIP of RTCPSxCTL register.

**Returns:**

None

### 27.2.2.9 RTC\_C\_disableInterrupt

Disables selected RTC interrupt sources.

**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	62
IAR 5.51.6	Size	50
IAR 5.51.6	Speed	56
MSPGCC 4.8.0	None	192
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	50

**Prototype:**

```
void
RTC_C_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\_C module.

**interruptMask** is a bit mask of the interrupts to disable. Mask value is the logical OR of any of the following:

- **RTC\_C\_TIME\_EVENT\_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC\_C\_CLOCK\_ALARM\_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC\_C\_CLOCK\_READ\_READY\_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC\_C\_PRESCALE\_TIMER0\_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC\_C\_PRESCALE\_TIMER1\_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC\_C\_OSCILLATOR\_FAULT\_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

**Returns:**

None

## 27.2.2.10 RTC\_C\_enableInterrupt

Enables selected RTC interrupt sources.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	88
CCS 4.2.1	Size	46
CCS 4.2.1	Speed	46
IAR 5.51.6	None	62
IAR 5.51.6	Size	50
IAR 5.51.6	Speed	56
MSPGCC 4.8.0	None	180
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	50

**Prototype:**

```
void
RTC_C_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\_C module.

**interruptMask** is a bit mask of the interrupts to enable. Mask value is the logical OR of any of the following:

- **RTC\_C\_TIME\_EVENT\_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC\_C\_CLOCK\_ALARM\_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC\_C\_CLOCK\_READ\_READY\_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC\_C\_PRESCALE\_TIMER0\_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC\_C\_PRESCALE\_TIMER1\_INTERRUPT** - asserts when Prescaler 1 event condition is met.

- **RTC\_C\_OSCILLATOR\_FAULT\_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

**Returns:**  
None

### 27.2.2.11 RTC\_C\_getCalendarTime

Returns the Calendar Time stored in the Calendar registers of the RTC.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	84
CCS 4.2.1	Size	68
CCS 4.2.1	Speed	68
IAR 5.51.6	None	108
IAR 5.51.6	Size	108
IAR 5.51.6	Speed	108
MSPGCC 4.8.0	None	182
MSPGCC 4.8.0	Size	62
MSPGCC 4.8.0	Speed	62

**Prototype:**

```
Calendar
RTC_C_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\_C module.

**Returns:**

A Calendar structure containing the current time.

### 27.2.2.12 RTC\_C\_getCounterValue

Returns the value of the Counter register.

**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	64
IAR 5.51.6	Size	46
IAR 5.51.6	Speed	46
MSPGCC 4.8.0	None	134
MSPGCC 4.8.0	Size	62
MSPGCC 4.8.0	Speed	62

**Prototype:**

```
uint32_t
RTC_C_getCounterValue(uint32_t baseAddress)
```

**Description:**

This function returns the value of the counter register for the RTC\_C module. It will return the 32-bit value no matter the size set during initialization. The RTC should be held before trying to use this function.

**Parameters:**

**baseAddress** is the base address of the RTC\_C module.

**Returns:**

The raw value of the full 32-bit Counter Register.

### 27.2.2.13 RTC\_C\_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_C_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\_C module.

**interruptFlagMask** is a bit mask of the interrupt flags to return the status of. Mask value is the logical OR of any of the following:

- **RTC\_C\_TIME\_EVENT\_INTERRUPT** - asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
- **RTC\_C\_CLOCK\_ALARM\_INTERRUPT** - asserts when alarm condition in Calendar mode is met.
- **RTC\_C\_CLOCK\_READ\_READY\_INTERRUPT** - asserts when Calendar registers are settled.
- **RTC\_C\_PRESCALE\_TIMER0\_INTERRUPT** - asserts when Prescaler 0 event condition is met.
- **RTC\_C\_PRESCALE\_TIMER1\_INTERRUPT** - asserts when Prescaler 1 event condition is met.
- **RTC\_C\_OSCILLATOR\_FAULT\_INTERRUPT** - asserts if there is a problem with the 32kHz oscillator, while the RTC is running.

**Returns:**

Logical OR of any of the following:

- **RTC\_C\_TIME\_EVENT\_INTERRUPT** asserts when counter overflows in counter mode or when Calendar event condition defined by `defineCalendarEvent()` is met.
  - **RTC\_C\_CLOCK\_ALARM\_INTERRUPT** asserts when alarm condition in Calendar mode is met.
  - **RTC\_C\_CLOCK\_READ\_READY\_INTERRUPT** asserts when Calendar registers are settled.
  - **RTC\_C\_PRESCALE\_TIMER0\_INTERRUPT** asserts when Prescaler 0 event condition is met.
  - **RTC\_C\_PRESCALE\_TIMER1\_INTERRUPT** asserts when Prescaler 1 event condition is met.
  - **RTC\_C\_OSCILLATOR\_FAULT\_INTERRUPT** asserts if there is a problem with the 32kHz oscillator, while the RTC is running.
- indicating the status of the masked interrupts

### 27.2.2.14 RTC\_C\_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_C_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\_C module.

**prescaleSelect** is the prescaler to obtain the value of. Valid values are:

- **RTC\_C\_PRESCALE\_0**
- **RTC\_C\_PRESCALE\_1**

#### Returns:

The value of the specified prescaler count register

### 27.2.2.15 RTC\_C\_holdClock

Holds the RTC.

#### 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	30
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

#### Prototype:

```
void
RTC_C_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\_C module.

**Returns:**

None

### 27.2.2.16 RTC\_C\_initCalendar

Initializes the settings to operate the RTC in calendar mode.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	148
CCS 4.2.1	Size	76
CCS 4.2.1	Speed	76
IAR 5.51.6	None	134
IAR 5.51.6	Size	120
IAR 5.51.6	Speed	120
MSPGCC 4.8.0	None	254
MSPGCC 4.8.0	Size	108
MSPGCC 4.8.0	Speed	108

**Prototype:**

```
void
RTC_C_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\_C module.

**CalendarTime** is the pointer to the structure containing the values for the Calendar to be initialized to. Valid values should be of type pointer to Calendar and should contain the following members and corresponding values:

**Seconds** between 0-59 **Minutes** between 0-59 **Hours** between 0-24 **DayOfWeek** between 0-6 **DayOfMonth** between 0-31 **Year** between 0-4095 NOTE: Values beyond the ones specified may result in erratic behavior.

**formatSelect** is the format for the Calendar registers to use. Valid values are:

- **RTC\_C\_FORMAT\_BINARY** [Default]
- **RTC\_C\_FORMAT\_BCD**  
Modified bits are **RTCB CD** of **RTCCTL1** register.

**Returns:**

None

### 27.2.2.17 RTC\_C\_initCounter

Initializes the settings to operate the RTC in Counter mode.

**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	48
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	28
MSPGCC 4.8.0	None	130
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

**Prototype:**

```
void
RTC_C_initCounter(uint32_t baseAddress,
                  uint16_t clockSelect,
                  uint16_t counterSizeSelect)
```

**Description:**

This function initializes the Counter mode of the RTC\_C. Setting the clock source and counter size will allow an interrupt from the RTCTEVIFG once an overflow to the counter register occurs.

**Parameters:**

**baseAddress** is the base address of the RTC\_C module.

**clockSelect** is the selected clock for the counter mode to use. Valid values are:

- **RTC\_C\_CLOCKSELECT\_32KHZ\_OSC**
- **RTC\_C\_CLOCKSELECT\_RT1PS**  
Modified bits are **RTCSEL** of **RTCCTL1** register.

**counterSizeSelect** is the size of the counter. Valid values are:

- **RTC\_C\_COUNTERSIZE\_8BIT** [Default]
- **RTC\_C\_COUNTERSIZE\_16BIT**
- **RTC\_C\_COUNTERSIZE\_24BIT**
- **RTC\_C\_COUNTERSIZE\_32BIT**  
Modified bits are **RTCCTEV** of **RTCCTL1** register.

**Returns:**

None

### 27.2.2.18 RTC\_C\_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_C_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\_C\_ALARM\_OFF for any alarm settings that should not be apart of the alarm condition.

**Parameters:**

**baseAddress** is the base address of the RTC\_C module.

**minutesAlarm** is the alarm condition for the minutes. Valid values are:

- RTC\_C\_ALARMCONDITION\_OFF [Default]
- An integer between 0-59

**hoursAlarm** is the alarm condition for the hours. Valid values are:

- RTC\_C\_ALARMCONDITION\_OFF [Default]
- An integer between 0-24

**dayOfWeekAlarm** is the alarm condition for the day of week. Valid values are:

- RTC\_C\_ALARMCONDITION\_OFF [Default]
- An integer between 0-6

**dayOfMonthAlarm** is the alarm condition for the day of the month. Valid values are:

- RTC\_C\_ALARMCONDITION\_OFF [Default]
- An integer between 0-31

**Returns:**

None

## 27.2.2.19 RTC\_C\_setCalendarEvent

Sets a single specified Calendar interrupt condition.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	56
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	38
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	110
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

**Prototype:**

```
void
RTC_C_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\_C module.

**eventSelect** is the condition selected. Valid values are:

- **RTC\_C\_CALENDAREVENT\_MINUTECHANGE** - assert interrupt on every minute
  - **RTC\_C\_CALENDAREVENT\_HOURLCHANGE** - assert interrupt on every hour
  - **RTC\_C\_CALENDAREVENT\_NOON** - assert interrupt when hour is 12
  - **RTC\_C\_CALENDAREVENT\_MIDNIGHT** - assert interrupt when hour is 0
- Modified bits are **RTCCTEV** of **RTCCTL** register.

**Returns:**  
None

### 27.2.2.20 RTC\_C\_setCalibrationData

Sets the specified calibration for the RTC.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	56
CCS 4.2.1	Size	26
CCS 4.2.1	Speed	26
IAR 5.51.6	None	38
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

**Prototype:**

```
void
RTC_C_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).

**Parameters:**

**baseAddress** is the base address of the RTC\_C module.

**offsetDirection** is the direction that the calibration offset will go. Valid values are:

- **RTC\_C\_CALIBRATION\_DOWN1PPM** - calibrate at steps of -1
- **RTC\_C\_CALIBRATION\_UP1PPM** - calibrate at steps of +1

Modified bits are **RTC0CAL5** of **RTC0CAL** register.

**offsetValue** is the value that the offset will be a factor of; a valid value is any integer from 1-240.

Modified bits are **RTC0CALx** of **RTC0CAL** register.

**Returns:**  
None

### 27.2.2.21 RTC\_C\_setCalibrationFrequency

Allows and Sets the frequency output to RTCCLK pin for calibration measurement.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	56
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	38
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	86
MSPGCC 4.8.0	Size	28
MSPGCC 4.8.0	Speed	28

**Prototype:**

```
void
RTC_C_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\_C module.

**frequencySelect** is the frequency output to RTCCLK. Valid values are:

- **RTC\_C\_CALIBRATIONFREQ\_OFF** [Default] - turn off calibration output
  - **RTC\_C\_CALIBRATIONFREQ\_512HZ** - output signal at 512Hz for calibration
  - **RTC\_C\_CALIBRATIONFREQ\_256HZ** - output signal at 256Hz for calibration
  - **RTC\_C\_CALIBRATIONFREQ\_1HZ** - output signal at 1Hz for calibration
- Modified bits are **RTCCALF** of **RTCCTL3** register.

**Returns:**

None

## 27.2.2.22 RTC\_C\_setCounterValue

Sets the value of the Counter register.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	146
CCS 4.2.1	Size	82
CCS 4.2.1	Speed	84
IAR 5.51.6	None	106
IAR 5.51.6	Size	80
IAR 5.51.6	Speed	80
MSPGCC 4.8.0	None	200
MSPGCC 4.8.0	Size	112
MSPGCC 4.8.0	Speed	194

**Prototype:**

```
void
RTC_C_setCounterValue(uint32_t baseAddress,
                     uint32_t counterValue)
```

**Description:**

This function sets the counter register of the RTC\_C module.

**Parameters:**

**baseAddress** is the base address of the RTC\_C module.

**counterValue** is the value to set the Counter register to; a valid value may be any 32-bit integer.

**Returns:**  
None

### 27.2.2.23 RTC\_C\_setPrescaleValue

Sets the selected Prescaler value.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	70
CCS 4.2.1	Size	30
CCS 4.2.1	Speed	30
IAR 5.51.6	None	54
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	88
MSPGCC 4.8.0	Size	48
MSPGCC 4.8.0	Speed	60

**Prototype:**

```
void
RTC_C_setPrescaleValue(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\_C module.

**prescaleSelect** is the prescaler to set the value for. Valid values are:

- RTC\_C\_PRESCALE\_0
- RTC\_C\_PRESCALE\_1

**prescaleCounterValue** is the specified value to set the prescaler to. Valid values are any integer between 0-255  
Modified bits are **RTxPS** of **RTxPS** register.

**Returns:**  
None

### 27.2.2.24 RTC\_C\_setTemperatureCompensation

Sets the specified temperature compensation for the RTC.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	66
CCS 4.2.1	Size	34
CCS 4.2.1	Speed	34
IAR 5.51.6	None	60
IAR 5.51.6	Size	44
IAR 5.51.6	Speed	44
MSPGCC 4.8.0	None	102
MSPGCC 4.8.0	Size	42
MSPGCC 4.8.0	Speed	42

**Prototype:**

```
bool
RTC_C_setTemperatureCompensation(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 +1-ppm or -1-ppm, and the offsetValue should be from 1-240 and is multiplied by the direction setting (i.e. +1-ppm \* 8 (offsetValue) = +8-ppm).

**Parameters:**

**baseAddress** is the base address of the RTC\_C module.

**offsetDirection** is the direction that the calibration offset will go Valid values are:

- **RTC\_C\_COMPENSATION\_DOWN1PPM**
- **RTC\_C\_COMPENSATION\_UP1PPM**

Modified bits are **RTCTCMPS** of **RTCTCMP** register.

**offsetValue** is the value that the offset will be a factor of; a valid value is any integer from 1-240.

Modified bits are **RTCTCMPx** of **RTCTCMP** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of setting the temperature compensation

### 27.2.2.25 RTC\_C\_startClock

Starts the RTC.

**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	30
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	58
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	22

**Prototype:**

```
void
RTC_C_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\_C module.

**Returns:**

None

## 27.3 Programming Example

The following example shows how to initialize and use the RTC\_C API to setup Calender Mode with the current time and various interrupts.

```
//Initialize Calendar Mode of RTC_C
/*
Base Address of the RTC_C_A
Pass in current time, initialized above
Use BCD as Calendar Register Format
*/
RTC_C_calendarInit(RTC_C_BASE,
    currentTime,
    RTC_C_FORMAT_BCD);

//Setup Calendar Alarm for 5:00pm on the 5th day of the week.
//Note: Does not specify day of the week.
RTC_C_setCalendarAlarm(RTC_C_BASE,
    0x00,
    0x17,
    RTC_C_ALARMCONDITION_OFF,
    0x05);

//Specify an interrupt to assert every minute
RTC_C_setCalendarEvent(RTC_C_BASE,
    RTC_C_CALENDAREVENT_MINUTECHANGE);

//Enable interrupt for RTC_C Ready Status, which asserts when the RTC_C
//Calendar registers are ready to read.
//Also, enable interrupts for the Calendar alarm and Calendar event.
RTC_C_enableInterrupt(RTC_C_BASE,
    RTC_CRDYIE + RTC_CTEVIE + RTC_CAIE);

//Start RTC_C Clock
RTC_C_startClock(RTC_C_BASE);

//Enter LPM3 mode with interrupts enabled
__bis_SR_register(LPM3_bits + GIE);
__no_operation();
```

## 28 24-Bit Sigma Delta Converter (SD24\_B)

Introduction .....	308
API Functions .....	308
Programming Example .....	326

### 28.1 Introduction

The SD24\_B module consists of up to eight independent sigma-delta analog-to-digital converters. The converters are based on second-order oversampling sigma-delta modulators and digital decimation filters. The decimation filters are comb type filters with selectable oversampling ratios of up to 1024. Additional filtering can be done in software.

A sigma-delta analog-to-digital converter basically consists of two parts: the analog part

- called modulator - and the digital part - a decimation filter. The modulator of the SD24\_B provides a bit stream of zeros and ones to the digital decimation filter. The digital filter averages the bitstream from the modulator over a given number of bits (specified by the oversampling rate) and provides samples at a reduced rate for further processing to the CPU.

As commonly known averaging can be used to increase the signal-to-noise performance of a conversion. With a conventional ADC each factor-of-4 oversampling can improve the SNR by about 6 dB or 1 bit. To achieve a 16-bit resolution out of a simple 1-bit ADC would require an impractical oversampling rate of  $4^{15} = 1.073.741.824$ . To overcome this limitation the sigma-delta modulator implements a technique called noise-shaping - due to an implemented feedback-loop and integrators the quantization noise is pushed to higher frequencies and thus much lower oversampling rates are sufficient to achieve high resolutions.

This driver is contained in `sd24_b.c`, with `sd24_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	1170
CCS 4.2.1	Size	552
CCS 4.2.1	Speed	550
IAR 5.51.6	None	744
IAR 5.51.6	Size	452
IAR 5.51.6	Speed	504
MSPGCC 4.8.0	None	1792
MSPGCC 4.8.0	Size	654
MSPGCC 4.8.0	Speed	642

### 28.2 API Functions

#### Functions

- void [SD24\\_B\\_clearInterrupt](#) (uint32\_t baseAddress, uint8\_t converter, uint16\_t mask)
- void [SD24\\_B\\_configureConverter](#) (uint32\_t baseAddress, uint8\_t converter, uint8\_t alignment, uint8\_t startSelect, uint8\_t conversionMode)
- void [SD24\\_B\\_configureConverterAdvanced](#) (uint32\_t baseAddress, uint8\_t converter, uint8\_t alignment, uint8\_t startSelect, uint8\_t conversionMode, uint8\_t dataFormat, uint8\_t sampleDelay, uint16\_t oversampleRatio, uint8\_t gain)



- void [SD24\\_B\\_configureDMATrigger](#) (uint32\_t baseAddress, uint16\_t interruptFlag)
- void [SD24\\_B\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t converter, uint16\_t mask)
- void [SD24\\_B\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t converter, uint16\_t mask)
- uint16\_t [SD24\\_B\\_getHighWordResults](#) (uint32\_t baseAddress, uint8\_t converter)
- uint16\_t [SD24\\_B\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t converter, uint16\_t mask)
- uint32\_t [SD24\\_B\\_getResults](#) (uint32\_t baseAddress, uint8\_t converter)
- void [SD24\\_B\\_init](#) (uint32\_t baseAddress, uint16\_t clockSourceSelect, uint16\_t clockPreDivider, uint16\_t clockDivider, uint16\_t referenceSelect)
- void [SD24\\_B\\_setConverterDataFormat](#) (uint32\_t baseAddress, uint8\_t converter, uint8\_t dataFormat)
- void [SD24\\_B\\_setGain](#) (uint32\_t baseAddress, uint8\_t converter, uint8\_t gain)
- void [SD24\\_B\\_setInterruptDelay](#) (uint32\_t baseAddress, uint8\_t converter, uint8\_t sampleDelay)
- void [SD24\\_B\\_setOversampling](#) (uint32\_t baseAddress, uint8\_t converter, uint16\_t oversampleRatio)
- void [SD24\\_B\\_startConverterConversion](#) (uint32\_t baseAddress, uint8\_t converter)
- void [SD24\\_B\\_startGroupConversion](#) (uint32\_t baseAddress, uint8\_t group)
- void [SD24\\_B\\_stopConverterConversion](#) (uint32\_t baseAddress, uint8\_t converter)
- void [SD24\\_B\\_stopGroupConversion](#) (uint32\_t baseAddress, uint8\_t group)

## 28.2.1 Detailed Description

The SD24\_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 SD24\_B.

The SD24\_B initialization and conversion functions are

- [SD24\\_B\\_init\(\)](#)
- [SD24\\_B\\_configureConverter\(\)](#)
- [SD24\\_B\\_configureConverterAdvanced\(\)](#)
- [SD24\\_B\\_startGroupConversion\(\)](#)
- [SD24\\_B\\_stopGroupConversion\(\)](#)
- [SD24\\_B\\_stopConverterConversion\(\)](#)
- [SD24\\_B\\_startConverterConversion\(\)](#)
- [SD24\\_B\\_configureDMATrigger\(\)](#)
- [SD24\\_B\\_getResults\(\)](#)
- [SD24\\_B\\_getHighWordResults\(\)](#)

The SD24\_B interrupts are handled by

- [SD24\\_B\\_enableInterrupt\(\)](#)
- [SD24\\_B\\_disableInterrupt\(\)](#)
- [SD24\\_B\\_clearInterrupt\(\)](#)
- [SD24\\_B\\_getInterruptStatus\(\)](#)

Auxiliary features of the SD24\_B are handled by

- [SD24\\_B\\_setConverterDataFormat\(\)](#)
- [SD24\\_B\\_setInterruptDelay\(\)](#)
- [SD24\\_B\\_setOversampling\(\)](#)
- [SD24\\_B\\_setGain\(\)](#)

## 28.2.2 Function Documentation

### 28.2.2.1 SD24\_B\_clearInterrupt

Clears interrupts for the SD24\_B Module.

#### 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	28
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

#### Prototype:

```
void
SD24_B_clearInterrupt(uint32_t baseAddress,
                      uint8_t converter,
                      uint16_t mask)
```

#### Description:

This function clears interrupt flags for the SD24\_B module.

#### Parameters:

**baseAddress** is the base address of the SD24\_B module.

**converter** is the selected converter. Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

**mask** is the bit mask of the converter interrupt sources to clear. Mask value is the logical OR of any of the following:

- SD24\_B\_CONVERTER\_INTERRUPT
  - SD24\_B\_CONVERTER\_OVERFLOW\_INTERRUPT
- Modified bits are SD24OVIFGx of SD24BIFG register.

#### Returns:

None

### 28.2.2.2 SD24\_B\_configureConverter

Configure SD24\_B converter.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	76
CCS 4.2.1	Size	44
CCS 4.2.1	Speed	42
IAR 5.51.6	None	52
IAR 5.51.6	Size	42
IAR 5.51.6	Speed	42
MSPGCC 4.8.0	None	102
MSPGCC 4.8.0	Size	42
MSPGCC 4.8.0	Speed	42

**Prototype:**

```
void
SD24_B_configureConverter(uint32_t baseAddress,
                          uint8_t converter,
                          uint8_t alignment,
                          uint8_t startSelect,
                          uint8_t conversionMode)
```

**Description:**

This function initializes a converter of the SD24\_B module. Upon completion the converter will be ready for a conversion and can be started with the [SD24\\_B\\_startGroupConversion\(\)](#) or [SD24\\_B\\_startConverterConversion\(\)](#) depending on the startSelect parameter. Additional configuration such as data format can be configured in [SD24\\_B\\_setConverterDataFormat\(\)](#).

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter that will be configured. Check check datasheet for available converters on device.

Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

**alignment** selects how the data will be aligned in result Valid values are:

- SD24\_B\_ALIGN\_RIGHT [Default]
- SD24\_B\_ALIGN\_LEFT

Modified bits are **SD24ALGN** of **SD24BCCTLx** register.

**startSelect** selects what will trigger the start of the converter Valid values are:

- SD24\_B\_CONVERSION\_SELECT\_SD24SC [Default]
- SD24\_B\_CONVERSION\_SELECT\_EXT1
- SD24\_B\_CONVERSION\_SELECT\_EXT2
- SD24\_B\_CONVERSION\_SELECT\_EXT3
- SD24\_B\_CONVERSION\_SELECT\_GROUP0
- SD24\_B\_CONVERSION\_SELECT\_GROUP1
- SD24\_B\_CONVERSION\_SELECT\_GROUP2
- SD24\_B\_CONVERSION\_SELECT\_GROUP3

Modified bits are **SD24SCSx** of **SD24BCCTLx** register.

**conversionMode** determines whether the converter will do continuous samples or a single sample Valid values are:

- SD24\_B\_CONTINUOUS\_MODE [Default]
- SD24\_B\_SINGLE\_MODE

Modified bits are **SD24SNGL** of **SD24BCCTLx** register.

**Returns:**

None

### 28.2.2.3 SD24\_B\_configureConverterAdvanced

Configure SD24\_B converter - Advanced Configure.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	158
CCS 4.2.1	Size	104
CCS 4.2.1	Speed	102
IAR 5.51.6	None	130
IAR 5.51.6	Size	94
IAR 5.51.6	Speed	94
MSPGCC 4.8.0	None	244
MSPGCC 4.8.0	Size	104
MSPGCC 4.8.0	Speed	104

#### Prototype:

```
void
SD24_B_configureConverterAdvanced(uint32_t baseAddress,
                                uint8_t converter,
                                uint8_t alignment,
                                uint8_t startSelect,
                                uint8_t conversionMode,
                                uint8_t dataFormat,
                                uint8_t sampleDelay,
                                uint16_t oversampleRatio,
                                uint8_t gain)
```

#### Description:

This function initializes a converter of the SD24\_B module. Upon completion the converter will be ready for a conversion and can be started with the [SD24\\_B\\_startGroupConversion\(\)](#) or [SD24\\_B\\_startConverterConversion\(\)](#) depending on the startSelect parameter.

#### Parameters:

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter that will be configured. Check check datasheet for available converters on device.

Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

**alignment** selects how the data will be aligned in result Valid values are:

- SD24\_B\_ALIGN\_RIGHT [Default]
- SD24\_B\_ALIGN\_LEFT

Modified bits are SD24ALGN of SD24BCCTLx register.

**startSelect** selects what will trigger the start of the converter Valid values are:

- SD24\_B\_CONVERSION\_SELECT\_SD24SC [Default]
- SD24\_B\_CONVERSION\_SELECT\_EXT1
- SD24\_B\_CONVERSION\_SELECT\_EXT2
- SD24\_B\_CONVERSION\_SELECT\_EXT3
- SD24\_B\_CONVERSION\_SELECT\_GROUP0
- SD24\_B\_CONVERSION\_SELECT\_GROUP1
- SD24\_B\_CONVERSION\_SELECT\_GROUP2
- SD24\_B\_CONVERSION\_SELECT\_GROUP3

Modified bits are SD24SCSx of SD24BCCTLx register.

**conversionMode** determines whether the converter will do continuous samples or a single sample Valid values are:

- SD24\_B\_CONTINUOUS\_MODE [Default]
- SD24\_B\_SINGLE\_MODE

Modified bits are SD24SNGL of SD24BCCTLx register.

**dataFormat** selects how the data format of the results Valid values are:

- SD24\_B\_DATA\_FORMAT\_BINARY [Default]
- SD24\_B\_DATA\_FORMAT\_2COMPLEMENT

Modified bits are SD24DFx of SD24BCCTLx register.

**sampleDelay** selects the delay for the interrupt Valid values are:

- SD24\_B\_FOURTH\_SAMPLE\_INTERRUPT [Default]
- SD24\_B\_THIRD\_SAMPLE\_INTERRUPT
- SD24\_B\_SECOND\_SAMPLE\_INTERRUPT
- SD24\_B\_FIRST\_SAMPLE\_INTERRUPT

Modified bits are SD24INTDLYx of SD24INCTLx register.

**oversampleRatio** selects oversampling ratio for the converter Valid values are:

- SD24\_B\_OVERSAMPLE\_32
- SD24\_B\_OVERSAMPLE\_64
- SD24\_B\_OVERSAMPLE\_128
- SD24\_B\_OVERSAMPLE\_256
- SD24\_B\_OVERSAMPLE\_512
- SD24\_B\_OVERSAMPLE\_1024

Modified bits are SD24OSRx of SD24BOSRx register.

**gain** selects the gain for the converter Valid values are:

- SD24\_B\_GAIN\_1 [Default]
- SD24\_B\_GAIN\_2
- SD24\_B\_GAIN\_4
- SD24\_B\_GAIN\_8
- SD24\_B\_GAIN\_16
- SD24\_B\_GAIN\_32
- SD24\_B\_GAIN\_64
- SD24\_B\_GAIN\_128

Modified bits are SD24GAINx of SD24BINCTLx register.

**Returns:**

None

#### 28.2.2.4 SD24\_B\_configureDMATrigger

Configures the converter that triggers a DMA transfer.

**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	14
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

**Prototype:**

```
void
SD24_B_configureDMATrigger(uint32_t baseAddress,
                           uint16_t interruptFlag)
```

**Description:**

This function chooses which interrupt will trigger a DMA transfer.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**interruptFlag** selects the converter interrupt that triggers a DMA transfer. Valid values are:

- SD24\_B\_DMA\_TRIGGER\_IFG0
- SD24\_B\_DMA\_TRIGGER\_IFG1
- SD24\_B\_DMA\_TRIGGER\_IFG2
- SD24\_B\_DMA\_TRIGGER\_IFG3
- SD24\_B\_DMA\_TRIGGER\_IFG4
- SD24\_B\_DMA\_TRIGGER\_IFG5
- SD24\_B\_DMA\_TRIGGER\_IFG6
- SD24\_B\_DMA\_TRIGGER\_IFG7
- SD24\_B\_DMA\_TRIGGER\_TRGIFG

Modified bits are **SD24DMAx** of **SD24BCTL1** register.

**Returns:**

None

### 28.2.2.5 SD24\_B\_disableInterrupt

Disables interrupts for the SD24\_B Module.

**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	28
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	64
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

**Prototype:**

```
void
SD24_B_disableInterrupt(uint32_t baseAddress,
                        uint8_t converter,
                        uint16_t mask)
```

**Description:**

This function disables interrupts for the SD24\_B module.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** is the selected converter. Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

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

- SD24\_B\_CONVERTER\_INTERRUPT
  - SD24\_B\_CONVERTER\_OVERFLOW\_INTERRUPT
- Modified bits are SD24OVIEx of SD24BIE register.

Modified bits of SD24BIE register.

**Returns:**

None

### 28.2.2.6 SD24\_B\_enableInterrupt

Enables interrupts for the SD24\_B Module.

**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	28
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	14
MSPGCC 4.8.0	None	62
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

**Prototype:**

```
void
SD24_B_enableInterrupt(uint32_t baseAddress,
                      uint8_t converter,
                      uint16_t mask)
```

**Description:**

This function enables interrupts for the SD24\_B module. Does not clear interrupt flags.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** is the selected converter. Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

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

- SD24\_B\_CONVERTER\_INTERRUPT
  - SD24\_B\_CONVERTER\_OVERFLOW\_INTERRUPT
- Modified bits are SD24OVIEx of SD24BIE register.

**Returns:**

None

### 28.2.2.7 SD24\_B\_getHighWordResults

Returns the high word results for a converter.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	50
CCS 4.2.1	Size	16
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	60
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

#### Prototype:

```
uint16_t
SD24_B_getHighWordResults(uint32_t baseAddress,
                           uint8_t converter)
```

#### Description:

This function gets the results from the SD24MEMHx register and returns it.

#### Parameters:

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter who's results will be returned Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

#### Returns:

Result of conversion

### 28.2.2.8 SD24\_B\_getInterruptStatus

Returns the interrupt status for the SD24\_B Module.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	38
CCS 4.2.1	Size	18
CCS 4.2.1	Speed	22
IAR 5.51.6	None	20
IAR 5.51.6	Size	14
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26



**Prototype:**

```
uint16_t
SD24_B_getInterruptStatus(uint32_t baseAddress,
                          uint8_t converter,
                          uint16_t mask)
```

**Description:**

This function returns interrupt flag statuses for the SD24\_B module.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** is the selected converter. Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

**mask** is the bit mask of the converter interrupt sources to return. Mask value is the logical OR of any of the following:

- SD24\_B\_CONVERTER\_INTERRUPT
- SD24\_B\_CONVERTER\_OVERFLOW\_INTERRUPT

**Returns:**

Logical OR of any of the following:

- SD24\_B\_CONVERTER\_INTERRUPT
  - SD24\_B\_CONVERTER\_OVERFLOW\_INTERRUPT
- indicating the status of the masked interrupts

### 28.2.2.9 SD24\_B\_getResults

Returns the results for a converter.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	66
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	22
IAR 5.51.6	None	30
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	20
MSPGCC 4.8.0	None	118
MSPGCC 4.8.0	Size	42
MSPGCC 4.8.0	Speed	42

**Prototype:**

```
uint32_t
SD24_B_getResults(uint32_t baseAddress,
                  uint8_t converter)
```

**Description:**

This function gets the results from the SD24BMEMLx and SD24MEMHx registers and concatenates them to form a long. The actual result is a maximum 24 bits.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter who's results will be returned Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

**Returns:**

Result of conversion

### 28.2.2.10 SD24\_B\_init

Initializes the SD24\_B Module.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	98
CCS 4.2.1	Size	62
CCS 4.2.1	Speed	60
IAR 5.51.6	None	88
IAR 5.51.6	Size	78
IAR 5.51.6	Speed	78
MSPGCC 4.8.0	None	160
MSPGCC 4.8.0	Size	58
MSPGCC 4.8.0	Speed	58

**Prototype:**

```
void
SD24_B_init(uint32_t baseAddress,
            uint16_t clockSourceSelect,
            uint16_t clockPreDivider,
            uint16_t clockDivider,
            uint16_t referenceSelect)
```

**Description:**

This function initializes the SD24\_B module sigma-delta analog-to-digital conversions. Specifically the function sets up the clock source for the SD24\_B core to use for conversions. Upon completion of the initialization the SD24\_B interrupt registers will be reset and the given parameters will be set. The converter configuration settings are independent of this function. The values you choose for the clock divider and predivider are used to determine the effective clock frequency. The formula used is:  $f_{sd24} = f_{clk} / (divider * predivider)$

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**clockSourceSelect** selects the clock that will be used as the SD24\_B core Valid values are:

- SD24\_B\_CLOCKSOURCE\_MCLK [Default]
  - SD24\_B\_CLOCKSOURCE\_SMCLK
  - SD24\_B\_CLOCKSOURCE\_ACLK
  - SD24\_B\_CLOCKSOURCE\_SD24CLK
- Modified bits are **SD24SSEL** of **SD24BCTL0** register.

**clockPreDivider** selects the amount that the clock will be predivided Valid values are:

- SD24\_B\_PRECLOCKDIVIDER\_1 [Default]
- SD24\_B\_PRECLOCKDIVIDER\_2
- SD24\_B\_PRECLOCKDIVIDER\_4
- SD24\_B\_PRECLOCKDIVIDER\_8

- SD24\_B\_PRECLOCKDIVIDER\_16
- SD24\_B\_PRECLOCKDIVIDER\_32
- SD24\_B\_PRECLOCKDIVIDER\_64
- SD24\_B\_PRECLOCKDIVIDER\_128

Modified bits are SD24PDIVx of SD24BCTL0 register.

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

- SD24\_B\_CLOCKDIVIDER\_1 [Default]
- SD24\_B\_CLOCKDIVIDER\_2
- SD24\_B\_CLOCKDIVIDER\_3
- SD24\_B\_CLOCKDIVIDER\_4
- SD24\_B\_CLOCKDIVIDER\_5
- SD24\_B\_CLOCKDIVIDER\_6
- SD24\_B\_CLOCKDIVIDER\_7
- SD24\_B\_CLOCKDIVIDER\_8
- SD24\_B\_CLOCKDIVIDER\_9
- SD24\_B\_CLOCKDIVIDER\_10
- SD24\_B\_CLOCKDIVIDER\_11
- SD24\_B\_CLOCKDIVIDER\_12
- SD24\_B\_CLOCKDIVIDER\_13
- SD24\_B\_CLOCKDIVIDER\_14
- SD24\_B\_CLOCKDIVIDER\_15
- SD24\_B\_CLOCKDIVIDER\_16
- SD24\_B\_CLOCKDIVIDER\_17
- SD24\_B\_CLOCKDIVIDER\_18
- SD24\_B\_CLOCKDIVIDER\_19
- SD24\_B\_CLOCKDIVIDER\_20
- SD24\_B\_CLOCKDIVIDER\_21
- SD24\_B\_CLOCKDIVIDER\_22
- SD24\_B\_CLOCKDIVIDER\_23
- SD24\_B\_CLOCKDIVIDER\_24
- SD24\_B\_CLOCKDIVIDER\_25
- SD24\_B\_CLOCKDIVIDER\_26
- SD24\_B\_CLOCKDIVIDER\_27
- SD24\_B\_CLOCKDIVIDER\_28
- SD24\_B\_CLOCKDIVIDER\_29
- SD24\_B\_CLOCKDIVIDER\_30
- SD24\_B\_CLOCKDIVIDER\_31
- SD24\_B\_CLOCKDIVIDER\_32

Modified bits are SD24DIVx of SD24BCTL0 register.

**referenceSelect** selects the reference source for the SD24\_B core Valid values are:

- SD24\_B\_REF\_EXTERNAL [Default]
- SD24\_B\_REF\_INTERNAL

Modified bits are SD24REFS of SD24BCTL0 register.

**Returns:**

None

### 28.2.2.11 SD24\_B\_setConverterDataFormat

Set SD24\_B converter data format.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	58
CCS 4.2.1	Size	22
CCS 4.2.1	Speed	22
IAR 5.51.6	None	28
IAR 5.51.6	Size	20
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	96
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

**Prototype:**

```
void
SD24_B_setConverterDataFormat(uint32_t baseAddress,
                             uint8_t converter,
                             uint8_t dataFormat)
```

**Description:**

This function sets the converter format so that the resulting data can be viewed in either binary or 2's complement.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter that will be configured. Check check datasheet for available converters on device.

Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

**dataFormat** selects how the data format of the results Valid values are:

- SD24\_B\_DATA\_FORMAT\_BINARY [Default]
- SD24\_B\_DATA\_FORMAT\_2COMPLEMENT

Modified bits are **SD24DFx** of **SD24BCCTLx** register.

**Returns:**

None

### 28.2.2.12 SD24\_B\_setGain

Configures the gain for the converter.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	60
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	30
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

**Prototype:**

```
void
SD24_B_setGain(uint32_t baseAddress,
               uint8_t converter,
               uint8_t gain)
```

**Description:**

This function configures the gain for a single converter.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter that will be configured Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

**gain** selects the gain for the converter Valid values are:

- SD24\_B\_GAIN\_1 [Default]
- SD24\_B\_GAIN\_2
- SD24\_B\_GAIN\_4
- SD24\_B\_GAIN\_8
- SD24\_B\_GAIN\_16
- SD24\_B\_GAIN\_32
- SD24\_B\_GAIN\_64
- SD24\_B\_GAIN\_128

Modified bits are **SD24GAINx** of **SD24BINCTLx** register.

**Returns:**

None

### 28.2.2.13 SD24\_B\_setInterruptDelay

Configures the delay for an interrupt to trigger.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	60
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	30
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	84
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

**Prototype:**

```
void
SD24_B_setInterruptDelay(uint32_t baseAddress,
                         uint8_t converter,
                         uint8_t sampleDelay)
```

**Description:**

This function configures the delay for the first interrupt service request for the corresponding converter. This feature delays the interrupt request for a completed conversion by up to four conversion cycles allowing the digital filter to settle prior to generating an interrupt request.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter that will be stopped Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

**sampleDelay** selects the delay for the interrupt Valid values are:

- SD24\_B\_FOURTH\_SAMPLE\_INTERRUPT [Default]
- SD24\_B\_THIRD\_SAMPLE\_INTERRUPT
- SD24\_B\_SECOND\_SAMPLE\_INTERRUPT
- SD24\_B\_FIRST\_SAMPLE\_INTERRUPT

Modified bits are SD24INTDLYx of SD24INCTLx register.

**Returns:**

None

## 28.2.2.14 SD24\_B\_setOversampling

Configures the oversampling ratio for a converter.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	62
CCS 4.2.1	Size	22
CCS 4.2.1	Speed	22
IAR 5.51.6	None	28
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	18
MSPGCC 4.8.0	None	86
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

**Prototype:**

```
void
SD24_B_setOversampling(uint32_t baseAddress,
                       uint8_t converter,
                       uint16_t oversampleRatio)
```

**Description:**

This function configures the oversampling ratio for a given converter.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter that will be configured Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2

- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

**oversampleRatio** selects oversampling ratio for the converter Valid values are:

- SD24\_B\_OVERSAMPLE\_32
- SD24\_B\_OVERSAMPLE\_64
- SD24\_B\_OVERSAMPLE\_128
- SD24\_B\_OVERSAMPLE\_256
- SD24\_B\_OVERSAMPLE\_512
- SD24\_B\_OVERSAMPLE\_1024

Modified bits are **SD24OSRx** of **SD24BOSRx** register.

**Returns:**

None

### 28.2.2.15 SD24\_B\_startConverterConversion

Start Conversion for Converter.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	52
CCS 4.2.1	Size	22
CCS 4.2.1	Speed	22
IAR 5.51.6	None	30
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	16
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

**Prototype:**

```
void
SD24_B_startConverterConversion(uint32_t baseAddress,
                                uint8_t converter)
```

**Description:**

This function starts a single converter.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter that will be started Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

Modified bits are **SD24SC** of **SD24BCCTLx** register.

**Returns:**

None

### 28.2.2.16 SD24\_B\_startGroupConversion

Start Conversion Group.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	86
CCS 4.2.1	Size	42
CCS 4.2.1	Speed	42
IAR 5.51.6	None	60
IAR 5.51.6	Size	40
IAR 5.51.6	Speed	46
MSPGCC 4.8.0	None	152
MSPGCC 4.8.0	Size	48
MSPGCC 4.8.0	Speed	42

#### Prototype:

```
void
SD24_B_startGroupConversion(uint32_t baseAddress,
                           uint8_t group)
```

#### Description:

This function starts all the converters that are associated with a group. To set a converter to a group use the [SD24\\_B\\_configureConverter\(\)](#) function.

#### Parameters:

**baseAddress** is the base address of the SD24\_B module.

**group** selects the group that will be started Valid values are:

- SD24\_B\_GROUP0
- SD24\_B\_GROUP1
- SD24\_B\_GROUP2
- SD24\_B\_GROUP3

Modified bits are **SD24DGRP<sub>x</sub>SC** of **SD24BCTL1** register.

#### Returns:

None

### 28.2.2.17 SD24\_B\_stopConverterConversion

Stop Conversion for Converter.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	52
CCS 4.2.1	Size	22
CCS 4.2.1	Speed	22
IAR 5.51.6	None	30
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	16
MSPGCC 4.8.0	None	76
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

#### Prototype:

```
void
SD24_B_stopConverterConversion(uint32_t baseAddress,
                              uint8_t converter)
```



**Description:**

This function stops a single converter.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**converter** selects the converter that will be stopped Valid values are:

- SD24\_B\_CONVERTER\_0
- SD24\_B\_CONVERTER\_1
- SD24\_B\_CONVERTER\_2
- SD24\_B\_CONVERTER\_3
- SD24\_B\_CONVERTER\_4
- SD24\_B\_CONVERTER\_5
- SD24\_B\_CONVERTER\_6
- SD24\_B\_CONVERTER\_7

Modified bits are **SD24SC** of **SD24BCCTLx** register.

**Returns:**

None

### 28.2.2.18 SD24\_B\_stopGroupConversion

Stop Conversion Group.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	86
CCS 4.2.1	Size	42
CCS 4.2.1	Speed	42
IAR 5.51.6	None	60
IAR 5.51.6	Size	40
IAR 5.51.6	Speed	46
MSPGCC 4.8.0	None	152
MSPGCC 4.8.0	Size	48
MSPGCC 4.8.0	Speed	42

**Prototype:**

```
void
SD24_B_stopGroupConversion(uint32_t baseAddress,
                           uint8_t group)
```

**Description:**

This function stops all the converters that are associated with a group. To set a converter to a group use the [SD24\\_B\\_configureConverter\(\)](#) function.

**Parameters:**

**baseAddress** is the base address of the SD24\_B module.

**group** selects the group that will be stopped Valid values are:

- SD24\_B\_GROUP0
- SD24\_B\_GROUP1
- SD24\_B\_GROUP2
- SD24\_B\_GROUP3

Modified bits are **SD24DGRPSC** of **SD24BCTL1** register.

**Returns:**

None

## 28.3 Programming Example

The following example shows how to initialize and use the SD24\_B API to start a single channel, single conversion.

```

unsigned long results;

SD24_B_init(SD24_BASE,
            SD24_B_CLOCKSOURCE_SMCLK,
            SD24_B_PRECLOCKDIVIDER_1,
            SD24_B_CLOCKDIVIDER_1,
            SD24_B_REF_INTERNAL);                                // Select internal REF
                                                                // Select SMCLK as SD24_B clock source

SD24_B_configureConverter(SD24_BASE,
                          SD24_B_CONVERTER_2,
                          SD24_B_ALIGN_RIGHT,
                          SD24_B_CONVERSION_SELECT_SD24SC,
                          SD24_B_SINGLE_MODE);

__delay_cycles(0x3600);                                         // Delay for 1.5V REF startup

while (1)
{
    SD24_B_startConverterConversion(SD24_BASE,
                                     SD24_B_CONVERTER_2);      // Set b

    // Poll interrupt flag for channel 2
    while( SD24_B_getInterruptStatus(SD24_BASE,
                                     SD24_B_CONVERTER_2
                                     SD24_CONVERTER_INTERRUPT) == 0 );

    results = SD24_B_getResults(SD24_BASE,
                                SD24_B_CONVERTER_2);           // Save CH2 results (clears IFG)

    __no_operation();                                           // SET BREAKPOINT HERE
}

```

## 29 SFR Module

Introduction .....	327
API Functions .....	327
Programming Example .....	332

### 29.1 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

### 29.2 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)

#### 29.2.1 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  $\sim$ RST/NMI pin is controlled by

- [SFR\\_setResetPinPullResistor\(\)](#)
- [SFR\\_setNMIEdge\(\)](#)
- [SFR\\_setResetNMIPinFunction\(\)](#)

## 29.2.2 Function Documentation

### 29.2.2.1 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 should be cleared Mask value is the logical OR of any of the following:

- **SFR\_JTAG\_OUTBOX\_INTERRUPT** - JTAG outbox interrupt enable
- **SFR\_JTAG\_INBOX\_INTERRUPT** - JTAG inbox interrupt enable
- **SFR\_NMI\_PIN\_INTERRUPT** - NMI pin interrupt enable, if NMI function is chosen
- **SFR\_VACANT\_MEMORY\_ACCESS\_INTERRUPT** - Vacant memory access interrupt enable
- **SFR\_OSCILLATOR\_FAULT\_INTERRUPT** - Oscillator fault interrupt enable
- **SFR\_WATCHDOG\_INTERVAL\_TIMER\_INTERRUPT** - Watchdog interval timer interrupt enable
- **SFR\_FLASH\_CONTROLLER\_ACCESS\_VIOLATION\_INTERRUPT** - Flash controller access violation interrupt enable

#### Returns:

None

### 29.2.2.2 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 enable
- **SFR\_JTAG\_INBOX\_INTERRUPT** - JTAG inbox interrupt enable
- **SFR\_NMI\_PIN\_INTERRUPT** - NMI pin interrupt enable, if NMI function is chosen
- **SFR\_VACANT\_MEMORY\_ACCESS\_INTERRUPT** - Vacant memory access interrupt enable
- **SFR\_OSCILLATOR\_FAULT\_INTERRUPT** - Oscillator fault interrupt enable
- **SFR\_WATCHDOG\_INTERVAL\_TIMER\_INTERRUPT** - Watchdog interval timer interrupt enable
- **SFR\_FLASH\_CONTROLLER\_ACCESS\_VIOLATION\_INTERRUPT** - Flash controller access violation interrupt enable

**Returns:**

None

### 29.2.2.3 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.

**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 enable
- **SFR\_JTAG\_INBOX\_INTERRUPT** - JTAG inbox interrupt enable
- **SFR\_NMI\_PIN\_INTERRUPT** - NMI pin interrupt enable, if NMI function is chosen
- **SFR\_VACANT\_MEMORY\_ACCESS\_INTERRUPT** - Vacant memory access interrupt enable
- **SFR\_OSCILLATOR\_FAULT\_INTERRUPT** - Oscillator fault interrupt enable
- **SFR\_WATCHDOG\_INTERVAL\_TIMER\_INTERRUPT** - Watchdog interval timer interrupt enable
- **SFR\_FLASH\_CONTROLLER\_ACCESS\_VIOLATION\_INTERRUPT** - Flash controller access violation interrupt enable

**Returns:**

None

## 29.2.2.4 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 enable
- **SFR\_JTAG\_INBOX\_INTERRUPT** - JTAG inbox interrupt enable
- **SFR\_NMI\_PIN\_INTERRUPT** - NMI pin interrupt enable, if NMI function is chosen
- **SFR\_VACANT\_MEMORY\_ACCESS\_INTERRUPT** - Vacant memory access interrupt enable
- **SFR\_OSCILLATOR\_FAULT\_INTERRUPT** - Oscillator fault interrupt enable
- **SFR\_WATCHDOG\_INTERVAL\_TIMER\_INTERRUPT** - Watchdog interval timer interrupt enable
- **SFR\_FLASH\_CONTROLLER\_ACCESS\_VIOLATION\_INTERRUPT** - Flash controller access violation interrupt enable

**Returns:**

Logical OR of any of the following:

- **SFR\_JTAG\_OUTBOX\_INTERRUPT** JTAG outbox interrupt enable
- **SFR\_JTAG\_INBOX\_INTERRUPT** JTAG inbox interrupt enable
- **SFR\_NMI\_PIN\_INTERRUPT** NMI pin interrupt enable, if NMI function is chosen
- **SFR\_VACANT\_MEMORY\_ACCESS\_INTERRUPT** Vacant memory access interrupt enable

- **SFR\_OSCILLATOR\_FAULT\_INTERRUPT** Oscillator fault interrupt enable
  - **SFR\_WATCHDOG\_INTERVAL\_TIMER\_INTERRUPT** Watchdog interval timer interrupt enable
  - **SFR\_FLASH\_CONTROLLER\_ACCESS\_VIOLATION\_INTERRUPT** Flash controller access violation interrupt enable
- indicating the status of the masked interrupts

### 29.2.2.5 SFR\_setNMIEdge

Sets the edge direction that will assert an NMI from a signal on the  $\sim$ 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  $\sim$ RST/NMI pin if the NMI function is active. To activate the NMI function of the  $\sim$ RST/NMI use the [SFR\\_setResetNMIPinFunction\(\)](#) passing SFR\_RESETPINFUNC\_NMI into the resetPinFunction parameter.

#### Parameters:

**edgeDirection** is the direction that the signal on the  $\sim$ RST/NMI pin should go to signal an interrupt, if enabled. Valid values are:

- **SFR\_NMI\_RISINGEDGE** [Default]
  - **SFR\_NMI\_FALLINGEDGE**
- Modified bits are **SYSNMIIES** of **SFRRPCR** register.

#### Returns:

None

### 29.2.2.6 SFR\_setResetNMIPinFunction

Sets the function of the  $\sim$ 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  $\sim$ RST/NMI pin, whether in reset mode which will assert a reset if a low signal is observed on that pin, or an NMI which will assert an interrupt from an edge of the signal dependent on the setting of the edgeDirection parameter in [SFR\\_setNMIEdge\(\)](#).

**Parameters:**

**resetPinFunction** is the function that the  $\sim$ RST/NMI pin should take on. Valid values are:

- **SFR\_RESETPINFUNC\_RESET** [Default]
- **SFR\_RESETPINFUNC\_NMI**  
Modified bits are **SYSNMI** of **SFRRPCR** register.

**Returns:**

None

### 29.2.2.7 SFR\_setResetPinPullResistor

Sets the pull-up/down resistor on the  $\sim$ 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  $\sim$ RST/NMI pin to the settings from the pullResistorSetup parameter.

**Parameters:**

**pullResistorSetup** is the selection of how the pull-up/down resistor on the  $\sim$ RST/NMI pin should be setup or disabled. Valid values are:

- **SFR\_RESISTORDISABLE**
- **SFR\_RESISTORENABLE\_PULLUP** [Default]
- **SFR\_RESISTORENABLE\_PULLDOWN**  
Modified bits are **SYSRSTUP** of **SFRRPCR** register.

**Returns:**

None

## 29.3 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));
```

## 30 SYS Module

Introduction .....	334
API Functions .....	334
Programming Example .....	345

### 30.1 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 with 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

### 30.2 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)

## 30.2.1 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\(\)](#)

## 30.2.2 Function Documentation

### 30.2.2.1 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

### 30.2.2.2 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

### 30.2.2.3 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

### 30.2.2.4 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

### 30.2.2.5 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

### 30.2.2.6 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

### 30.2.2.7 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

### 30.2.2.8 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

### 30.2.2.9 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

### 30.2.2.10 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\_BSEENTRY\_INDICATED**
- **SYS\_BSEENTRY\_NOTINDICATED**  
indicating if a BSL entry sequence was detected

### 30.2.2.11 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.

### 30.2.2.12 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.

### 30.2.2.13 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.

### 30.2.2.14 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

### 30.2.2.15 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

### 30.2.2.16 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

### 30.2.2.17 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

### 30.2.2.18 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

## 30.3 Programming Example

The following example shows how to initialize and use the SYS API

```
SYS_enableBSLProtect(SYS_BASE);
```

## 31 TEC

Introduction .....	346
API Functions .....	346
Programming Example .....	353

### 31.1 Introduction

Timer Event Control (TEC) module is the interface between Timer modules and the external events. This chapter describes the TEC Module.

TEC is a module that connects different Timer modules to each other and routes the external signals to the Timer modules. TEC contains the control registers to configure the routing between the Timer modules, and it also has the enable register bits and the interrupt enable and interrupt flags for external event inputs. TEC features include:

- Enabling of internal and external clear signals
- Routing of internal signals (between Timer\_D instances) and external clear signals
- Support of external fault input signals
- Interrupt vector generation of external fault and clear signals.
- Generating feedback signals to the Timer capture/compare channels to affect the timer outputs

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

### 31.2 API Functions

#### Functions

- void [TEC\\_clearExternalClearStatus](#) (uint32\_t baseAddress)
- void [TEC\\_clearExternalFaultStatus](#) (uint32\_t baseAddress, uint8\_t mask)
- void [TEC\\_clearInterruptFlag](#) (uint32\_t baseAddress, uint8\_t mask)
- void [TEC\\_configureExternalClearInput](#) (uint32\_t baseAddress, uint8\_t signalType, uint8\_t signalHold, uint8\_t polarityBit)
- void [TEC\\_configureExternalFaultInput](#) (uint32\_t baseAddress, uint8\_t selectedExternalFault, uint16\_t signalType, uint8\_t signalHold, uint8\_t polarityBit)
- void [TEC\\_disableAuxiliaryClearSignal](#) (uint32\_t baseAddress)
- void [TEC\\_disableExternalClearInput](#) (uint32\_t baseAddress)
- void [TEC\\_disableExternalFaultInput](#) (uint32\_t baseAddress, uint8\_t channelEventBlock)
- void [TEC\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- void [TEC\\_enableAuxiliaryClearSignal](#) (uint32\_t baseAddress)
- void [TEC\\_enableExternalClearInput](#) (uint32\_t baseAddress)
- void [TEC\\_enableExternalFaultInput](#) (uint32\_t baseAddress, uint8\_t channelEventBlock)
- void [TEC\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- uint8\_t [TEC\\_getExternalClearStatus](#) (uint32\_t baseAddress)
- uint8\_t [TEC\\_getExternalFaultStatus](#) (uint32\_t baseAddress, uint8\_t mask)
- uint8\_t [TEC\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t mask)

#### 31.2.1 Detailed Description

The tec configuration is handled by

- [TEC\\_configureExternalClearInput\(\)](#)

- `TEC_configureExternalFaultInput()`
- `TEC_enableExternalFaultInput()`
- `TEC_disableExternalFaultInput()`
- `TEC_enableExternalClearInput()`
- `TEC_disableExternalClearInput()`
- `TEC_enableAuxiliaryClearSignal()`
- `TEC_disableAuxiliaryClearSignal()`

The interrupt and status operations are handled by

- `TEC_enableExternalFaultInput()`
- `TEC_disableExternalFaultInput()`
- `TEC_clearInterruptFlag()`
- `TEC_getInterruptStatus()`
- `TEC_enableInterrupt()`
- `TEC_disableInterrupt()`
- `TEC_getExternalFaultStatus()`
- `TEC_clearExternalFaultStatus()`
- `TEC_getExternalClearStatus()`
- `TEC_clearExternalClearStatus()`

## 31.2.2 Function Documentation

### 31.2.2.1 `TEC_clearExternalClearStatus`

Clears the Timer Event Control External Clear Status.

**Prototype:**

```
void
TEC_clearExternalClearStatus(uint32_t baseAddress)
```

**Parameters:**

***baseAddress*** is the base address of the TEC module.

**Description:**

Modified bits of **TECxINT** register.

**Returns:**

None

### 31.2.2.2 `TEC_clearExternalFaultStatus`

Clears the Timer Event Control External Fault Status.

**Prototype:**

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

**Parameters:**

***baseAddress*** is the base address of the TEC module.

***mask*** is the masked status flag be cleared Mask value is the logical OR of any of the following:

- `TEC_CE0`
- `TEC_CE1`

- **TEC\_CE2**
- **TEC\_CE3** - (available on TEC5 TEC7)
- **TEC\_CE4** - (available on TEC5 TEC7)
- **TEC\_CE5** - (only available on TEC7)
- **TEC\_CE6** - (only available on TEC7)

**Description:**

Modified bits of **TECxINT** register.

**Returns:**

None

### 31.2.2.3 TEC\_clearInterruptFlag

Clears the Timer Event Control Interrupt flag.

**Prototype:**

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

**Parameters:**

**baseAddress** is the base address of the TEC module.

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

- **TEC\_EXTERNAL\_FAULT\_INTERRUPT** - External fault interrupt flag
- **TEC\_EXTERNAL\_CLEAR\_INTERRUPT** - External clear interrupt flag
- **TEC\_AUXILIARY\_CLEAR\_INTERRUPT** - Auxiliary clear interrupt flag

**Description:**

Modified bits of **TECxINT** register.

**Returns:**

None

### 31.2.2.4 TEC\_configureExternalClearInput

Configures the Timer Event Control External Clear Input.

**Prototype:**

```
void
TEC_configureExternalClearInput(uint32_t baseAddress,
                              uint8_t signalType,
                              uint8_t signalHold,
                              uint8_t polarityBit)
```

**Parameters:**

**baseAddress** is the base address of the TEC module.

**signalType** is the selected signal type Valid values are:

- **TEC\_EXTERNAL\_CLEAR\_SIGNALTYPE\_EDGE\_SENSITIVE** [Default]
- **TEC\_EXTERNAL\_CLEAR\_SIGNALTYPE\_LEVEL\_SENSITIVE**

**signalHold** is the selected signal hold Valid values are:

- **TEC\_EXTERNAL\_CLEAR\_SIGNAL\_NOT\_HELD** [Default]
- **TEC\_EXTERNAL\_CLEAR\_SIGNAL\_HELD**

**polarityBit** is the selected signal type Valid values are:

- **TEC\_EXTERNAL\_CLEAR\_POLARITY\_FALLING\_EDGE\_OR\_LOW\_LEVEL** [Default]
- **TEC\_EXTERNAL\_CLEAR\_POLARITY\_RISING\_EDGE\_OR\_HIGH\_LEVEL**



**Description:**

Modified bits of **TECxCTL2** register.

**Returns:**

None

### 31.2.2.5 TEC\_configureExternalFaultInput

Configures the Timer Event Control External Fault Input.

**Prototype:**

```
void
TEC_configureExternalFaultInput (uint32_t baseAddress,
                                uint8_t selectedExternalFault,
                                uint16_t signalType,
                                uint8_t signalHold,
                                uint8_t polarityBit)
```

**Parameters:**

**baseAddress** is the base address of the TEC module.

**selectedExternalFault** is the selected external fault Valid values are:

- **TEC\_EXTERNAL\_FAULT\_0**
- **TEC\_EXTERNAL\_FAULT\_1**
- **TEC\_EXTERNAL\_FAULT\_2**
- **TEC\_EXTERNAL\_FAULT\_3**
- **TEC\_EXTERNAL\_FAULT\_4**
- **TEC\_EXTERNAL\_FAULT\_5**
- **TEC\_EXTERNAL\_FAULT\_6**

**signalType** is the selected signal type Valid values are:

- **TEC\_EXTERNAL\_FAULT\_SIGNALTYPE\_EDGE\_SENSITIVE** [Default]
- **TEC\_EXTERNAL\_FAULT\_SIGNALTYPE\_LEVEL\_SENSITIVE**

**signalHold** is the selected signal hold Valid values are:

- **TEC\_EXTERNAL\_FAULT\_SIGNAL\_NOT\_HELD** [Default]
- **TEC\_EXTERNAL\_FAULT\_SIGNAL\_HELD**

**polarityBit** is the selected signal type Valid values are:

- **TEC\_EXTERNAL\_FAULT\_POLARITY\_FALLING\_EDGE\_OR\_LOW\_LEVEL** [Default]
- **TEC\_EXTERNAL\_FAULT\_POLARITY\_RISING\_EDGE\_OR\_HIGH\_LEVEL**

**Description:**

Modified bits of **TECxCTL2** register.

**Returns:**

None

### 31.2.2.6 TEC\_disableAuxiliaryClearSignal

Disable the Timer Event Control Auxiliary Clear Signal.

**Prototype:**

```
void
TEC_disableAuxiliaryClearSignal (uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TEC module.

**Description:**

Modified bits of **TECxCTL2** register.

**Returns:**

None

### 31.2.2.7 TEC\_disableExternalClearInput

Disable the Timer Event Control External Clear Input.

**Prototype:**

```
void
TEC_disableExternalClearInput (uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TEC module.

**Description:**

Modified bits of **TECxCTL2** register.

**Returns:**

None

### 31.2.2.8 TEC\_disableExternalFaultInput

Disable the Timer Event Control External fault input.

**Prototype:**

```
void
TEC_disableExternalFaultInput (uint32_t baseAddress,
                               uint8_t channelEventBlock)
```

**Parameters:**

**baseAddress** is the base address of the TEC module.

**channelEventBlock** selects the channel event block Valid values are:

- **TEC\_CE0**
- **TEC\_CE1**
- **TEC\_CE2**
- **TEC\_CE3** - (available on TEC5 TEC7)
- **TEC\_CE4** - (available on TEC5 TEC7)
- **TEC\_CE5** - (only available on TEC7)
- **TEC\_CE6** - (only available on TEC7)

**Description:**

Modified bits of **TECxCTL0** register.

**Returns:**

None

### 31.2.2.9 TEC\_disableInterrupt

Disables individual Timer Event Control interrupt sources.

**Prototype:**

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

**Description:**

Disables the indicated Timer Event Control 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 TEC module.

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

- **TEC\_EXTERNAL\_FAULT\_INTERRUPT** - External fault interrupt flag
- **TEC\_EXTERNAL\_CLEAR\_INTERRUPT** - External clear interrupt flag
- **TEC\_AUXILIARY\_CLEAR\_INTERRUPT** - Auxiliary clear interrupt flag

Modified bits of **TECxINT** register.

**Returns:**  
None

### 31.2.2.10 TEC\_enableAuxiliaryClearSignal

Enable the Timer Event Control Auxiliary Clear Signal.

**Prototype:**

```
void
TEC_enableAuxiliaryClearSignal(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TEC module.

**Description:**

Modified bits of **TECxCTL2** register.

**Returns:**  
None

### 31.2.2.11 TEC\_enableExternalClearInput

Enable the Timer Event Control External Clear Input.

**Prototype:**

```
void
TEC_enableExternalClearInput(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TEC module.

**Description:**

Modified bits of **TECxCTL2** register.

**Returns:**  
None

### 31.2.2.12 TEC\_enableExternalFaultInput

Enable the Timer Event Control External fault input.

**Prototype:**

```
void
TEC_enableExternalFaultInput(uint32_t baseAddress,
                             uint8_t channelEventBlock)
```

**Parameters:**

**baseAddress** is the base address of the TEC module.

**channelEventBlock** selects the channel event block Valid values are:

- **TEC\_CEO**

- **TEC\_CE1**
- **TEC\_CE2**
- **TEC\_CE3** - (available on TEC5 TEC7)
- **TEC\_CE4** - (available on TEC5 TEC7)
- **TEC\_CE5** - (only available on TEC7)
- **TEC\_CE6** - (only available on TEC7)

**Description:**

Modified bits of **TECxCTL0** register.

**Returns:**

None

### 31.2.2.13 TEC\_enableInterrupt

Enables individual Timer Event Control interrupt sources.

**Prototype:**

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

**Description:**

Enables the indicated Timer Event Control 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 TEC module.

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

- **TEC\_EXTERNAL\_FAULT\_INTERRUPT** - External fault interrupt flag
- **TEC\_EXTERNAL\_CLEAR\_INTERRUPT** - External clear interrupt flag
- **TEC\_AUXILIARY\_CLEAR\_INTERRUPT** - Auxiliary clear interrupt flag

Modified bits of **TECxINT** register.

**Returns:**

None

### 31.2.2.14 TEC\_getExternalClearStatus

Gets the current Timer Event Control External Clear Status.

**Prototype:**

```
uint8_t
TEC_getExternalClearStatus (uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TEC module.

**Returns:**

One of the following:

- **TEC\_EXTERNAL\_CLEAR\_DETECTED**
- **TEC\_EXTERNAL\_CLEAR\_NOT\_DETECTED**  
indicating the status of the external clear

### 31.2.2.15 uint8\_t TEC\_getExternalFaultStatus (uint32\_t *baseAddress*, uint8\_t *mask*)

Gets the current Timer Event Control External Fault Status.

This returns the Timer Event Control fault status for the module.

**Parameters:**

***baseAddress*** is the base address of the TEC module.

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

- **TEC\_CE0**
- **TEC\_CE1**
- **TEC\_CE2**
- **TEC\_CE3** - (available on TEC5 TEC7)
- **TEC\_CE4** - (available on TEC5 TEC7)
- **TEC\_CE5** - (only available on TEC7)
- **TEC\_CE6** - (only available on TEC7)

**Returns:**

Logical OR of any of the following:

- **TEC\_CE0**
  - **TEC\_CE1**
  - **TEC\_CE2**
  - **TEC\_CE3** (available on TEC5 TEC7)
  - **TEC\_CE4** (available on TEC5 TEC7)
  - **TEC\_CE5** (only available on TEC7)
  - **TEC\_CE6** (only available on TEC7)
- indicating the external fault status of the masked channel event blocks

### 31.2.2.16 uint8\_t TEC\_getInterruptStatus (uint32\_t *baseAddress*, uint8\_t *mask*)

Gets the current Timer Event Control interrupt status.

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

**Parameters:**

***baseAddress*** is the base address of the TEC module.

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

- **TEC\_EXTERNAL\_FAULT\_INTERRUPT** - External fault interrupt flag
- **TEC\_EXTERNAL\_CLEAR\_INTERRUPT** - External clear interrupt flag
- **TEC\_AUXILIARY\_CLEAR\_INTERRUPT** - Auxiliary clear interrupt flag

**Returns:**

Logical OR of any of the following:

- **TEC\_EXTERNAL\_FAULT\_INTERRUPT** External fault interrupt flag
  - **TEC\_EXTERNAL\_CLEAR\_INTERRUPT** External clear interrupt flag
  - **TEC\_AUXILIARY\_CLEAR\_INTERRUPT** Auxiliary clear interrupt flag
- indicating the status of the masked interrupts

## 31.3 Programming Example

The following example shows how to use the TEC API.

```
{
    TIMER_D_startCounter(TIMER_D1_BASE,
        TIMERD_UP_MODE);

    // Configure TD1 TEC External Clear
    // Need to physically connect P2.0/TD0.2 to P2.7/TEC1CLR
    GPIO_setAsPeripheralModuleFunctionInputPin(
        GPIO_PORT_P2,
        GPIO_PIN7
    );

    // High Level trigger, ext clear enable
    TEC_configureExternalClearInput(TEC1_BASE,

    TEC_EXTERNAL_CLEAR_SIGNALTYPE_LEVEL_SENS,
    TEC_EXTERNAL_CLEAR_SIGNAL_NOT_HELD,
    TEC_EXTERNAL_CLEAR_POLARITY_RISING_EDGE_C
    );

    TEC_enableExternalClearInput(TEC1_BASE);
}
```

## 32 TIMER\_A

Introduction .....	355
API Functions .....	356
Programming Example .....	380

### 32.1 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

## 32.2 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\\_startContinuousMode](#) (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)

## 32.2.1 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\(\)](#)

## 32.2.2 Function Documentation

### 32.2.2.1 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

### 32.2.2.2 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 **TAxCTLn** register.

**Returns:**

None

### 32.2.2.3 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

### 32.2.2.4 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

### 32.2.2.5 `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 captureCompare interrupt.  
Valid values are:

- **TIMER\_A\_CCIE\_CCR0\_INTERRUPT\_ENABLE**
- **TIMER\_A\_CCIE\_CCR0\_INTERRUPT\_DISABLE** [Default]

**timerClear** decides if TIMER\_A clock divider, count direction, count need to be reset. Valid values are:

- **TIMER\_A\_DO\_CLEAR**
- **TIMER\_A\_SKIP\_CLEAR** [Default]

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register and bits of **TAxCCR0** register.

**Returns:**

None

### 32.2.2.6 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 captureCompare interrupt. Valid values are:

- **TIMER\_A\_CCIE\_CCR0\_INTERRUPT\_ENABLE**
- **TIMER\_A\_CCIE\_CCR0\_INTERRUPT\_DISABLE** [Default]

**timerClear** decides if TIMER\_A clock divider, count direction, count need to be reset. Valid values are:

- **TIMER\_A\_DO\_CLEAR**
- **TIMER\_A\_SKIP\_CLEAR** [Default]

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register and bits of **TAxCCR0** register.

**Returns:**

None

### 32.2.2.7 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 `TAxCTLn` register.

**Returns:**

None

### 32.2.2.8 `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

### 32.2.2.9 `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

### 32.2.2.10 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

### 32.2.2.11 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

### 32.2.2.12 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`

### 32.2.2.13 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

### 32.2.2.14 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

### 32.2.2.15 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

### 32.2.2.16 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`

32.2.2.17 `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`

32.2.2.18 `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\_CC1xA
- TIMER\_A\_CAPTURE\_INPUTSELECT\_CC1xB
- TIMER\_A\_CAPTURE\_INPUTSELECT\_GND
- TIMER\_A\_CAPTURE\_INPUTSELECT\_Vcc

**synchronizeCaptureSource** decides if capture source should be synchronized with timer clock Valid values are:

- TIMER\_A\_CAPTURE\_ASYNCHRONOUS [Default]
- TIMER\_A\_CAPTURE\_SYNCHRONOUS

**captureInterruptEnable** is to enable or disable timer captureCompare interrupt. Valid values are:

- TIMER\_A\_CAPTURECOMPARE\_INTERRUPT\_DISABLE [Default]
- TIMER\_A\_CAPTURECOMPARE\_INTERRUPT\_ENABLE

**captureOutputMode** specifies the output mode. Valid values are:

- TIMER\_A\_OUTPUTMODE\_OUTBITVALUE [Default]
- TIMER\_A\_OUTPUTMODE\_SET
- TIMER\_A\_OUTPUTMODE\_TOGGLE\_RESET
- TIMER\_A\_OUTPUTMODE\_SET\_RESET
- TIMER\_A\_OUTPUTMODE\_TOGGLE
- TIMER\_A\_OUTPUTMODE\_RESET
- TIMER\_A\_OUTPUTMODE\_TOGGLE\_SET
- TIMER\_A\_OUTPUTMODE\_RESET\_SET

**Description:**

Modified bits of **TAxCCTLn** register.

**Returns:**

None

### 32.2.2.19 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 captureCompare interrupt. Valid values are:

- **TIMER\_A\_CAPTURECOMPARE\_INTERRUPT\_DISABLE** [Default]
- **TIMER\_A\_CAPTURECOMPARE\_INTERRUPT\_ENABLE**

**compareOutputMode** specifies the output mode. Valid values are:

- **TIMER\_A\_OUTPUTMODE\_OUTBITVALUE** [Default]
- **TIMER\_A\_OUTPUTMODE\_SET**
- **TIMER\_A\_OUTPUTMODE\_TOGGLE\_RESET**
- **TIMER\_A\_OUTPUTMODE\_SET\_RESET**
- **TIMER\_A\_OUTPUTMODE\_TOGGLE**
- **TIMER\_A\_OUTPUTMODE\_RESET**
- **TIMER\_A\_OUTPUTMODE\_TOGGLE\_SET**
- **TIMER\_A\_OUTPUTMODE\_RESET\_SET**

**compareValue** is the count to be compared with in compare mode

#### Description:

Modified bits of **TAxCCRn** register and bits of **TAxCTLn** register.

#### Returns:

None



### 32.2.2.20 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

### 32.2.2.21 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

### 32.2.2.22 TIMER\_A\_startContinuousMode

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_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

### 32.2.2.23 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

### 32.2.2.24 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

### 32.2.2.25 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 captureCompare interrupt. Valid values are:

- **TIMER\_A\_CCIE\_CCR0\_INTERRUPT\_ENABLE**
- **TIMER\_A\_CCIE\_CCR0\_INTERRUPT\_DISABLE** [Default]

**timerClear** decides if timer clock divider, count direction, count need to be reset. Valid values are:

- **TIMER\_A\_DO\_CLEAR**
- **TIMER\_A\_SKIP\_CLEAR** [Default]

#### Description:

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register and bits of **TAxCCR0** register.

#### Returns:

None

### 32.2.2.26 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 captureCompare interrupt. Valid values are:

- **TIMER\_A\_CCIE\_CCR0\_INTERRUPT\_ENABLE**
- **TIMER\_A\_CCIE\_CCR0\_INTERRUPT\_DISABLE** [Default]

**timerClear** decides if timer clock divider, count direction, count need to be reset. Valid values are:

- **TIMER\_A\_DO\_CLEAR**
- **TIMER\_A\_SKIP\_CLEAR** [Default]

**Description:**

Modified bits of **TAxCTL** register, bits of **TAxCTL0** register and bits of **TAxCCR0** register.

**Returns:**

None

### 32.2.2.27 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

## 32.3 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();  
}
```

## 33 TIMER\_B

Introduction .....	382
API Functions .....	383
Programming Example .....	410

### 33.1 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

## 33.2 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\\_startContinuousMode](#) (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)

### 33.2.1 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\(\)](#)

## 33.2.2 Function Documentation

### 33.2.2.1 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

### 33.2.2.2 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

### 33.2.2.3 `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

### 33.2.2.4 `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

### 33.2.2.5 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

### 33.2.2.6 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

### 33.2.2.7 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

### 33.2.2.8 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

### 33.2.2.9 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

### 33.2.2.10 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

### 33.2.2.11 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

### 33.2.2.12 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

### 33.2.2.13 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

### 33.2.2.14 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

### 33.2.2.15 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



### 33.2.2.16 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**

### 33.2.2.17 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**

33.2.2.18 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\_CC1xA** [Default]
- **TIMER\_B\_CAPTURE\_INPUTSELECT\_CC1xB**
- **TIMER\_B\_CAPTURE\_INPUTSELECT\_GND**
- **TIMER\_B\_CAPTURE\_INPUTSELECT\_Vcc**

**synchronizeCaptureSource** decides if capture source should be synchronized with TIMER\_B clock Valid values are:

- **TIMER\_B\_CAPTURE\_ASYNCHRONOUS** [Default]
- **TIMER\_B\_CAPTURE\_SYNCHRONOUS**

**captureInterruptEnable** is to enable or disable TIMER\_B capture compare interrupt. Valid values are:

- **TIMER\_B\_CAPTURECOMPARE\_INTERRUPT\_DISABLE** [Default]
- **TIMER\_B\_CAPTURECOMPARE\_INTERRUPT\_ENABLE**

**captureOutputMode** specifies the output mode. Valid values are:

- **TIMER\_B\_OUTPUTMODE\_OUTBITVALUE** [Default]
- **TIMER\_B\_OUTPUTMODE\_SET**
- **TIMER\_B\_OUTPUTMODE\_TOGGLE\_RESET**
- **TIMER\_B\_OUTPUTMODE\_SET\_RESET**
- **TIMER\_B\_OUTPUTMODE\_TOGGLE**
- **TIMER\_B\_OUTPUTMODE\_RESET**
- **TIMER\_B\_OUTPUTMODE\_TOGGLE\_SET**
- **TIMER\_B\_OUTPUTMODE\_RESET\_SET**

**Description:**

Modified bits of **TBxCCTLn** register.

**Returns:**

None

### 33.2.2.19 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

### 33.2.2.20 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

### 33.2.2.21 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

### 33.2.2.22 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

### 33.2.2.23 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

### 33.2.2.24 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

### 33.2.2.25 TIMER\_B\_startContinuousMode

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_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

### 33.2.2.26 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

### 33.2.2.27 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

### 33.2.2.28 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

### 33.2.2.29 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

### 33.2.2.30 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

## 33.3 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
    );
}
```

## 34 TIMER\_D

Introduction .....	411
API Functions .....	412
Programming Example .....	436

### 34.1 Introduction

Timer\_D is a 16-bit timer/counter with multiple capture/compare registers. Timer\_D can support multiple capture/compares, interval timing, and PWM outputs both in general and high resolution modes. Timer\_D also has extensive interrupt capabilities. Interrupts may be generated from the counter on overflow conditions, from each of the capture/compare registers.

This peripheral API handles Timer D hardware peripheral.

TIMER\_D features include:

- Asynchronous 16-bit timer/counter with four operating modes and four selectable lengths
- Selectable and configurable clock source
- Configurable capture/compare registers
- Controlling rising and falling PWM edges by combining two neighbor TDCCR registers in one compare channel output
- Configurable outputs with PWM capability
- High-resolution mode with a fine clock frequency up to 16 times the timer input clock frequency
- Double-buffered compare registers with synchronized loading
- Interrupt vector register for fast decoding of all Timer\_D interrupts

Differences From Timer\_B Timer\_D is identical to Timer\_B with the following exceptions:

- Timer\_D supports high-resolution mode.
- Timer\_D supports the combination of two adjacent TDCCR<sub>x</sub> registers in one capture/compare channel.
- Timer\_D supports the dual capture event mode.
- Timer\_D supports external fault input, external clear input, and signal. See the TEC chapter for detailed information.
- Timer\_D can synchronize with a second timer instance when available. See the TEC chapter for detailed information.

TIMER\_D can operate in 3 modes

- Continuous Mode
- Up Mode
- Down Mode

TIMER\_D Interrupts may be generated on counter overflow conditions and during capture compare events.

The TIMER\_D may also be used to generate PWM outputs. PWM outputs can be generated by initializing the compare mode with [TIMER\\_D\\_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\\_D\\_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\\_D\\_generatePWM\(\)](#) or a combination of [TIMER\\_D\\_initCompare\(\)](#) and timer start APIs

The TimerD API provides a set of functions for dealing with the TimerD 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 `timerd.c`, with `timerd.h` containing the API definitions for use by applications.

## 34.2 API Functions

### Functions

- void [TIMER\\_D\\_clear](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_clearCaptureCompareInterruptFlag](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister)
- void [TIMER\\_D\\_clearHighResInterruptStatus](#) (uint32\_t baseAddress, uint16\_t mask)
- void [TIMER\\_D\\_clearTimerInterruptFlag](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_combineTDCCRToGeneratePWM](#) (uint32\_t baseAddress, uint16\_t clockSource, uint16\_t clockSourceDivider, uint16\_t clockingMode, uint16\_t timerPeriod, uint16\_t combineCCRRegistersCombination, uint16\_t compareOutputMode, uint16\_t dutyCycle1, uint16\_t dutyCycle2)
- void [TIMER\\_D\\_configureContinuousMode](#) (uint32\_t baseAddress, uint16\_t clockSource, uint16\_t clockSourceDivider, uint16\_t clockingMode, uint16\_t timerInterruptEnable\_TDIE, uint16\_t timerClear)
- uint8\_t [TIMER\\_D\\_configureHighResGeneratorInFreeRunningMode](#) (uint32\_t baseAddress, uint8\_t desiredHighResFrequency)
- void [TIMER\\_D\\_configureHighResGeneratorInRegulatedMode](#) (uint32\_t baseAddress, uint16\_t clockSource, uint16\_t clockSourceDivider, uint16\_t clockingMode, uint8\_t highResClockMultiplyFactor, uint8\_t highResClockDivider)
- void [TIMER\\_D\\_configureUpDownMode](#) (uint32\_t baseAddress, uint16\_t clockSource, uint16\_t clockSourceDivider, uint16\_t clockingMode, uint16\_t timerPeriod, uint16\_t timerInterruptEnable\_TDIE, uint16\_t captureCompareInterruptEnable\_CCR0\_CCIE, uint16\_t timerClear)
- void [TIMER\\_D\\_configureUpMode](#) (uint32\_t baseAddress, uint16\_t clockSource, uint16\_t clockSourceDivider, uint16\_t clockingMode, uint16\_t timerPeriod, uint16\_t timerInterruptEnable\_TDIE, uint16\_t captureCompareInterruptEnable\_CCR0\_CCIE, uint16\_t timerClear)
- void [TIMER\\_D\\_disableCaptureCompareInterrupt](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister)
- void [TIMER\\_D\\_disableHighResClockEnhancedAccuracy](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_disableHighResFastWakeup](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_DisableHighResGeneratorForceON](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_disableHighResInterrupt](#) (uint32\_t baseAddress, uint16\_t mask)
- void [TIMER\\_D\\_disableTimerInterrupt](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_enableCaptureCompareInterrupt](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister)
- void [TIMER\\_D\\_enableHighResClockEnhancedAccuracy](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_enableHighResFastWakeup](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_EnableHighResGeneratorForceON](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_enableHighResInterrupt](#) (uint32\_t baseAddress, uint16\_t mask)
- void [TIMER\\_D\\_enableTimerInterrupt](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_generatePWM](#) (uint32\_t baseAddress, uint16\_t clockSource, uint16\_t clockSourceDivider, uint16\_t clockingMode, uint16\_t timerPeriod, uint16\_t compareRegister, uint16\_t compareOutputMode, uint16\_t dutyCycle)
- uint16\_t [TIMER\\_D\\_getCaptureCompareCount](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister)
- uint8\_t [TIMER\\_D\\_getCaptureCompareInputSignal](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister)
- uint32\_t [TIMER\\_D\\_getCaptureCompareInterruptStatus](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister, uint16\_t mask)
- uint16\_t [TIMER\\_D\\_getCaptureCompareLatchCount](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister)
- uint16\_t [TIMER\\_D\\_getCounterValue](#) (uint32\_t baseAddress)
- uint16\_t [TIMER\\_D\\_getHighResInterruptStatus](#) (uint32\_t baseAddress, uint16\_t mask)
- uint8\_t [TIMER\\_D\\_getOutputForOutputModeOutBitValue](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister)
- uint8\_t [TIMER\\_D\\_getSynchronizedCaptureCompareInput](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister, uint16\_t synchronized)
- uint32\_t [TIMER\\_D\\_getTimerInterruptStatus](#) (uint32\_t baseAddress)
- void [TIMER\\_D\\_initCapture](#) (uint32\_t baseAddress, uint16\_t captureRegister, uint16\_t captureMode, uint16\_t captureInputSelect, uint16\_t synchronizeCaptureSource, uint16\_t captureInterruptEnable, uint16\_t captureOutputMode, uint8\_t channelCaptureMode)
- void [TIMER\\_D\\_initCompare](#) (uint32\_t baseAddress, uint16\_t compareRegister, uint16\_t compareInterruptEnable, uint16\_t compareOutputMode, uint16\_t compareValue)
- void [TIMER\\_D\\_initCompareLatchLoadEvent](#) (uint16\_t baseAddress, uint16\_t compareRegister, uint16\_t compareLatchLoadEvent)
- void [TIMER\\_D\\_selectCounterLength](#) (uint16\_t baseAddress, uint16\_t counterLength)
- void [TIMER\\_D\\_selectHighResClockRange](#) (uint32\_t baseAddress, uint16\_t highResClockRange)



- void [TIMER\\_D\\_selectHighResCoarseClockRange](#) (uint32\_t baseAddress, uint16\_t highResCoarseClockRange)
- void [TIMER\\_D\\_selectLatchingGroup](#) (uint16\_t baseAddress, uint16\_t groupLatch)
- void [TIMER\\_D\\_setCompareValue](#) (uint32\_t baseAddress, uint16\_t compareRegister, uint16\_t compareValue)
- void [TIMER\\_D\\_setOutputForOutputModeOutBitValue](#) (uint32\_t baseAddress, uint16\_t captureCompareRegister, uint8\_t outputModeOutBitValue)
- void [TIMER\\_D\\_startCounter](#) (uint32\_t baseAddress, uint16\_t timerMode)
- void [TIMER\\_D\\_stop](#) (uint32\_t baseAddress)

### 34.2.1 Detailed Description

The TIMER\_D 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.

TimerD configuration and initialization is handled by

- [TIMER\\_D\\_startCounter\(\)](#),
- [TIMER\\_D\\_configureContinuousMode\(\)](#),
- [TIMER\\_D\\_configureUpMode\(\)](#),
- [TIMER\\_D\\_configureUpDownMode\(\)](#),
- [TIMER\\_D\\_startContinuousMode\(\)](#),
- [TIMER\\_D\\_startUpMode\(\)](#),
- [TIMER\\_D\\_startUpDownMode\(\)](#),
- [TIMER\\_D\\_initCapture\(\)](#),
- [TIMER\\_D\\_initCompare\(\)](#),
- [TIMER\\_D\\_clear\(\)](#),
- [TIMER\\_D\\_stop\(\)](#),
- [TIMER\\_D\\_configureHighResGeneratorInFreeRunningMode\(\)](#),
- [TIMER\\_D\\_configureHighResGeneratorInRegulatedMode\(\)](#),
- [TIMER\\_D\\_combineTDCCRTToGeneratePWM\(\)](#),
- [TIMER\\_D\\_selectLatchingGroup\(\)](#),
- [TIMER\\_D\\_selectCounterLength\(\)](#),
- [TIMER\\_D\\_initCompareLatchLoadEvent\(\)](#),
- [TIMER\\_D\\_disableHighResFastWakeup\(\)](#),
- [TIMER\\_D\\_enableHighResFastWakeup\(\)](#),
- [TIMER\\_D\\_disableHighResClockEnhancedAccuracy\(\)](#),
- [TIMER\\_D\\_enableHighResClockEnhancedAccuracy\(\)](#),
- [TIMER\\_D\\_DisableHighResGeneratorForceON\(\)](#),
- [TIMER\\_D\\_EnableHighResGeneratorForceON\(\)](#),
- [TIMER\\_D\\_selectHighResCoarseClockRange\(\)](#),
- [TIMER\\_D\\_selectHighResClockRange\(\)](#)

TimerD outputs are handled by

- [TIMER\\_D\\_getSynchronizedCaptureCompareInput\(\)](#),
- [TIMER\\_D\\_getOutputForOutputModeOutBitValue\(\)](#),
- [TIMER\\_D\\_setOutputForOutputModeOutBitValue\(\)](#),
- [TIMER\\_D\\_generatePWM\(\)](#),
- [TIMER\\_D\\_getCaptureCompareCount\(\)](#),
- [TIMER\\_D\\_setCompareValue\(\)](#),
- [TIMER\\_D\\_getCaptureCompareLatchCount\(\)](#),
- [TIMER\\_D\\_getCaptureCompareInputSignal\(\)](#),

- `TIMER_D_getCounterValue()`

The interrupt handler for the TimerD interrupt is managed with

- `TIMER_D_enableTimerInterrupt()`,
- `TIMER_D_disableTimerInterrupt()`,
- `TIMER_D_getTimerInterruptStatus()`,
- `TIMER_D_enableCaptureCompareInterrupt()`,
- `TIMER_D_disableCaptureCompareInterrupt()`,
- `TIMER_D_getCaptureCompareInterruptStatus()`,
- `TIMER_D_clearCaptureCompareInterruptFlag()`
- `TIMER_D_clearTimerInterruptFlag()`,
- `TIMER_D_enableHighResInterrupt()`,
- `TIMER_D_disableTimerInterrupt()`,
- `TIMER_D_getHighResInterruptStatus()`,
- `TIMER_D_clearHighResInterruptStatus()`

Timer\_D High Resolution handling APIs

- `TIMER_D_getHighResInterruptStatus()`,
- `TIMER_D_clearHighResInterruptStatus()`,
- `TIMER_D_disableHighResFastWakeup()`,
- `TIMER_D_enableHighResFastWakeup()`,
- `TIMER_D_disableHighResClockEnhancedAccuracy()`,
- `TIMER_D_enableHighResClockEnhancedAccuracy()`,
- `TIMER_D_DisableHighResGeneratorForceON()`,
- `TIMER_D_EnableHighResGeneratorForceON()`,
- `TIMER_D_selectHighResCoarseClockRange()`,
- `TIMER_D_selectHighResClockRange()`,
- `TIMER_D_configureHighResGeneratorInFreeRunningMode()`,
- `TIMER_D_configureHighResGeneratorInRegulatedMode()`

## 34.2.2 Function Documentation

### 34.2.2.1 `TIMER_D_clear`

Reset/Clear the timer clock divider, count direction, count.

**Prototype:**

```
void
TIMER_D_clear(uint32_t baseAddress)
```

**Parameters:**

***baseAddress*** is the base address of the `TIMER_D` module.

**Description:**

Modified bits of **TDxCTL0** register.

**Returns:**

None

### 34.2.2.2 TIMER\_D\_clearCaptureCompareInterruptFlag

Clears the capture-compare interrupt flag.

**Prototype:**

```
void
TIMER_D_clearCaptureCompareInterruptFlag(uint32_t baseAddress,
                                         uint16_t captureCompareRegister)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** selects the Capture-compare register being used. Valid values are:

- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6

**Description:**

Modified bits are CCIFG of TDxCCTLn register.

**Returns:**

None

### 34.2.2.3 TIMER\_D\_clearHighResInterruptStatus

Clears High Resolution interrupt status.

**Prototype:**

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

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**mask** is the mask for the interrupts to clear Mask value is the logical OR of any of the following:

- TIMER\_D\_HIGH\_RES\_FREQUENCY\_UNLOCK
- TIMER\_D\_HIGH\_RES\_FREQUENCY\_LOCK
- TIMER\_D\_HIGH\_RES\_FAIL\_HIGH
- TIMER\_D\_HIGH\_RES\_FAIL\_LOW

**Description:**

Modified bits of TDxHINT register.

**Returns:**

None

### 34.2.2.4 TIMER\_D\_clearTimerInterruptFlag

Clears the Timer TDIFG interrupt flag.

**Prototype:**

```
void
TIMER_D_clearTimerInterruptFlag(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**Description:**

Modified bits are **TDIFG** of **TDxCTL0** register.

**Returns:**

None

### 34.2.2.5 TIMER\_D\_combineTDCCRToGeneratePWM

Combine TDCCR to get PWM.

**Prototype:**

```
void
TIMER_D_combineTDCCRToGeneratePWM(uint32_t baseAddress,
                                   uint16_t clockSource,
                                   uint16_t clockSourceDivider,
                                   uint16_t clockingMode,
                                   uint16_t timerPeriod,
                                   uint16_t combineCCRRegistersCombination,
                                   uint16_t compareOutputMode,
                                   uint16_t dutyCycle1,
                                   uint16_t dutyCycle2)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**clockSource** selects Clock source. Valid values are:

- **TIMER\_D\_CLOCKSOURCE\_EXTERNAL\_TDCLK** [Default]
- **TIMER\_D\_CLOCKSOURCE\_ACLK**
- **TIMER\_D\_CLOCKSOURCE\_SMCLK**
- **TIMER\_D\_CLOCKSOURCE\_INVERTED\_EXTERNAL\_TDCLK**

**clockSourceDivider** is the divider for clock source. Valid values are:

- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_1** [Default]
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_2**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_4**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_8**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_3**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_5**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_6**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_7**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_10**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_12**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_14**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_16**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_20**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_24**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_28**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_32**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_40**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_48**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_56**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_64**

**clockingMode** is the selected clock mode register values. Valid values are:

- **TIMER\_D\_CLOCKINGMODE\_EXTERNAL\_CLOCK** [Default]
- **TIMER\_D\_CLOCKINGMODE\_HIRES\_LOCAL\_CLOCK**
- **TIMER\_D\_CLOCKINGMODE\_AUXILIARY\_CLK**

**timerPeriod** is the specified timer period

**combineCCRRegistersCombination** selects desired CCR registers to combine Valid values are:

- **TIMER\_D\_COMBINE\_CCR1\_CCR2**
- **TIMER\_D\_COMBINE\_CCR3\_CCR4** - (available on TIMER\_D5, TIMER\_D7)
- **TIMER\_D\_COMBINE\_CCR5\_CCR6** - (available only on TIMER\_D7)

**compareOutputMode** specifies the output mode. Valid values are:

- **TIMER\_D\_OUTPUTMODE\_OUTBITVALUE** [Default]
- **TIMER\_D\_OUTPUTMODE\_SET**
- **TIMER\_D\_OUTPUTMODE\_TOGGLE\_RESET**
- **TIMER\_D\_OUTPUTMODE\_SET\_RESET**
- **TIMER\_D\_OUTPUTMODE\_TOGGLE**
- **TIMER\_D\_OUTPUTMODE\_RESET**
- **TIMER\_D\_OUTPUTMODE\_TOGGLE\_SET**
- **TIMER\_D\_OUTPUTMODE\_RESET\_SET**

**dutyCycle1** specifies the dutycycle for the generated waveform

**dutyCycle2** specifies the dutycycle for the generated waveform

**Description:**

Modified bits of **TDxCCTLn** register, bits of **TDxCCR0** register, bits of **TDxCCTL0** register, bits of **TDxCTL0** register and bits of **TDxCTL1** register.

**Returns:**

None

### 34.2.2.6 TIMER\_D\_configureContinuousMode

Configures timer in continuous mode.

**Prototype:**

```
void
TIMER_D_configureContinuousMode(uint32_t baseAddress,
                                uint16_t clockSource,
                                uint16_t clockSourceDivider,
                                uint16_t clockingMode,
                                uint16_t timerInterruptEnable_TDIE,
                                uint16_t timerClear)
```

**Description:**

This API does not start the timer. Timer needs to be started when required using the **TIMER\_D\_start** API.

**Parameters:**

**baseAddress** is the base address of the **TIMER\_D** module.

**clockSource** selects Clock source. Valid values are:

- **TIMER\_D\_CLOCKSOURCE\_EXTERNAL\_TDCLK** [Default]
- **TIMER\_D\_CLOCKSOURCE\_ACLK**
- **TIMER\_D\_CLOCKSOURCE\_SMCLK**
- **TIMER\_D\_CLOCKSOURCE\_INVERTED\_EXTERNAL\_TDCLK**

**clockSourceDivider** is the divider for clock source. Valid values are:

- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_1** [Default]
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_2**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_4**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_8**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_3**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_5**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_6**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_7**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_10**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_12**

- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_14**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_16**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_20**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_24**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_28**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_32**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_40**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_48**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_56**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_64**

**clockingMode** is the selected clock mode register values. Valid values are:

- **TIMER\_D\_CLOCKINGMODE\_EXTERNAL\_CLOCK** [Default]
- **TIMER\_D\_CLOCKINGMODE\_HIRES\_LOCAL\_CLOCK**
- **TIMER\_D\_CLOCKINGMODE\_AUXILIARY\_CLK**

**timerInterruptEnable\_TDIE** is to enable or disable timer interrupt Valid values are:

- **TIMER\_D\_TDIE\_INTERRUPT\_ENABLE**
- **TIMER\_D\_TDIE\_INTERRUPT\_DISABLE** [Default]

**timerClear** decides if timer clock divider, count direction, count need to be reset. Valid values are:

- **TIMER\_D\_DO\_CLEAR**
- **TIMER\_D\_SKIP\_CLEAR** [Default]

Modified bits of **TDxCTL0** register and bits of **TDxCTL1** register.

**Returns:**

None

### 34.2.2.7 TIMER\_D\_configureHighResGeneratorInFreeRunningMode

Configures TIMER\_D in free running mode.

**Prototype:**

```
uint8_t
TIMER_D_configureHighResGeneratorInFreeRunningMode(uint32_t baseAddress,
                                                    uint8_t desiredHighResFrequency)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**desiredHighResFrequency** selects the desired High Resolution frequency used. Valid values are:

- **TIMER\_D\_HIGHRES\_64MHZ**
- **TIMER\_D\_HIGHRES\_128MHZ**
- **TIMER\_D\_HIGHRES\_200MHZ**
- **TIMER\_D\_HIGHRES\_256MHZ**

**Description:**

Modified bits of **TDxHCTL1** register, bits of **TDxHCTL0** register and bits of **TDxCTL1** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL

### 34.2.2.8 TIMER\_D\_configureHighResGeneratorInRegulatedMode

Configures TIMER\_D in Regulated mode.

**Prototype:**

```
void
TIMER_D_configureHighResGeneratorInRegulatedMode(uint32_t baseAddress,
                                                  uint16_t clockSource,
                                                  uint16_t clockSourceDivider,
                                                  uint16_t clockingMode,
                                                  uint8_t highResClockMultiplyFactor,
                                                  uint8_t highResClockDivider)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**clockSource** selects Clock source. Valid values are:

- TIMER\_D\_CLOCKSOURCE\_EXTERNAL\_TDCLK [Default]
- TIMER\_D\_CLOCKSOURCE\_ACLK
- TIMER\_D\_CLOCKSOURCE\_SMCLK
- TIMER\_D\_CLOCKSOURCE\_INVERTED\_EXTERNAL\_TDCLK

**clockSourceDivider** is the divider for clock source. Valid values are:

- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_1 [Default]
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_2
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_4
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_8
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_3
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_5
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_6
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_7
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_10
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_12
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_14
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_16
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_20
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_24
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_28
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_32
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_40
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_48
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_56
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_64

**clockingMode** is the selected clock mode register values. Valid values are:

- TIMER\_D\_CLOCKINGMODE\_EXTERNAL\_CLOCK [Default]
- TIMER\_D\_CLOCKINGMODE\_HIRES\_LOCAL\_CLOCK
- TIMER\_D\_CLOCKINGMODE\_AUXILIARY\_CLK

**highResClockMultiplyFactor** selects the high resolution multiply factor. Valid values are:

- TIMER\_D\_HIGHRES\_CLK\_MULTIPLY\_FACTOR\_8x
- TIMER\_D\_HIGHRES\_CLK\_MULTIPLY\_FACTOR\_16x

**highResClockDivider** selects the high resolution divider. Valid values are:

- TIMER\_D\_HIGHRES\_CLK\_DIVIDER\_1
- TIMER\_D\_HIGHRES\_CLK\_DIVIDER\_2
- TIMER\_D\_HIGHRES\_CLK\_DIVIDER\_4
- TIMER\_D\_HIGHRES\_CLK\_DIVIDER\_8

**Description:**

Modified bits of **TDxHCTL0** register, bits of **TDxCTL0** register and bits of **TDxCTL1** register.

**Returns:**

None

### 34.2.2.9 TIMER\_D\_configureUpDownMode

Configures timer in up down mode.

**Prototype:**

```
void
TIMER_D_configureUpDownMode (uint32_t baseAddress,
                             uint16_t clockSource,
                             uint16_t clockSourceDivider,
                             uint16_t clockingMode,
                             uint16_t timerPeriod,
                             uint16_t timerInterruptEnable_TDIE,
                             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\_D\_start API.

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**clockSource** selects Clock source. Valid values are:

- TIMER\_D\_CLOCKSOURCE\_EXTERNAL\_TDCLK [Default]
- TIMER\_D\_CLOCKSOURCE\_ACLK
- TIMER\_D\_CLOCKSOURCE\_SMCLK
- TIMER\_D\_CLOCKSOURCE\_INVERTED\_EXTERNAL\_TDCLK

**clockSourceDivider** is the divider for clock source. Valid values are:

- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_1 [Default]
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_2
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_4
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_8
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_3
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_5
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_6
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_7
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_10
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_12
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_14
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_16
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_20
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_24
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_28
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_32
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_40
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_48
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_56
- TIMER\_D\_CLOCKSOURCE\_DIVIDER\_64

**clockingMode** is the selected clock mode register values. Valid values are:

- TIMER\_D\_CLOCKINGMODE\_EXTERNAL\_CLOCK [Default]
- TIMER\_D\_CLOCKINGMODE\_HIRES\_LOCAL\_CLOCK
- TIMER\_D\_CLOCKINGMODE\_AUXILIARY\_CLK

**timerPeriod** is the specified timer period

**timerInterruptEnable\_TDIE** is to enable or disable timer interrupt Valid values are:

- TIMER\_D\_TDIE\_INTERRUPT\_ENABLE
- TIMER\_D\_TDIE\_INTERRUPT\_DISABLE [Default]

**captureCompareInterruptEnable\_CCR0\_CCIE** is to enable or disable timer CCR0 captureCompare interrupt. Valid values are:

- TIMER\_D\_CCIE\_CCR0\_INTERRUPT\_ENABLE
- TIMER\_D\_CCIE\_CCR0\_INTERRUPT\_DISABLE [Default]



**timerClear** decides if timer clock divider, count direction, count need to be reset. Valid values are:

- **TIMER\_D\_DO\_CLEAR**
- **TIMER\_D\_SKIP\_CLEAR** [Default]

Modified bits of **TDxCCR0** register, bits of **TDxCCTL0** register, bits of **TDxCTL0** register and bits of **TDxCTL1** register.

**Returns:**  
None

### 34.2.2.10 TIMER\_D\_configureUpMode

Configures timer in up mode.

**Prototype:**

```
void
TIMER_D_configureUpMode(uint32_t baseAddress,
                        uint16_t clockSource,
                        uint16_t clockSourceDivider,
                        uint16_t clockingMode,
                        uint16_t timerPeriod,
                        uint16_t timerInterruptEnable_TDIE,
                        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\_D\_start** API.

**Parameters:**

**baseAddress** is the base address of the **TIMER\_D** module.

**clockSource** selects Clock source. Valid values are:

- **TIMER\_D\_CLOCKSOURCE\_EXTERNAL\_TDCLK** [Default]
- **TIMER\_D\_CLOCKSOURCE\_ACLK**
- **TIMER\_D\_CLOCKSOURCE\_SMCLK**
- **TIMER\_D\_CLOCKSOURCE\_INVERTED\_EXTERNAL\_TDCLK**

**clockSourceDivider** is the divider for clock source. Valid values are:

- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_1** [Default]
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_2**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_4**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_8**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_3**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_5**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_6**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_7**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_10**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_12**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_14**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_16**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_20**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_24**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_28**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_32**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_40**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_48**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_56**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_64**

**clockingMode** is the selected clock mode register values. Valid values are:

- **TIMER\_D\_CLOCKINGMODE\_EXTERNAL\_CLOCK** [Default]

- **TIMER\_D\_CLOCKINGMODE\_HIRES\_LOCAL\_CLOCK**
- **TIMER\_D\_CLOCKINGMODE\_AUXILIARY\_CLK**

**timerPeriod** is the specified timer period. This is the value that gets written into the CCR0. Limited to 16 bits [uint16\_t]

**timerInterruptEnable\_TDIE** is to enable or disable timer interrupt Valid values are:

- **TIMER\_D\_TDIE\_INTERRUPT\_ENABLE**
- **TIMER\_D\_TDIE\_INTERRUPT\_DISABLE** [Default]

**captureCompareInterruptEnable\_CCR0\_CCIE** is to enable or disable timer CCR0 captureCompare interrupt. Valid values are:

- **TIMER\_D\_CCIE\_CCR0\_INTERRUPT\_ENABLE**
- **TIMER\_D\_CCIE\_CCR0\_INTERRUPT\_DISABLE** [Default]

**timerClear** decides if timer clock divider, count direction, count need to be reset. Valid values are:

- **TIMER\_D\_DO\_CLEAR**
- **TIMER\_D\_SKIP\_CLEAR** [Default]

Modified bits of **TDxCCR0** register, bits of **TDxCCTL0** register, bits of **TDxCTL0** register and bits of **TDxCTL1** register.

**Returns:**

None

### 34.2.2.11 TIMER\_D\_disableCaptureCompareInterrupt

Disable capture compare interrupt.

**Prototype:**

```
void
TIMER_D_disableCaptureCompareInterrupt (uint32_t baseAddress,
                                         uint16_t captureCompareRegister)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** is the selected capture compare register Valid values are:

- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6**

**Description:**

Modified bits of **TDxCCTLn** register.

**Returns:**

None

### 34.2.2.12 TIMER\_D\_disableHighResClockEnhancedAccuracy

Disable High Resolution Clock Enhanced Accuracy.

**Prototype:**

```
void
TIMER_D_disableHighResClockEnhancedAccuracy (uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**Description:**

Modified bits are **TDHEAEN** of **TDxHCTL0** register.

**Returns:**

None

### 34.2.2.13 TIMER\_D\_disableHighResFastWakeup

Disable High Resolution fast wakeup.

**Prototype:**

```
void
TIMER_D_disableHighResFastWakeup(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**Description:**

Modified bits are **TDHFW** of **TDxHCTL0** register.

**Returns:**

None

### 34.2.2.14 TIMER\_D\_DisableHighResGeneratorForceON

Disable High Resolution Clock Enhanced Accuracy.

**Prototype:**

```
void
TIMER_D_DisableHighResGeneratorForceON(uint32_t baseAddress)
```

**Description:**

High-resolution generator is on if the TIMER\_D counter

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

Modified bits are **TDHRON** of **TDxHCTL0** register.

**Returns:**

None

### 34.2.2.15 TIMER\_D\_disableHighResInterrupt

Disable High Resolution interrupt.

**Prototype:**

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

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**mask** is the mask of interrupts to disable Mask value is the logical OR of any of the following:

- **TIMER\_D\_HIGH\_RES\_FREQUENCY\_UNLOCK**
- **TIMER\_D\_HIGH\_RES\_FREQUENCY\_LOCK**
- **TIMER\_D\_HIGH\_RES\_FAIL\_HIGH**

#### ■ TIMER\_D\_HIGH\_RES\_FAIL\_LOW

**Description:**

Modified bits of **TDxHINT** register.

**Returns:**

None

### 34.2.2.16 TIMER\_D\_disableTimerInterrupt

Disable timer interrupt.

**Prototype:**

```
void
TIMER_D_disableTimerInterrupt (uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**Description:**

Modified bits of **TDxCTL0** register.

**Returns:**

None

### 34.2.2.17 TIMER\_D\_enableCaptureCompareInterrupt

Enable capture compare interrupt.

**Prototype:**

```
void
TIMER_D_enableCaptureCompareInterrupt (uint32_t baseAddress,
                                       uint16_t captureCompareRegister)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** is the selected capture compare register Valid values are:

- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6**

**Description:**

Modified bits of **TDxCCTLn** register.

**Returns:**

None

### 34.2.2.18 TIMER\_D\_enableHighResClockEnhancedAccuracy

Enable High Resolution Clock Enhanced Accuracy.

**Prototype:**

```
void  
TIMER_D_enableHighResClockEnhancedAccuracy(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**Description:**

Modified bits are **TDHEAEN** of **TDxHCTL0** register.

**Returns:**

None

### 34.2.2.19 TIMER\_D\_enableHighResFastWakeup

Enable High Resolution fast wakeup.

**Prototype:**

```
void  
TIMER_D_enableHighResFastWakeup(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**Description:**

Modified bits are **TDHFW** of **TDxHCTL0** register.

**Returns:**

None

### 34.2.2.20 TIMER\_D\_EnableHighResGeneratorForceON

Enable High Resolution Clock Enhanced Accuracy.

**Prototype:**

```
void  
TIMER_D_EnableHighResGeneratorForceON(uint32_t baseAddress)
```

**Description:**

High-resolution generator is on in all TIMER\_D MCx modes. The PMM remains in high-current mode.

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

Modified bits are **TDHRON** of **TDxHCTL0** register.

**Returns:**

None

### 34.2.2.21 TIMER\_D\_enableHighResInterrupt

Enable High Resolution interrupt.

**Prototype:**

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

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**mask** is the mask of interrupts to enable Mask value is the logical OR of any of the following:

- **TIMER\_D\_HIGH\_RES\_FREQUENCY\_UNLOCK**
- **TIMER\_D\_HIGH\_RES\_FREQUENCY\_LOCK**
- **TIMER\_D\_HIGH\_RES\_FAIL\_HIGH**
- **TIMER\_D\_HIGH\_RES\_FAIL\_LOW**

**Description:**

Modified bits of **TDxHINT** register.

**Returns:**

None

### 34.2.2.22 TIMER\_D\_enableTimerInterrupt

Enable timer interrupt.

**Prototype:**

```
void
TIMER_D_enableTimerInterrupt (uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**Description:**

Modified bits of **TDxCTL0** register.

**Returns:**

None

### 34.2.2.23 TIMER\_D\_generatePWM

Generate a PWM with timer running in up mode.

**Prototype:**

```
void
TIMER_D_generatePWM (uint32_t baseAddress,
                     uint16_t clockSource,
                     uint16_t clockSourceDivider,
                     uint16_t clockingMode,
                     uint16_t timerPeriod,
                     uint16_t compareRegister,
                     uint16_t compareOutputMode,
                     uint16_t dutyCycle)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**clockSource** selects Clock source. Valid values are:

- **TIMER\_D\_CLOCKSOURCE\_EXTERNAL\_TDCLK** [Default]

- **TIMER\_D\_CLOCKSOURCE\_ACLK**
- **TIMER\_D\_CLOCKSOURCE\_SMCLK**
- **TIMER\_D\_CLOCKSOURCE\_INVERTED\_EXTERNAL\_TDCLK**

**clockSourceDivider** is the divider for clock source. Valid values are:

- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_1** [Default]
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_2**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_4**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_8**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_3**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_5**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_6**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_7**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_10**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_12**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_14**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_16**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_20**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_24**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_28**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_32**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_40**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_48**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_56**
- **TIMER\_D\_CLOCKSOURCE\_DIVIDER\_64**

**clockingMode** is the selected clock mode register values. Valid values are:

- **TIMER\_D\_CLOCKINGMODE\_EXTERNAL\_CLOCK** [Default]
- **TIMER\_D\_CLOCKINGMODE\_HIRES\_LOCAL\_CLOCK**
- **TIMER\_D\_CLOCKINGMODE\_AUXILIARY\_CLK**

**timerPeriod** is the specified timer period

**compareRegister** selects the compare register being used. Valid values are:

- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6**

**compareOutputMode** specifies the output mode. Valid values are:

- **TIMER\_D\_OUTPUTMODE\_OUTBITVALUE** [Default]
- **TIMER\_D\_OUTPUTMODE\_SET**
- **TIMER\_D\_OUTPUTMODE\_TOGGLE\_RESET**
- **TIMER\_D\_OUTPUTMODE\_SET\_RESET**
- **TIMER\_D\_OUTPUTMODE\_TOGGLE**
- **TIMER\_D\_OUTPUTMODE\_RESET**
- **TIMER\_D\_OUTPUTMODE\_TOGGLE\_SET**
- **TIMER\_D\_OUTPUTMODE\_RESET\_SET**

**dutyCycle** specifies the dutycycle for the generated waveform

**Description:**

Modified bits of **TDxCCTLn** register, bits of **TDxCCR0** register, bits of **TDxCCTL0** register, bits of **TDxCTL0** register and bits of **TDxCTL1** register.

**Returns:**

None

### 34.2.2.24 TIMER\_D\_getCaptureCompareCount

Get current capturecompare count.

**Prototype:**

```
uint16_t
TIMER_D_getCaptureCompareCount (uint32_t baseAddress,
                                uint16_t captureCompareRegister)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** selects the Capture register being used. Valid values are:

- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6

**Returns:**

current count as uint16\_t

### 34.2.2.25 uint8\_t TIMER\_D\_getCaptureCompareInputSignal (uint32\_t baseAddress, uint16\_t captureCompareRegister)

Get current capturecompare input signal.

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** selects the Capture register being used. Valid values are:

- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6

**Returns:**

One of the following:

- TIMER\_D\_CAPTURECOMPARE\_INPUT
- 0x00  
indicating the current input signal

### 34.2.2.26 uint32\_t TIMER\_D\_getCaptureCompareInterruptStatus (uint32\_t baseAddress, uint16\_t captureCompareRegister, uint16\_t mask)

Return capture compare interrupt status.

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** is the selected capture compare register Valid values are:

- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1



- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6**

**mask** is the mask for the interrupt status Mask value is the logical OR of any of the following:

- **TIMER\_D\_CAPTURE\_OVERFLOW**
- **TIMER\_D\_CAPTURECOMPARE\_INTERRUPT\_FLAG**

**Returns:**

Logical OR of any of the following:

- **TIMER\_D\_CAPTURE\_OVERFLOW**
- **TIMER\_D\_CAPTURECOMPARE\_INTERRUPT\_FLAG**  
indicating the status of the masked flags

### 34.2.2.27 uint16\_t TIMER\_D\_getCaptureCompareLatchCount (uint32\_t *baseAddress*, uint16\_t *captureCompareRegister*)

Get current capture compare latch register count.

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** selects the Capture register being used. Valid values are:

- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6**

**Returns:**

current count as uint16\_t

### 34.2.2.28 uint16\_t TIMER\_D\_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\_D\_THRESHOLD** define in the corresponding header file can be modified so that the votes must be closer together for a consensus to occur.

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**Returns:**

Majority vote of timer count value

### 34.2.2.29 uint16\_t TIMER\_D\_getHighResInterruptStatus (uint32\_t *baseAddress*, uint16\_t *mask*)

Returns High Resolution interrupt status.

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**mask** is the mask for the interrupt status Mask value is the logical OR of any of the following:

- **TIMER\_D\_HIGH\_RES\_FREQUENCY\_UNLOCK**
- **TIMER\_D\_HIGH\_RES\_FREQUENCY\_LOCK**
- **TIMER\_D\_HIGH\_RES\_FAIL\_HIGH**
- **TIMER\_D\_HIGH\_RES\_FAIL\_LOW**

**Description:**

Modified bits of **TDxHINT** register.

**Returns:**

Logical OR of any of the following:

- **TIMER\_D\_HIGH\_RES\_FREQUENCY\_UNLOCK**
  - **TIMER\_D\_HIGH\_RES\_FREQUENCY\_LOCK**
  - **TIMER\_D\_HIGH\_RES\_FAIL\_HIGH**
  - **TIMER\_D\_HIGH\_RES\_FAIL\_LOW**
- indicating the status of the masked interrupts

### 34.2.2.30 TIMER\_D\_getOutputForOutputModeOutBitValue

Get output bit for output mode.

**Prototype:**

```
uint8_t
TIMER_D_getOutputForOutputModeOutBitValue (uint32_t baseAddress,
                                             uint16_t captureCompareRegister)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** selects the Capture register being used. Valid values are:

- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6**

**Returns:**

One of the following:

- **TIMER\_D\_OUTPUTMODE\_OUTBITVALUE\_HIGH**
- **TIMER\_D\_OUTPUTMODE\_OUTBITVALUE\_LOW**

### 34.2.2.31 uint8\_t TIMER\_D\_getSynchronizedCaptureCompareInput (uint32\_t baseAddress, uint16\_t captureCompareRegister, uint16\_t synchronized)

Get synchronized capturecompare input.

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** selects the Capture register being used. Valid values are:

- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2**

- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6

**synchronized** is to select type of capture compare input. Valid values are:

- TIMER\_D\_READ\_SYNCHRONIZED\_CAPTURECOMPAREINPUT
- TIMER\_D\_READ\_CAPTURE\_COMPARE\_INPUT

**Returns:**

One of the following:

- TIMER\_D\_CAPTURECOMPARE\_INPUT\_HIGH
- TIMER\_D\_CAPTURECOMPARE\_INPUT\_LOW

### 34.2.2.32 uint32\_t TIMER\_D\_getTimerInterruptStatus (uint32\_t *baseAddress*)

Get timer interrupt status.

**Parameters:**

***baseAddress*** is the base address of the TIMER\_D module.

**Returns:**

One of the following:

- TIMER\_D\_INTERRUPT\_NOT\_PENDING
  - TIMER\_D\_INTERRUPT\_PENDING
- indicating the timer interrupt status

### 34.2.2.33 void TIMER\_D\_initCapture (uint32\_t *baseAddress*, uint16\_t *captureRegister*, uint16\_t *captureMode*, uint16\_t *captureInputSelect*, uint16\_t *synchronizeCaptureSource*, uint16\_t *captureInterruptEnable*, uint16\_t *captureOutputMode*, uint8\_t *channelCaptureMode*)

Initializes Capture Mode.

**Parameters:**

***baseAddress*** is the base address of the TIMER\_D 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\_D\_CAPTURECOMPARE\_REGISTER\_0
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6

***captureMode*** is the capture mode selected. Valid values are:

- TIMER\_D\_CAPTUREMODE\_NO\_CAPTURE [Default]
- TIMER\_D\_CAPTUREMODE\_RISING\_EDGE
- TIMER\_D\_CAPTUREMODE\_FALLING\_EDGE
- TIMER\_D\_CAPTUREMODE\_RISING\_AND\_FALLING\_EDGE

***captureInputSelect*** decides the Input Select Valid values are:

- TIMER\_D\_CAPTURE\_INPUTSELECT\_CC1xA [Default]
- TIMER\_D\_CAPTURE\_INPUTSELECT\_CC1xB
- TIMER\_D\_CAPTURE\_INPUTSELECT\_GND

- **TIMER\_D\_CAPTURE\_INPUTSELECT\_Vcc**

**synchronizeCaptureSource** decides if capture source should be synchronized with timer clock Valid values are:

- **TIMER\_D\_CAPTURE\_ASYNCHRONOUS** [Default]
- **TIMER\_D\_CAPTURE\_SYNCHRONOUS**

**Description:**

Modified bits of **TDxCCTLn** register and bits of **TDxCTL2** register.

**Returns:**

None

### 34.2.2.34 TIMER\_D\_initCompare

Initializes Compare Mode.

**Prototype:**

```
void
TIMER_D_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\_D** module.

**compareRegister** selects the Capture register being used. Valid values are:

- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5**
- **TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6**

**compareInterruptEnable** is to enable or disable timer captureCompare interrupt. Valid values are:

- **TIMER\_D\_CAPTURECOMPARE\_INTERRUPT\_ENABLE**
- **TIMER\_D\_CAPTURECOMPARE\_INTERRUPT\_DISABLE** [Default]

**compareOutputMode** specifies the output mode. Valid values are:

- **TIMER\_D\_OUTPUTMODE\_OUTBITVALUE** [Default]
- **TIMER\_D\_OUTPUTMODE\_SET**
- **TIMER\_D\_OUTPUTMODE\_TOGGLE\_RESET**
- **TIMER\_D\_OUTPUTMODE\_SET\_RESET**
- **TIMER\_D\_OUTPUTMODE\_TOGGLE**
- **TIMER\_D\_OUTPUTMODE\_RESET**
- **TIMER\_D\_OUTPUTMODE\_TOGGLE\_SET**
- **TIMER\_D\_OUTPUTMODE\_RESET\_SET**

**compareValue** is the count to be compared with in compare mode

**Description:**

Modified bits of **TDxCCTLn** register and bits of **TDxCCRn** register.

**Returns:**

None

### 34.2.2.35 TIMER\_D\_initCompareLatchLoadEvent

Selects Compare Latch Load Event.

**Prototype:**

```
void
TIMER_D_initCompareLatchLoadEvent(uint16_t baseAddress,
                                   uint16_t compareRegister,
                                   uint16_t compareLatchLoadEvent)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**compareRegister** selects the compare register being used. Valid values are:

- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6

**compareLatchLoadEvent** selects the latch load event Valid values are:

- TIMER\_D\_LATCH\_ON\_WRITE\_TO\_TDxCCTLn\_COMPARE\_REGISTER [Default]
- TIMER\_D\_LATCH\_WHEN\_COUNTER\_COUNTS\_TO\_0\_IN\_UP\_OR\_CONT\_MODE
- TIMER\_D\_LATCH\_WHEN\_COUNTER\_COUNTS\_TO\_0\_IN\_UPDOWN\_MODE
- TIMER\_D\_LATCH\_WHEN\_COUNTER\_COUNTS\_TO\_CURRENT\_COMPARE\_LATCH\_VALUE

**Description:**

Modified bits are CLLD of TDxCCTLn register.

**Returns:**

None

### 34.2.2.36 TIMER\_D\_selectCounterLength

Selects TIMER\_D counter length.

**Prototype:**

```
void
TIMER_D_selectCounterLength(uint16_t baseAddress,
                             uint16_t counterLength)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**counterLength** selects the value of counter length. Valid values are:

- TIMER\_D\_COUNTER\_16BIT [Default]
- TIMER\_D\_COUNTER\_12BIT
- TIMER\_D\_COUNTER\_10BIT
- TIMER\_D\_COUNTER\_8BIT

**Description:**

Modified bits are CNTL of TDxCTL0 register.

**Returns:**

None

### 34.2.2.37 TIMER\_D\_selectHighResClockRange

Select High Resolution Clock Range Selection.

**Prototype:**

```
void
TIMER_D_selectHighResClockRange(uint32_t baseAddress,
                                uint16_t highResClockRange)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**highResClockRange** selects the High Resolution Clock Range. Refer to datasheet for frequency details Valid values are:

- **TIMER\_D\_CLOCK\_RANGE0** [Default]
- **TIMER\_D\_CLOCK\_RANGE1**
- **TIMER\_D\_CLOCK\_RANGE2**

**Returns:**

None

### 34.2.2.38 void TIMER\_D\_selectHighResCoarseClockRange (uint32\_t baseAddress, uint16\_t highResCoarseClockRange)

Select High Resolution Coarse Clock Range.

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**highResCoarseClockRange** selects the High Resolution Coarse Clock Range Valid values are:

- **TIMER\_D\_HIGHRES\_BELOW\_15MHz** [Default]
- **TIMER\_D\_HIGHRES\_ABOVE\_15MHz**

**Description:**

Modified bits are **TDHCLKCR** of **TDxHCTL1** register.

**Returns:**

None

### 34.2.2.39 TIMER\_D\_selectLatchingGroup

Selects TIMER\_D Latching Group.

**Prototype:**

```
void
TIMER_D_selectLatchingGroup(uint16_t baseAddress,
                             uint16_t groupLatch)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**groupLatch** selects the group latch Valid values are:

- **TIMER\_D\_GROUP\_NONE** [Default]
- **TIMER\_D\_GROUP\_CL12\_CL23\_CL56**
- **TIMER\_D\_GROUP\_CL123\_CL456**
- **TIMER\_D\_GROUP\_ALL**

**Description:**

Modified bits are **TDCLGRP** of **TDxCTL0** register.

**Returns:**

None

### 34.2.2.40 TIMER\_D\_setCompareValue

Sets the value of the capture-compare register.

**Prototype:**

```
void
TIMER_D_setCompareValue(uint32_t baseAddress,
                        uint16_t compareRegister,
                        uint16_t compareValue)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**compareRegister** selects the Capture register being used. Valid values are:

- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6

**compareValue** is the count to be compared with in compare mode

**Description:**

Modified bits of TDxCCRn register.

**Returns:**

None

### 34.2.2.41 TIMER\_D\_setOutputForOutputModeOutBitValue

Set output bit for output mode.

**Prototype:**

```
void
TIMER_D_setOutputForOutputModeOutBitValue(uint32_t baseAddress,
                                           uint16_t captureCompareRegister,
                                           uint8_t outputModeOutBitValue)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**captureCompareRegister** selects the Capture register being used. Valid values are:

- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_0
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_1
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_2
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_3
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_4
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_5
- TIMER\_D\_CAPTURECOMPARE\_REGISTER\_6

**outputModeOutBitValue** the value to be set for out bit Valid values are:

- TIMER\_D\_OUTPUTMODE\_OUTBITVALUE\_HIGH
- TIMER\_D\_OUTPUTMODE\_OUTBITVALUE\_LOW

**Description:**

Modified bits of TDxCCTLn register.

**Returns:**

None

### 34.2.2.42 TIMER\_D\_startCounter

Starts TIMER\_D counter.

**Prototype:**

```
void
TIMER_D_startCounter(uint32_t baseAddress,
                    uint16_t timerMode)
```

**Description:**

NOTE: This function assumes that the timer has been previously configured using TIMER\_D\_configureContinuousMode, TIMER\_D\_configureUpMode or TIMER\_D\_configureUpDownMode.

**Parameters:**

**baseAddress** is the base address of the TIMER\_DA module.

**timerMode** selects the mode of the timer Valid values are:

- **TIMER\_D\_STOP\_MODE**
- **TIMER\_D\_UP\_MODE**
- **TIMER\_D\_CONTINUOUS\_MODE** [Default]
- **TIMER\_D\_UPDOWN\_MODE**

Modified bits of **TDxCTL0** register.

**Returns:**

None

### 34.2.2.43 TIMER\_D\_stop

Stops the timer.

**Prototype:**

```
void
TIMER_D_stop(uint32_t baseAddress)
```

**Parameters:**

**baseAddress** is the base address of the TIMER\_D module.

**Description:**

Modified bits of **TDxCTL0** register.

**Returns:**

None

## 34.3 Programming Example

The following example shows some TimerD operations using the APIs

```
{
    //Start TimerD
    TIMER_D_configureUpDownMode( TIMER_A1_BASE,
    TIMER_D_CLOCKSOURCE_SMCLK,
    TIMER_D_CLOCKSOURCE_DIVIDER_1,
    TIMER_PERIOD,
    TIMER_D_TAIE_INTERRUPT_DISABLE,
    TIMER_D_CCIE_CCR0_INTERRUPT_DISABLE,
    TIMER_D_DO_CLEAR
    );

    TIMER_D_startCounter( TIMER_A1_BASE,
```



```
        TIMER_D_UPDOWN_MODE
    );

    //Initialize compare registers to generate PWM1
    TIMER_D_initCompare(TIMER_A1_BASE,
        TIMER_D_CAPTURECOMPARE_REGISTER_1,
        TIMER_D_CAPTURECOMPARE_INTERRUPT_ENABLE,
        TIMER_D_OUTPUTMODE_TOGGLE_SET,
        DUTY_CYCLE1
    );
    //Initialize compare registers to generate PWM2
    TIMER_D_initCompare(TIMER_A1_BASE,
        TIMER_D_CAPTURECOMPARE_REGISTER_2,
        TIMER_D_CAPTURECOMPARE_INTERRUPT_DISABLE,
        TIMER_D_OUTPUTMODE_TOGGLE_SET,
        DUTY_CYCLE2
    );

    //Enter LPM0
    __bis_SR_register(LPM0_bits);

    //For debugger
    __no_operation();
}
```

## 35 Tag Length Value

Introduction .....	438
API Functions .....	438
Programming Example .....	443

### 35.1 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

### 35.2 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)`

#### 35.2.1 Detailed Description

The APIs that help in querying the information in the TLV structure are listed

- `TLV_getInfo()` This function retrieves the value of a tag and the length of the tag.
- `TLV_getDeviceType()` This function retrieves the unique device ID from the TLV structure.
- `TLV_getMemory()` The returned value is zero if the end of the memory list is reached.
- `TLV_getPeripheral()` The returned value is zero if the specified tag value (peripheral) is not available in the device.
- `TLV_getInterrupt()` The returned value is zero if the specified interrupt vector is not defined.

## 35.2.2 Function Documentation

### 35.2.2.1 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.

### 35.2.2.2 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\_LDRTAG
- 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

### 35.2.2.3 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 if the specified interrupt vector is not defined.

### 35.2.2.4 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.

### 35.2.2.5 uint16\_t TLV\_getPeripheral (uint8\_t tag, uint8\_t instance)

Gets peripheral information from the TLV.

The Peripheral Descriptor tag is split into two portions a list of the available flash memory blocks followed by a list of available peripherals. This function is used to parse through the second portion and can be used to check if a specific peripheral is present in a device. The function calls `TLV_getPeripheral()` recursively until the end of the memory list and consequently the beginning of the peripheral list is reached. <

**Code Metrics:**

Compiler	Optimization	Code Size
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.

## 35.3 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
            );
```

## 36 Unified Clock System (UCS)

Introduction .....	444
API Functions .....	445
Programming Example .....	462

### 36.1 Introduction

The UCS is based on five available clock sources (VLO, REFO, XT1, XT2, and DCO) providing signals to three system clocks (MCLK, SMCLK, ACLK). Different low power modes are achieved by turning off the MCLK, SMCLK, ACLK, and integrated LDO.

- VLO - Internal very-low-power low-frequency oscillator. 10 kHz ( $\pm 0.5\%/^{\circ}\text{C}$ ,  $\pm 4\text{V}$ )
- REFO - Reference oscillator. 32 kHz ( $\pm 1\%$ ,  $\pm 3\%$  over full temp range)
- XT1 (LFXT1, HFXT1) - Ultra-low-power oscillator, compatible with low-frequency 32768-Hz watch crystals and with standard XT1 (LFXT1, HFXT1) crystals, resonators, or external clock sources in the 4-MHz to 32-MHz range, including digital inputs. Most commonly used as 32-kHz watch crystal oscillator.
- XT2 - Optional high-frequency oscillator that can be used with standard crystals, resonators, or external clock sources in the 4-MHz to 32-MHz range, including digital inputs.
- DCO - Internal digitally-controlled oscillator (DCO) that can be stabilized by a frequency lock loop (FLL) that sets the DCO to a specified multiple of a reference frequency.

System Clocks and Functionality on the MSP430 MCLK Master Clock Services the CPU. Commonly sourced by DCO. Is available in Active mode only SMCLK Subsystem Master Clock Services 'fast' system peripherals. Commonly sourced by DCO. Is available in Active mode, LPM0 and LPM1 ACLK Auxiliary Clock Services 'slow' system peripherals. Commonly used for 32-kHz signal. Is available in Active mode, LPM0 to LPM3

System clocks of the MSP430x5xx generation are automatically enabled, regardless of the LPM mode of operation, if they are required for the proper operation of the peripheral module that they source. This additional flexibility of the UCS, along with improved fail-safe logic, provides a robust clocking scheme for all applications.

**Fail-Safe logic** The UCS fail-safe logic plays an important part in providing a robust clocking scheme for MSP430x5xx and MSP430x6xx applications. This feature hinges on the ability to detect an oscillator fault for the XT1 in both low- and high-frequency modes (XT1LFOFFG and XT1HFOFFG respectively), the high-frequency XT2 (XT2OFFG), and the DCO (DCOFFG). These flags are set and latched when the respective oscillator is enabled but not operating properly; therefore, they must be explicitly cleared in software

The oscillator fault flags on previous MSP430 generations are not latched and are asserted only as long as the failing condition exists. Therefore, an important difference between the families is that the fail-safe behavior in a 5xx-based MSP430 remains active until both the OFIFG and the respective fault flag are cleared in software.

This fail-safe behavior is implemented at the oscillator level, at the system clock level and, consequently, at the module level. Some notable highlights of this behavior are described below. For the full description of fail-safe behavior and conditions, see the MSP430x5xx/MSP430x6xx Family User's Guide (SLAU208).

- Low-frequency crystal oscillator 1 (LFXT1) The low-frequency (32768 Hz) crystal oscillator is the default reference clock to the FLL. An asserted XT1LFOFFG switches the FLL reference from the failing LFXT1 to the internal 32-kHz REFO. This can influence the DCO accuracy, because the FLL crystal ppm specification is typically tighter than the REFO accuracy over temperature and voltage of  $\pm 3\%$ .
- System Clocks (ACLK, SMCLK, MCLK) A fault on the oscillator that is sourcing a system clock switches the source from the failing oscillator to the DCO oscillator (DCOCLKDIV). This is true for all clock sources except the LFXT1. As previously described, a fault on the LFXT1 switches the source to the REFO. Since ACLK is the active clock in LPM3 there is a notable difference in the LPM3 current consumption when the REFO is the clock source ( $\sim 3\text{ }\mu\text{A}$  active) versus the LFXT1 ( $\sim 300\text{ nA}$  active).
- Modules (WDT\_A) In watchdog mode, when SMCLK or ACLK fails, the clock source defaults to the VLOCLK.

This driver is contained in `ucs.c`, with `ucs.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	1590
CCS 4.2.1	Size	1160
CCS 4.2.1	Speed	1168
IAR 5.51.6	None	1254
IAR 5.51.6	Size	808
IAR 5.51.6	Speed	1190
MSPGCC 4.8.0	None	3512
MSPGCC 4.8.0	Size	1326
MSPGCC 4.8.0	Speed	2228

## 36.2 API Functions

### Functions

- void [UCS\\_bypassXT1](#) (uint8\_t highOrLowFrequency)
- bool [UCS\\_bypassXT1WithTimeout](#) (uint8\_t highOrLowFrequency, uint16\_t timeout)
- void [UCS\\_bypassXT2](#) (void)
- bool [UCS\\_bypassXT2WithTimeout](#) (uint16\_t timeout)
- uint16\_t [UCS\\_clearAllOscFlagsWithTimeout](#) (uint16\_t timeout)
- void [UCS\\_clearFaultFlag](#) (uint8\_t mask)
- void [UCS\\_clockSignalInit](#) (uint8\_t selectedClockSignal, uint16\_t clockSource, uint16\_t clockSourceDivider)
- void [UCS\\_disableClockRequest](#) (uint8\_t selectClock)
- void [UCS\\_enableClockRequest](#) (uint8\_t selectClock)
- uint8\_t [UCS\\_faultFlagStatus](#) (uint8\_t mask)
- uint32\_t [UCS\\_getACLK](#) (void)
- uint32\_t [UCS\\_getMCLK](#) (void)
- uint32\_t [UCS\\_getSMCLK](#) (void)
- void [UCS\\_HFXT1Start](#) (uint16\_t xt1drive)
- bool [UCS\\_HFXT1StartWithTimeout](#) (uint16\_t xt1drive, uint16\_t timeout)
- void [UCS\\_initFLL](#) (uint16\_t fsystem, uint16\_t ratio)
- void [UCS\\_initFLLSettle](#) (uint16\_t fsystem, uint16\_t ratio)
- void [UCS\\_LFXT1Start](#) (uint16\_t xt1drive, uint8\_t xcap)
- bool [UCS\\_LFXT1StartWithTimeout](#) (uint16\_t xt1drive, uint8\_t xcap, uint16\_t timeout)
- void [UCS\\_setExternalClockSource](#) (uint32\_t XT1CLK\_frequency, uint32\_t XT2CLK\_frequency)
- void [UCS\\_SMCLKOff](#) (void)
- void [UCS\\_SMCLKOn](#) (void)
- void [UCS\\_XT1Off](#) (void)
- void [UCS\\_XT2Off](#) (void)
- void [UCS\\_XT2Start](#) (uint16\_t xt2drive)
- bool [UCS\\_XT2StartWithTimeout](#) (uint16\_t xt2drive, uint16\_t timeout)

### 36.2.1 Detailed Description

The UCS API is broken into three groups of functions: those that deal with clock configuration and control

General UCS configuration and initialization is handled by

- [UCS\\_clockSignalInit\(\)](#),

- [UCS\\_initFLLSettle\(\)](#),
- [UCS\\_enableClockRequest\(\)](#),
- [UCS\\_disableClockRequest\(\)](#),
- [UCS\\_SMCLKOff\(\)](#),
- [UCS\\_SMCLKOn\(\)](#)

External crystal specific configuration and initialization is handled by

- [UCS\\_setExternalClockSource\(\)](#),
- [UCS\\_LFXT1Start\(\)](#),
- [UCS\\_HFXT1Start\(\)](#),
- [UCS\\_bypassXT1\(\)](#),
- [UCS\\_LFXT1StartWithTimeout\(\)](#),
- [UCS\\_HFXT1StartWithTimeout\(\)](#),
- [UCS\\_bypassXT1WithTimeout\(\)](#),
- [UCS\\_XT1Off\(\)](#),
- [UCS\\_XT2Start\(\)](#),
- [UCS\\_XT2Off\(\)](#),
- [UCS\\_bypassXT2\(\)](#),
- [UCS\\_XT2StartWithTimeout\(\)](#),
- [UCS\\_bypassXT2WithTimeout\(\)](#)
- [UCS\\_clearAllOscFlagsWithTimeout\(\)](#)

[UCS\\_setExternalClockSource](#) must be called if an external crystal XT1 or XT2 is used and the user intends to call [UCS\\_getMCLK](#), [UCS\\_getSMCLK](#) or [UCS\\_getACLK](#) APIs. If not, it is not necessary to invoke this API.

Failure to invoke [UCS\\_clockSignalInit\(\)](#) sets the clock signals to the default modes ACLK default mode - [UCS\\_XT1CLK\\_SELECT](#) SMCLK default mode - [UCS\\_DCOCLKDIV\\_SELECT](#) MCLK default mode - [UCS\\_DCOCLKDIV\\_SELECT](#)

Also fail-safe mode behavior takes effect when a selected mode fails.

The status and configuration query are done by

- [UCS\\_faultFlagStatus\(\)](#),
- [UCS\\_clearFaultFlag\(\)](#),
- [UCS\\_getACLK\(\)](#),
- [UCS\\_getSMCLK\(\)](#),
- [UCS\\_getMCLK\(\)](#)

## 36.2.2 Function Documentation

### 36.2.2.1 UCS\_bypassXT1

Bypass the XT1 crystal oscillator.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	66
CCS 4.2.1	Size	54
CCS 4.2.1	Speed	56
IAR 5.51.6	None	56
IAR 5.51.6	Size	30
IAR 5.51.6	Speed	94
MSPGCC 4.8.0	None	212
MSPGCC 4.8.0	Size	62
MSPGCC 4.8.0	Speed	64

**Prototype:**

```
void
UCS_bypassXT1(uint8_t highOrLowFrequency)
```

**Description:**

Bypasses the XT1 crystal oscillator. Loops until all oscillator fault flags are cleared, with no timeout.

**Parameters:**

**highOrLowFrequency** selects high frequency or low frequency mode for XT1. Valid values are:

- **UCS\_XT1\_HIGH\_FREQUENCY**
- **UCS\_XT1\_LOW\_FREQUENCY** [Default]

Modified bits of **UCSCTL7** register, bits of **UCSCTL6** register and bits of **SFRIFG** register.

**Returns:**

None

### 36.2.2.2 UCS\_bypassXT1WithTimeout

Bypasses the XT1 crystal oscillator with time out.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	92
CCS 4.2.1	Size	72
CCS 4.2.1	Speed	72
IAR 5.51.6	None	78
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	244
MSPGCC 4.8.0	Size	82
MSPGCC 4.8.0	Speed	90

**Prototype:**

```
bool
UCS_bypassXT1WithTimeout(uint8_t highOrLowFrequency,
                          uint16_t timeout)
```

**Description:**

Bypasses the XT1 crystal oscillator with time out. Loops until all oscillator fault flags are cleared or until a timeout counter is decremented and equals to zero.

**Parameters:**

**highOrLowFrequency** selects high frequency or low frequency mode for XT1. Valid values are:

- **UCS\_XT1\_HIGH\_FREQUENCY**
- **UCS\_XT1\_LOW\_FREQUENCY** [Default]

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

Modified bits of **UCSCTL7** register, bits of **UCSCTL6** register and bits of **SFRIFG** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL

### 36.2.2.3 UCS\_bypassXT2

Bypasses the XT2 crystal oscillator.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	24
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	24
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	6
MSPGCC 4.8.0	None	88
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	28

**Prototype:**

```
void
UCS_bypassXT2(void)
```

**Description:**

Bypasses the XT2 crystal oscillator, which supports crystal frequencies between 4 MHz and 32 MHz. Loops until all oscillator fault flags are cleared, with no timeout.

Modified bits of **UCSCTL7** register, bits of **UCSCTL6** register and bits of **SFRIFG** register.

**Returns:**

None

### 36.2.2.4 UCS\_bypassXT2WithTimeout

Bypasses the XT2 crystal oscillator with timeout.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	48
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	36
IAR 5.51.6	None	40
IAR 5.51.6	Size	28
IAR 5.51.6	Speed	36
MSPGCC 4.8.0	None	118
MSPGCC 4.8.0	Size	48
MSPGCC 4.8.0	Speed	48

**Prototype:**

```
bool
UCS_bypassXT2WithTimeout(uint16_t timeout)
```

**Description:**

Bypasses the XT2 crystal oscillator, which supports crystal frequencies between 4 MHz and 32 MHz. Loops until all oscillator fault flags are cleared or until a timeout counter is decremented and equals to zero.

**Parameters:**

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

Modified bits of **UCSCTL7** register, bits of **UCSCTL6** register and bits of **SFRIFG** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL

## 36.2.2.5 UCS\_clearAllOscFlagsWithTimeout

Clears all the Oscillator Flags.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	40
CCS 4.2.1	Size	30
CCS 4.2.1	Speed	30
IAR 5.51.6	None	36
IAR 5.51.6	Size	30
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	100
MSPGCC 4.8.0	Size	36
MSPGCC 4.8.0	Speed	36

**Prototype:**

```
uint16_t
UCS_clearAllOscFlagsWithTimeout(uint16_t timeout)
```

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

Logical OR of any of the following:

- **UCS\_XT2OFFG** XT2 oscillator fault flag
- **UCS\_XT1HFOFFG** XT1 oscillator fault flag (HF mode)
- **UCS\_XT1LFOFFG** XT1 oscillator fault flag (LF mode)
- **UCS\_DCOFFG** DCO fault flag  
indicating the status of the oscillator fault flags

36.2.2.6 void UCS\_clearFaultFlag (uint8\_t *mask*)

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

**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

**Parameters:****mask** is the masked interrupt flag status to be returned. mask parameter can be any one of the following Valid values are:

- **UCS\_XT2OFFG** - XT2 oscillator fault flag
- **UCS\_XT1HFOFFG** - XT1 oscillator fault flag (HF mode)
- **UCS\_XT1LFOFFG** - XT1 oscillator fault flag (LF mode)
- **UCS\_DCOFFG** - DCO fault flag

**Description:**

Modified bits of **UCSCTL7** register.

**Returns:**

None

### 36.2.2.7 UCS\_clockSignalInit

Initializes a clock signal.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	208
CCS 4.2.1	Size	136
CCS 4.2.1	Speed	144
IAR 5.51.6	None	150
IAR 5.51.6	Size	134
IAR 5.51.6	Speed	134
MSPGCC 4.8.0	None	526
MSPGCC 4.8.0	Size	168
MSPGCC 4.8.0	Speed	172

**Prototype:**

```
void
UCS_clockSignalInit(uint8_t selectedClockSignal,
                    uint16_t clockSource,
                    uint16_t clockSourceDivider)
```

**Description:**

This function initializes each of the clock signals. The user must ensure that this function is called for each clock signal. If not, the default state is assumed for the particular clock signal. Refer MSP430Ware documentation for UCS module or Device Family User's Guide for details of default clock signal states.

**Parameters:**

***selectedClockSignal*** selected clock signal Valid values are:

- **UCS\_ACLK**
- **UCS\_MCLK**
- **UCS\_SMCLK**
- **UCS\_FLLREF**

***clockSource*** is clock source for the selectedClockSignal Valid values are:

- **UCS\_XT1CLK\_SELECT**
- **UCS\_VLOCLK\_SELECT**
- **UCS\_REFOCLK\_SELECT**
- **UCS\_DCOCLK\_SELECT**
- **UCS\_DCOCLKDIV\_SELECT**
- **UCS\_XT2CLK\_SELECT**

***clockSourceDivider*** selected the clock divider to calculate clocksignal from clock source. Valid values are:

- **UCS\_CLOCK\_DIVIDER\_1** [Default]
- **UCS\_CLOCK\_DIVIDER\_2**
- **UCS\_CLOCK\_DIVIDER\_4**
- **UCS\_CLOCK\_DIVIDER\_8**

- **UCS\_CLOCK\_DIVIDER\_12** - [Valid only for UCS\_FLLREF]
- **UCS\_CLOCK\_DIVIDER\_16**
- **UCS\_CLOCK\_DIVIDER\_32** - [Not valid for UCS\_FLLREF]

Modified bits of **UCSCTL5** register, bits of **UCSCTL4** register and bits of **UCSCTL3** register.

**Returns:**

None

### 36.2.2.8 UCS\_disableClockRequest

Disables conditional module requests.

**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
UCS_disableClockRequest(uint8_t selectClock)
```

**Parameters:**

**selectClock** selects specific request disable Valid values are:

- **UCS\_ACLK**
- **UCS\_SMCLK**
- **UCS\_MCLK**
- **UCS\_MODOSC**

**Description:**

Modified bits of **UCSCTL8** register.

**Returns:**

None

### 36.2.2.9 UCS\_enableClockRequest

Enables conditional module requests.

**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
UCS_enableClockRequest(uint8_t selectClock)
```

**Parameters:**

**selectClock** selects specific request enables Valid values are:

- UCS\_ACLK
- UCS\_SMCLK
- UCS\_MCLK
- UCS\_MODOSC

**Description:**

Modified bits of **UCSCTL8** register.

**Returns:**

None

### 36.2.2.10 UCS\_faultFlagStatus

Gets the current UCS fault flag status.

**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
UCS_faultFlagStatus(uint8_t mask)
```

**Parameters:**

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

Valid values are:

- UCS\_XT2OFFG - XT2 oscillator fault flag
- UCS\_XT1HFOFFG - XT1 oscillator fault flag (HF mode)
- UCS\_XT1LFOFFG - XT1 oscillator fault flag (LF mode)
- UCS\_DCOFFG - DCO fault flag

### 36.2.2.11 uint32\_t UCS\_getACLK (void)

Get the current ACLK frequency.

Get the current ACLK frequency. The user of this API must ensure that UCS\_setExternalClockSource API was invoked before in case XT1 or XT2 is being used.

**Code Metrics:**



Compiler	Optimization	Code Size
CCS 4.2.1	None	50
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	42
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	70
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	286

**Returns:**

Current ACLK frequency in Hz

## 36.2.2.12 uint32\_t UCS\_getMCLK (void)

Get the current MCLK frequency.

Get the current MCLK frequency. The user of this API must ensure that UCS\_setExternalClockSource API was invoked before in case XT1 or XT2 is being used.

**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	30
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	48
MSPGCC 4.8.0	Size	22
MSPGCC 4.8.0	Speed	278

**Returns:**

Current MCLK frequency in Hz

## 36.2.2.13 uint32\_t UCS\_getSMCLK (void)

Get the current SMCLK frequency.

Get the current SMCLK frequency. The user of this API must ensure that UCS\_setExternalClockSource API was invoked before in case XT1 or XT2 is being used.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	46
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	36
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	74
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	286

**Returns:**

Current SMCLK frequency in Hz

**36.2.2.14 void UCS\_HFXT1Start (uint16\_t xt1drive)**

Initializes the XT1 crystal oscillator in low frequency mode.

Initializes the XT1 crystal oscillator in high frequency mode. Loops until all oscillator fault flags are cleared, with no timeout. See the device- specific data sheet for appropriate drive settings.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	64
CCS 4.2.1	Size	56
CCS 4.2.1	Speed	56
IAR 5.51.6	None	56
IAR 5.51.6	Size	16
IAR 5.51.6	Speed	68
MSPGCC 4.8.0	None	178
MSPGCC 4.8.0	Size	58
MSPGCC 4.8.0	Speed	60

**Parameters:****xt1drive** is the target drive strength for the XT1 crystal oscillator. Valid values are:

- **UCS\_XT1\_DRIVE0**
- **UCS\_XT1\_DRIVE1**
- **UCS\_XT1\_DRIVE2**
- **UCS\_XT1\_DRIVE3** [Default]

**Description:**Modified bits of **UCSCTL7** register, bits of **UCSCTL6** register and bits of **SFRIFG** register.**Returns:**

None

**36.2.2.15 UCS\_HFXT1StartWithTimeout**

Initializes the XT1 crystal oscillator in high frequency mode with timeout.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	84
CCS 4.2.1	Size	68
CCS 4.2.1	Speed	68
IAR 5.51.6	None	72
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	60
MSPGCC 4.8.0	None	208
MSPGCC 4.8.0	Size	76
MSPGCC 4.8.0	Speed	70

**Prototype:**

```
bool
UCS_HFXT1StartWithTimeout (uint16_t xt1drive,
                           uint16_t timeout)
```

**Description:**

Initializes the XT1 crystal oscillator in high frequency mode with timeout. Loops until all oscillator fault flags are cleared or until a timeout counter is decremented and equals to zero. See the device-specific data sheet for appropriate drive settings.

**Parameters:**

**xt1drive** is the target drive strength for the XT1 crystal oscillator. Valid values are:

- **UCS\_XT1\_DRIVE0**
- **UCS\_XT1\_DRIVE1**
- **UCS\_XT1\_DRIVE2**
- **UCS\_XT1\_DRIVE3** [Default]

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

Modified bits of **UCSCTL7** register, bits of **UCSCTL6** register and bits of **SFRIFG** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL

### 36.2.2.16 UCS\_initFLL

Initializes the DCO to operate a frequency that is a multiple of the reference frequency into the FLL.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	300
CCS 4.2.1	Size	224
CCS 4.2.1	Speed	224
IAR 5.51.6	None	228
IAR 5.51.6	Size	220
IAR 5.51.6	Speed	258
MSPGCC 4.8.0	None	508
MSPGCC 4.8.0	Size	264
MSPGCC 4.8.0	Speed	354

**Prototype:**

```
void
UCS_initFLL(uint16_t fsystem,
            uint16_t ratio)
```

**Description:**

Initializes the DCO to operate a frequency that is a multiple of the reference frequency into the FLL. Loops until all oscillator fault flags are cleared, with no timeout. If the frequency is greater than 16 MHz, the function sets the MCLK and SMCLK source to the undivided DCO frequency. Otherwise, the function sets the MCLK and SMCLK source to the DCOCLKDIV frequency.

**Parameters:**

**fsystem** is the target frequency for MCLK in kHz

**ratio** is the ratio  $x/y$ , where  $x = \text{fsystem}$  and  $y = \text{FLL reference frequency}$ .

Modified bits of **UCSCTL0** register, bits of **UCSCTL4** register, bits of **UCSCTL7** register, bits of **UCSCTL1** register, bits of **SFRIFG1** register and bits of **UCSCTL2** register.

**Returns:**

None

### 36.2.2.17 UCS\_initFLLSettle

Initializes the DCO to operate a frequency that is a multiple of the reference frequency into the FLL.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	78
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	50
IAR 5.51.6	None	58
IAR 5.51.6	Size	46
IAR 5.51.6	Speed	46
MSPGCC 4.8.0	None	86
MSPGCC 4.8.0	Size	66
MSPGCC 4.8.0	Speed	102

#### Prototype:

```
void
UCS_initFLLSettle(uint16_t fsystem,
                  uint16_t ratio)
```

#### Description:

Initializes the DCO to operate a frequency that is a multiple of the reference frequency into the FLL. Loops until all oscillator fault flags are cleared, with a timeout. If the frequency is greater than 16 MHz, the function sets the MCLK and SMCLK source to the undivided DCO frequency. Otherwise, the function sets the MCLK and SMCLK source to the DCOCLKDIV frequency. This function executes a software delay that is proportional in length to the ratio of the target FLL frequency and the FLL reference.

#### Parameters:

***fsystem*** is the target frequency for MCLK in kHz  
***ratio*** is the ratio  $x/y$ , where  $x = \text{fsystem}$  and  $y = \text{FLL reference frequency}$ .

Modified bits of **UCSCTL0** register, bits of **UCSCTL4** register, bits of **UCSCTL7** register, bits of **UCSCTL1** register, bits of **SFRIFG1** register and bits of **UCSCTL2** register.

#### Returns:

None

### 36.2.2.18 UCS\_LFXT1Start

Initializes the XT1 crystal oscillator in low frequency mode.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	82
CCS 4.2.1	Size	70
CCS 4.2.1	Speed	68
IAR 5.51.6	None	68
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	82
MSPGCC 4.8.0	None	192
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	74

#### Prototype:

```
void
UCS_LFXT1Start(uint16_t xtldrive,
               uint8_t xcap)
```

**Description:**

Initializes the XT1 crystal oscillator in low frequency mode. Loops until all oscillator fault flags are cleared, with no timeout. See the device- specific data sheet for appropriate drive settings.

**Parameters:**

**xt1drive** is the target drive strength for the XT1 crystal oscillator. Valid values are:

- **UCS\_XT1\_DRIVE0**
  - **UCS\_XT1\_DRIVE1**
  - **UCS\_XT1\_DRIVE2**
  - **UCS\_XT1\_DRIVE3** [Default]
- Modified bits are **XT1DRIVE** of **UCSCTL6** register.

**xcap** is the selected capacitor value. This parameter selects the capacitors applied to the LF crystal (XT1) or resonator in the LF mode. The effective capacitance (seen by the crystal) is  $C_{eff} = (CXIN + 2\text{ pF})/2$ . It is assumed that  $CXIN = CXOUT$  and that a parasitic capacitance of 2 pF is added by the package and the printed circuit board. For details about the typical internal and the effective capacitors, refer to the device-specific data sheet. Valid values are:

- **UCS\_XCAP\_0**
- **UCS\_XCAP\_1**
- **UCS\_XCAP\_2**
- **UCS\_XCAP\_3** [Default]

Modified bits are **XCAP** of **UCSCTL6** register.

**Returns:**

None

### 36.2.2.19 UCS\_LFXT1StartWithTimeout

Initializes the XT1 crystal oscillator in low frequency mode with timeout.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	106
CCS 4.2.1	Size	82
CCS 4.2.1	Speed	80
IAR 5.51.6	None	84
IAR 5.51.6	Size	40
IAR 5.51.6	Speed	74
MSPGCC 4.8.0	None	222
MSPGCC 4.8.0	Size	90
MSPGCC 4.8.0	Speed	84

**Prototype:**

```
bool
UCS_LFXT1StartWithTimeout(uint16_t xt1drive,
                           uint8_t xcap,
                           uint16_t timeout)
```

**Description:**

Initializes the XT1 crystal oscillator in low frequency mode with timeout. Loops until all oscillator fault flags are cleared or until a timeout counter is decremented and equals to zero. See the device-specific datasheet for appropriate drive settings.

**Parameters:**

**xt1drive** is the target drive strength for the XT1 crystal oscillator. Valid values are:

- **UCS\_XT1\_DRIVE0**
- **UCS\_XT1\_DRIVE1**
- **UCS\_XT1\_DRIVE2**

■ **UCS\_XT1\_DRIVE3** [Default]

**xcap** is the selected capacitor value. This parameter selects the capacitors applied to the LF crystal (XT1) or resonator in the LF mode. The effective capacitance (seen by the crystal) is  $C_{eff} = (C_{XIN} + 2 \text{ pF})/2$ . It is assumed that  $C_{XIN} = C_{XOUT}$  and that a parasitic capacitance of 2 pF is added by the package and the printed circuit board. For details about the typical internal and the effective capacitors, refer to the device-specific data sheet. Valid values are:

■ **UCS\_XCAP\_0**

■ **UCS\_XCAP\_1**

■ **UCS\_XCAP\_2**

■ **UCS\_XCAP\_3** [Default]

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

Modified bits of **UCSCTL7** register, bits of **UCSCTL6** register and bits of **SFRIFG** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL

### 36.2.2.20 UCS\_setExternalClockSource

Sets the external clock source.

**Code Metrics:**

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

**Prototype:**

```
void
UCS_setExternalClockSource(uint32_t XT1CLK_frequency,
                           uint32_t XT2CLK_frequency)
```

**Description:**

This function sets the external clock sources XT1 and XT2 crystal oscillator frequency values. This function must be called if an external crystal XT1 or XT2 is used and the user intends to call UCS\_getMCLK, UCS\_getSMCLK or UCS\_getACLK APIs. If not, it is not necessary to invoke this API.

**Parameters:**

**XT1CLK\_frequency** is the XT1 crystal frequencies in Hz

**XT2CLK\_frequency** is the XT2 crystal frequencies in Hz

**Returns:**

None

### 36.2.2.21 UCS\_SMCLKOff

Turns off SMCLK using the SMCLKOFF bit.

**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
UCS_SMCLKOff(void)
```

**Description:**

Modified bits of **UCSCTL6** register.

**Returns:**

None

### 36.2.2.22 UCS\_SMCLKOn

Turns ON SMCLK using the SMCLKOFF bit.

**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
UCS_SMCLKOn(void)
```

**Description:**

Modified bits of **UCSCTL6** register.

**Returns:**

None

### 36.2.2.23 UCS\_XT1Off

Stops the XT1 oscillator using the XT1OFF bit.

**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
UCS_XT1Off(void)
```

**Returns:**

None

### 36.2.2.24 void UCS\_XT2Off (void)

Stops the XT2 oscillator using the XT2OFF bit.

Modified bits of **UCSCTL6** register.

**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

**Returns:**

None

### 36.2.2.25 void UCS\_XT2Start (uint16\_t xt2drive)

Initializes the XT2 crystal oscillator.

Initializes the XT2 crystal oscillator, which supports crystal frequencies between 4 MHz and 32 MHz, depending on the selected drive strength. Loops until all oscillator fault flags are cleared, with no timeout. See the device-specific data sheet for appropriate drive settings.

**Code Metrics:**



Compiler	Optimization	Code Size
CCS 4.2.1	None	60
CCS 4.2.1	Size	52
CCS 4.2.1	Speed	52
IAR 5.51.6	None	52
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	36
MSPGCC 4.8.0	None	162
MSPGCC 4.8.0	Size	54
MSPGCC 4.8.0	Speed	56

**Parameters:**

**xt2drive** is the target drive strength for the XT2 crystal oscillator. Valid values are:

- **UCS\_XT2DRIVE\_4MHZ\_8MHZ**
- **UCS\_XT2DRIVE\_8MHZ\_16MHZ**
- **UCS\_XT2DRIVE\_16MHZ\_24MHZ**
- **UCS\_XT2DRIVE\_24MHZ\_32MHZ** [Default]

**Description:**

Modified bits of **UCSCTL7** register, bits of **UCSCTL6** register and bits of **SFRIFG** register.

**Returns:**

None

### 36.2.2.26 UCS\_XT2StartWithTimeout

Initializes the XT2 crystal oscillator with timeout.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	80
CCS 4.2.1	Size	64
CCS 4.2.1	Speed	64
IAR 5.51.6	None	68
IAR 5.51.6	Size	26
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	192
MSPGCC 4.8.0	Size	74
MSPGCC 4.8.0	Speed	72

**Prototype:**

```
bool
UCS_XT2StartWithTimeout(uint16_t xt2drive,
                        uint16_t timeout)
```

**Description:**

Initializes the XT2 crystal oscillator, which supports crystal frequencies between 4 MHz and 32 MHz, depending on the selected drive strength. Loops until all oscillator fault flags are cleared or until a timeout counter is decremented and equals to zero. See the device-specific data sheet for appropriate drive settings.

**Parameters:**

**xt2drive** is the target drive strength for the XT2 crystal oscillator. Valid values are:

- **UCS\_XT2DRIVE\_4MHZ\_8MHZ**
- **UCS\_XT2DRIVE\_8MHZ\_16MHZ**
- **UCS\_XT2DRIVE\_16MHZ\_24MHZ**
- **UCS\_XT2DRIVE\_24MHZ\_32MHZ** [Default]

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

Modified bits of **UCSCTL7** register, bits of **UCSCTL6** register and bits of **SFRIFG** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAIL

## 36.3 Programming Example

The following example shows some UCS operations using the APIs

```
// Set DCO FLL reference = REFO
UCS_clockSignalInit(UCS_BASE,
                    UCS_FLLREF,
                    UCS_REFOCLK_SELECT,
                    UCS_CLOCK_DIVIDER_1
                    );

// Set ACLK = REFO
UCS_clockSignalInit(UCS_BASE,
                    UCS_ACLK,
                    UCS_REFOCLK_SELECT,
                    UCS_CLOCK_DIVIDER_1
                    );

// Set Ratio and Desired MCLK Frequency and initialize DCO
UCS_initFLLSettle( UCS_BASE,
                  UCS_MCLK_DESIRED_FREQUENCY_IN_KHZ,
                  UCS_MCLK_FLLREF_RATIO
                  );

//Verify if the Clock settings are as expected
clockValue = UCS_getSMCLK (UCS_BASE);

while(1);
```

## 37 USCI Universal Asynchronous Receiver/Transmitter (USCI\_A\_UART)

Introduction .....	463
API Functions .....	464
Programming Example .....	475

### 37.1 Introduction

The MSP430Ware library for USCI\_A\_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

The modes of operations supported by the USCI\_A\_UART and the library include

- USCI\_A\_UART mode
- Idle-line multiprocessor mode
- Address-bit multiprocessor mode
- USCI\_A\_UART mode with automatic baud-rate detection

In USCI\_A\_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 `usci_a_uart.c`, with `usci_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 with 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	776
CCS 4.2.1	Size	364
CCS 4.2.1	Speed	364
IAR 5.51.6	None	542
IAR 5.51.6	Size	374
IAR 5.51.6	Speed	420
MSPGCC 4.8.0	None	1496
MSPGCC 4.8.0	Size	410
MSPGCC 4.8.0	Speed	528

## 37.2 API Functions

### Functions

- void [USCI\\_A\\_UART\\_clearInterruptFlag](#) (uint32\_t baseAddress, uint8\_t mask)
- void [USCI\\_A\\_UART\\_disable](#) (uint32\_t baseAddress)
- void [USCI\\_A\\_UART\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- void [USCI\\_A\\_UART\\_enable](#) (uint32\_t baseAddress)
- void [USCI\\_A\\_UART\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- uint8\_t [USCI\\_A\\_UART\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t mask)
- uint32\_t [USCI\\_A\\_UART\\_getReceiveBufferAddressForDMA](#) (uint32\_t baseAddress)
- uint32\_t [USCI\\_A\\_UART\\_getTransmitBufferAddressForDMA](#) (uint32\_t baseAddress)
- bool [USCI\\_A\\_UART\\_initAdvance](#) (uint32\_t baseAddress, uint8\_t selectClockSource, uint16\_t clockPrescaler, uint8\_t firstModReg, uint8\_t secondModReg, uint8\_t parity, uint8\_t msborLsbFirst, uint8\_t numberOfStopBits, uint8\_t uartMode, uint8\_t overSampling)
- uint8\_t [USCI\\_A\\_UART\\_queryStatusFlags](#) (uint32\_t baseAddress, uint8\_t mask)
- uint8\_t [USCI\\_A\\_UART\\_receiveData](#) (uint32\_t baseAddress)
- void [USCI\\_A\\_UART\\_resetDormant](#) (uint32\_t baseAddress)
- void [USCI\\_A\\_UART\\_setDormant](#) (uint32\_t baseAddress)
- void [USCI\\_A\\_UART\\_transmitAddress](#) (uint32\_t baseAddress, uint8\_t transmitAddress)
- void [USCI\\_A\\_UART\\_transmitBreak](#) (uint32\_t baseAddress)
- void [USCI\\_A\\_UART\\_transmitData](#) (uint32\_t baseAddress, uint8\_t transmitData)

### 37.2.1 Detailed Description

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

The USCI\_A\_UART API is broken into three groups of functions: those that deal with configuration and control of the USCI\_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 USCI\_A\_UART are handled by the

- [USCI\\_A\\_UART\\_init\(\)](#)
- [USCI\\_A\\_UART\\_initAdvance\(\)](#)
- [USCI\\_A\\_UART\\_enable\(\)](#)
- [USCI\\_A\\_UART\\_disable\(\)](#)
- [USCI\\_A\\_UART\\_setDormant\(\)](#)
- [USCI\\_A\\_UART\\_resetDormant\(\)](#)

Sending and receiving data via the USCI\_A\_UART is handled by the

- [USCI\\_A\\_UART\\_transmitData\(\)](#)
- [USCI\\_A\\_UART\\_receiveData\(\)](#)
- [USCI\\_A\\_UART\\_transmitAddress\(\)](#)
- [USCI\\_A\\_UART\\_transmitBreak\(\)](#)

Managing the USCI\_A\_UART interrupts and status are handled by the

- [USCI\\_A\\_UART\\_enableInterrupt\(\)](#)
- [USCI\\_A\\_UART\\_disableInterrupt\(\)](#)
- [USCI\\_A\\_UART\\_getInterruptStatus\(\)](#)

- [USCI\\_A\\_UART\\_clearInterruptFlag\(\)](#)
- [USCI\\_A\\_UART\\_queryStatusFlags\(\)](#)

DMA related

- [USCI\\_A\\_UART\\_getReceiveBufferAddressForDMA\(\)](#)
- [USCI\\_A\\_UART\\_getTransmitBufferAddressForDMA\(\)](#)

## 37.2.2 Function Documentation

### 37.2.2.1 USCI\_A\_UART\_clearInterruptFlag

Clears UART 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	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	72
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

#### Prototype:

```
void
USCI_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 USCI\_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:

- **USCI\_A\_UART\_RECEIVE\_INTERRUPT\_FLAG** - Receive interrupt flag
- **USCI\_A\_UART\_TRANSMIT\_INTERRUPT\_FLAG** - Transmit interrupt flag

Modified bits of **UCAxIFG** register.

#### Returns:

None

### 37.2.2.2 USCI\_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	34
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_A_UART_disable(uint32_t baseAddress)
```

**Description:**

This will disable operation of the UART block.

**Parameters:**

**baseAddress** is the base address of the USCI\_A\_UART module.

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

**Returns:**

None

### 37.2.2.3 USCI\_A\_UART\_disableInterrupt

Disables individual UART interrupt sources.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	78
CCS 4.2.1	Size	30
CCS 4.2.1	Speed	30
IAR 5.51.6	None	44
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	146
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

**Prototype:**

```
void
USCI_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 USCI\_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:

- **USCI\_A\_UART\_RECEIVE\_INTERRUPT** - Receive interrupt
- **USCI\_A\_UART\_TRANSMIT\_INTERRUPT** - Transmit interrupt

- **USCI\_A\_UART\_RECEIVE\_ERRONEOUSCHAR\_INTERRUPT** - Receive erroneous-character interrupt enable
- **USCI\_A\_UART\_BREAKCHAR\_INTERRUPT** - Receive break character interrupt enable

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

**Returns:**  
None

### 37.2.2.4 USCI\_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	36
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_A_UART_enable(uint32_t baseAddress)
```

**Description:**

This will enable operation of the UART block.

**Parameters:**

***baseAddress*** is the base address of the USCI\_A\_UART module.

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

**Returns:**  
None

### 37.2.2.5 USCI\_A\_UART\_enableInterrupt

Enables individual UART interrupt sources.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	66
CCS 4.2.1	Size	20
CCS 4.2.1	Speed	20
IAR 5.51.6	None	36
IAR 5.51.6	Size	30
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	102
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

**Prototype:**

```
void
USCI_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 USCI\_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:

- **USCI\_A\_UART\_RECEIVE\_INTERRUPT** - Receive interrupt
- **USCI\_A\_UART\_TRANSMIT\_INTERRUPT** - Transmit interrupt
- **USCI\_A\_UART\_RECEIVE\_ERRONEOUSCHAR\_INTERRUPT** - Receive erroneous-character interrupt enable
- **USCI\_A\_UART\_BREAKCHAR\_INTERRUPT** - Receive break character interrupt enable

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

**Returns:**

None

### 37.2.2.6 USCI\_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	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint8_t
USCI_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 USCI\_A\_UART module.

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

- **USCI\_A\_UART\_RECEIVE\_INTERRUPT\_FLAG** - Receive interrupt flag
- **USCI\_A\_UART\_TRANSMIT\_INTERRUPT\_FLAG** - Transmit interrupt flag

Modified bits of **UCAxIFG** register.

**Returns:**

Logical OR of any of the following:



- **USCI\_A\_UART\_RECEIVE\_INTERRUPT\_FLAG** Receive interrupt flag
- **USCI\_A\_UART\_TRANSMIT\_INTERRUPT\_FLAG** Transmit interrupt flag  
indicating the status of the masked flags

### 37.2.2.7 USCI\_A\_UART\_getReceiveBufferAddressForDMA

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
USCI_A_UART_getReceiveBufferAddressForDMA(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 USCI\_A\_UART module.

#### Returns:

Address of RX Buffer

### 37.2.2.8 USCI\_A\_UART\_getTransmitBufferAddressForDMA

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
USCI_A_UART_getTransmitBufferAddressForDMA(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 USCI\_A\_UART module.

**Returns:**

Address of TX Buffer

### 37.2.2.9 USCI\_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	230
CCS 4.2.1	Size	162
CCS 4.2.1	Speed	162
IAR 5.51.6	None	224
IAR 5.51.6	Size	170
IAR 5.51.6	Speed	170
MSPGCC 4.8.0	None	562
MSPGCC 4.8.0	Size	160
MSPGCC 4.8.0	Speed	272

**Prototype:**

```
bool
USCI_A_UART_initAdvance(uint32_t baseAddress,
                        uint8_t selectClockSource,
                        uint16_t clockPrescalar,
                        uint8_t firstModReg,
                        uint8_t secondModReg,
                        uint8_t parity,
                        uint8_t msborLsbFirst,
                        uint8_t numberOfStopBits,
                        uint8_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 [USCI\\_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](http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html)

**Parameters:**

**baseAddress** is the base address of the USCI\_A\_UART module.

**selectClockSource** selects Clock source. Valid values are:

- **USCI\_A\_UART\_CLOCKSOURCE\_SMCLK**
- **USCI\_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 UCxMCTLW.

**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 UCxMCTLW.

**parity** is the desired parity. Valid values are:

- **USCI\_A\_UART\_NO\_PARITY** [Default]

- USCI\_A\_UART\_ODD\_PARITY
- USCI\_A\_UART\_EVEN\_PARITY

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

- USCI\_A\_UART\_MSB\_FIRST
- USCI\_A\_UART\_LSB\_FIRST [Default]

**numberOfStopBits** indicates one/two STOP bits Valid values are:

- USCI\_A\_UART\_ONE\_STOP\_BIT [Default]
- USCI\_A\_UART\_TWO\_STOP\_BITS

**uartMode** selects the mode of operation Valid values are:

- USCI\_A\_UART\_MODE [Default]
- USCI\_A\_UART\_IDLE\_LINE\_MULTI\_PROCESSOR\_MODE
- USCI\_A\_UART\_ADDRESS\_BIT\_MULTI\_PROCESSOR\_MODE
- USCI\_A\_UART\_AUTOMATIC\_BAUDRATE\_DETECTION\_MODE

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

- USCI\_A\_UART\_OVERSAMPLING\_BAUDRATE\_GENERATION
- USCI\_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

### 37.2.2.10 USCI\_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	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint8_t
USCI_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 USCI\_A\_UART module.

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

- USCI\_A\_UART\_LISTEN\_ENABLE
- USCI\_A\_UART\_FRAMING\_ERROR
- USCI\_A\_UART\_OVERRUN\_ERROR
- USCI\_A\_UART\_PARITY\_ERROR
- USCI\_A\_UART\_BREAK\_DETECT
- USCI\_A\_UART\_RECEIVE\_ERROR

- USCI\_A\_UART\_ADDRESS\_RECEIVED
- USCI\_A\_UART\_IDLELINE
- USCI\_A\_UART\_BUSY

Modified bits of **UCAxSTAT** register.

**Returns:**

Logical OR of any of the following:

- USCI\_A\_UART\_LISTEN\_ENABLE
  - USCI\_A\_UART\_FRAMING\_ERROR
  - USCI\_A\_UART\_OVERRUN\_ERROR
  - USCI\_A\_UART\_PARITY\_ERROR
  - USCI\_A\_UART\_BREAK\_DETECT
  - USCI\_A\_UART\_RECEIVE\_ERROR
  - USCI\_A\_UART\_ADDRESS\_RECEIVED
  - USCI\_A\_UART\_IDLELINE
  - USCI\_A\_UART\_BUSY
- indicating the status of the masked interrupt flags

### 37.2.2.11 USCI\_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	32
CCS 4.2.1	Size	18
CCS 4.2.1	Speed	18
IAR 5.51.6	None	32
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

**Prototype:**

```
uint8_t
USCI_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 USCI\_A\_UART module.

Modified bits of **UCAxRXBUF** register.

**Returns:**

Returns the byte received from by the UART module, cast as an uint8\_t.

### 37.2.2.12 USCI\_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	36
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_A_UART_resetDormant (uint32_t baseAddress)
```

**Description:**

Not dormant. All received characters set UCRXIFG.

**Parameters:**

**baseAddress** is the base address of the USCI\_A\_UART module.

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

**Returns:**

None

### 37.2.2.13 USCI\_A\_UART\_setDormant

Sets the UART module in 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	34
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_A_UART_setDormant (uint32_t baseAddress)
```

**Description:**

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.

**Parameters:**

**baseAddress** is the base address of the USCI\_A\_UART module.

Modified bits of **UCAxCTL1** register.

**Returns:**  
None

### 37.2.2.14 USCI\_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	36
CCS 4.2.1	Size	10
CCS 4.2.1	Speed	10
IAR 5.51.6	None	18
IAR 5.51.6	Size	4
IAR 5.51.6	Speed	4
MSPGCC 4.8.0	None	60
MSPGCC 4.8.0	Size	18
MSPGCC 4.8.0	Speed	18

**Prototype:**

```
void
USCI_A_UART_transmitAddress(uint32_t baseAddress,
                           uint8_t transmitAddress)
```

**Parameters:**

**baseAddress** is the base address of the USCI\_A\_UART module.  
**transmitAddress** is the next byte to be transmitted

**Description:**

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

**Returns:**  
None

### 37.2.2.15 USCI\_A\_UART\_transmitBreak

Transmit break.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	62
CCS 4.2.1	Size	44
CCS 4.2.1	Speed	44
IAR 5.51.6	None	72
IAR 5.51.6	Size	38
IAR 5.51.6	Speed	64
MSPGCC 4.8.0	None	118
MSPGCC 4.8.0	Size	56
MSPGCC 4.8.0	Speed	62

**Prototype:**

```
void
USCI_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, USCI\_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 USCI\_A\_UART module.

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

**Returns:**

None

### 37.2.2.16 USCI\_A\_UART\_transmitData

Transmits a byte from the UART Module.

**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	36
IAR 5.51.6	Size	6
IAR 5.51.6	Speed	26
MSPGCC 4.8.0	None	78
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

**Prototype:**

```
void
USCI_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 USCI\_A\_UART module.

**transmitData** data to be transmitted from the UART module

Modified bits of **UCAxTXBUF** register.

**Returns:**

None

## 37.3 Programming Example

The following example shows how to use the USCI\_A\_UART API to initialize the USCI\_A\_UART, transmit characters, and receive characters.

```
if ( STATUS_FAIL == USCI_A_UART_init (USCI_A0_BASE,
                                     USCI_A_UART_CLOCKSOURCE_SMCLK,
                                     UCS_getSMCLK(UCS_BASE),
                                     BAUD_RATE,
```

```

        USCI_A_UART_NO_PARITY,
        USCI_A_UART_LSB_FIRST,
        USCI_A_UART_ONE_STOP_BIT,
        USCI_A_UART_MODE,
        USCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION ) )

    {
        return;
    }

    //Enable USCI_A_UART module for operation
    USCI_A_UART_enable (USCI_A0_BASE);

    //Enable Receive Interrupt
    USCI_A_UART_enableInterrupt (USCI_A0_BASE,
                                UCRXIE);

    //Transmit data
    USCI_A_UART_transmitData(USCI_A0_BASE,
                             transmitData++
                             );

    // Enter LPM3, interrupts enabled
    __bis_SR_register(LPM3_bits + GIE);
    __no_operation();
}

//*****
//
// This is the USCI_A0 interrupt vector service routine.
//
//*****
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    switch(__even_in_range(UCA0IV,4))
    {
        // Vector 2 - RXIFG
        case 2:
            // Echo back RXed character, confirm TX buffer is ready first

            // USCI_A0 TX buffer ready?
            while (!USCI_A_UART_interruptStatus(USCI_A0_BASE,
                                                UCTXIFG)
                );

            //Receive echoed data
            receivedData = USCI_A_UART_receiveData(USCI_A0_BASE);

            //Transmit next data
            USCI_A_UART_transmitData(USCI_A0_BASE,
                                     transmitData++
                                     );

            break;
        default: break;
    }
}

```



## 38 USCI Synchronous Peripheral Interface (USCI\_A\_SPI)

Introduction .....	477
API Functions .....	477
Programming Example .....	488

### 38.1 Introduction

The Serial Peripheral Interface Bus or USCI\_A\_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 3-wire USCI\_A\_SPI communication

The USCI\_A\_SPI module can be configured as either a master or a slave device.

The USCI\_A\_SPI module also includes a programmable bit rate clock divider and prescaler to generate the output serial clock derived from the SSI module's input clock.

This driver is contained in `usci_a_spi.c`, with `usci_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 with 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	608
CCS 4.2.1	Size	254
CCS 4.2.1	Speed	252
IAR 5.51.6	None	374
IAR 5.51.6	Size	286
IAR 5.51.6	Speed	286
MSPGCC 4.8.0	None	1138
MSPGCC 4.8.0	Size	290
MSPGCC 4.8.0	Speed	290

### 38.2 API Functions

#### Functions

- void [USCI\\_A\\_SPI\\_changeClockPhasePolarity](#) (uint32\_t baseAddress, uint8\_t clockPhase, uint8\_t clockPolarity)
- void [USCI\\_A\\_SPI\\_clearInterruptFlag](#) (uint32\_t baseAddress, uint8\_t mask)
- void [USCI\\_A\\_SPI\\_disable](#) (uint32\_t baseAddress)
- void [USCI\\_A\\_SPI\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- void [USCI\\_A\\_SPI\\_enable](#) (uint32\_t baseAddress)
- void [USCI\\_A\\_SPI\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- uint8\_t [USCI\\_A\\_SPI\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t mask)
- uint32\_t [USCI\\_A\\_SPI\\_getReceiveBufferAddressForDMA](#) (uint32\_t baseAddress)
- uint32\_t [USCI\\_A\\_SPI\\_getTransmitBufferAddressForDMA](#) (uint32\_t baseAddress)
- uint8\_t [USCI\\_A\\_SPI\\_isBusy](#) (uint32\_t baseAddress)

- void [USCI\\_A\\_SPI\\_masterChangeClock](#) (uint32\_t baseAddress, uint32\_t clockSourceFrequency, uint32\_t desiredSpiClock)
- bool [USCI\\_A\\_SPI\\_masterInit](#) (uint32\_t baseAddress, uint8\_t selectClockSource, uint32\_t clockSourceFrequency, uint32\_t desiredSpiClock, uint8\_t msbFirst, uint8\_t clockPhase, uint8\_t clockPolarity)
- uint8\_t [USCI\\_A\\_SPI\\_receiveData](#) (uint32\_t baseAddress)
- bool [USCI\\_A\\_SPI\\_slaveInit](#) (uint32\_t baseAddress, uint8\_t msbFirst, uint8\_t clockPhase, uint8\_t clockPolarity)
- void [USCI\\_A\\_SPI\\_transmitData](#) (uint32\_t baseAddress, uint8\_t transmitData)

## 38.2.1 Detailed Description

To use the module as a master, the user must call [USCI\\_A\\_SPI\\_masterInit\(\)](#) to configure the USCI\_A\_SPI Master. This is followed by enabling the USCI\_A\_SPI module using [USCI\\_A\\_SPI\\_enable\(\)](#). The interrupts are then enabled (if needed). It is recommended to enable the USCI\_A\_SPI module before enabling the interrupts. A data transmit is then initiated using [USCI\\_A\\_SPI\\_transmitData\(\)](#) and then when the receive flag is set, the received data is read using [USCI\\_A\\_SPI\\_receiveData\(\)](#) and this indicates that an RX/TX operation is complete.

To use the module as a slave, initialization is done using [USCI\\_A\\_SPI\\_slaveInit\(\)](#) and this is followed by enabling the module using [USCI\\_A\\_SPI\\_enable\(\)](#). Following this, the interrupts may be enabled as needed. When the receive flag is set, data is first transmitted using [USCI\\_A\\_SPI\\_transmitData\(\)](#) and this is followed by a data reception by [USCI\\_A\\_SPI\\_receiveData\(\)](#)

The USCI\_A\_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 USCI\_A\_SPI module are managed by

- [USCI\\_A\\_SPI\\_masterInit\(\)](#)
- [USCI\\_A\\_SPI\\_slaveInit\(\)](#)
- [USCI\\_A\\_SPI\\_disable\(\)](#)
- [USCI\\_A\\_SPI\\_enable\(\)](#)
- [USCI\\_A\\_SPI\\_masterChangeClock\(\)](#)
- [USCI\\_A\\_SPI\\_isBusy\(\)](#)

Data handling is done by

- [USCI\\_A\\_SPI\\_transmitData\(\)](#)
- [USCI\\_A\\_SPI\\_receiveData\(\)](#)

Interrupts from the USCI\_A\_SPI module are managed using

- [USCI\\_A\\_SPI\\_disableInterrupt\(\)](#)
- [USCI\\_A\\_SPI\\_enableInterrupt\(\)](#)
- [USCI\\_A\\_SPI\\_getInterruptStatus\(\)](#)
- [USCI\\_A\\_SPI\\_clearInterruptFlag\(\)](#)

DMA related

- [USCI\\_A\\_SPI\\_getReceiveBufferAddressForDMA\(\)](#)
- [USCI\\_A\\_SPI\\_getTransmitBufferAddressForDMA\(\)](#)

## 38.2.2 Function Documentation

### 38.2.2.1 USCI\_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	64
CCS 4.2.1	Size	26
CCS 4.2.1	Speed	26
IAR 5.51.6	None	40
IAR 5.51.6	Size	30
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	154
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

**Prototype:**

```
void
USCI_A_SPI_changeClockPhasePolarity(uint32_t baseAddress,
                                     uint8_t clockPhase,
                                     uint8_t clockPolarity)
```

**Parameters:**

**baseAddress** is the base address of the I2C Master module.

**clockPhase** is clock phase select. Valid values are:

- USCI\_A\_SPI\_PHASE\_DATA\_CHANGED\_ONFIRST\_CAPTURED\_ON\_NEXT [Default]
- USCI\_A\_SPI\_PHASE\_DATA\_CAPTURED\_ONFIRST\_CHANGED\_ON\_NEXT

**clockPolarity** Valid values are:

- USCI\_A\_SPI\_CLOCKPOLARITY\_INACTIVITY\_HIGH
- USCI\_A\_SPI\_CLOCKPOLARITY\_INACTIVITY\_LOW [Default]

**Description:**

Modified bits are **UCCKPL** and **UCCKPH** of **UCAxCTL0** register.

**Returns:**

None

### 38.2.2.2 USCI\_A\_SPI\_clearInterruptFlag

Clears the selected SPI interrupt status flag.

**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	72
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

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

**Parameters:**

**baseAddress** is the base address of the SPI module.

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

- USCI\_A\_SPI\_TRANSMIT\_INTERRUPT
- USCI\_A\_SPI\_RECEIVE\_INTERRUPT

**Description:**

Modified bits of **UCAxIFG** register.

**Returns:**

None

### 38.2.2.3 USCI\_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	34
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_A_SPI_disable(uint32_t baseAddress)
```

**Description:**

This will disable operation of the SPI block.

**Parameters:**

***baseAddress*** is the base address of the USCI SPI module.

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

**Returns:**

None

### 38.2.2.4 USCI\_A\_SPI\_disableInterrupt

Disables individual SPI 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	72
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_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 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:

- USCI\_A\_SPI\_TRANSMIT\_INTERRUPT
- USCI\_A\_SPI\_RECEIVE\_INTERRUPT

Modified bits of **UCAxIE** register.

**Returns:**

None

### 38.2.2.5 USCI\_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	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
USCI_A_SPI_enable(uint32_t baseAddress)
```

**Description:**

This will enable operation of the SPI block.

**Parameters:**

**baseAddress** is the base address of the USCI SPI module.

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

**Returns:**

None

### 38.2.2.6 USCI\_A\_SPI\_enableInterrupt

Enables individual SPI 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	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_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 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:

- USCI\_A\_SPI\_TRANSMIT\_INTERRUPT
- USCI\_A\_SPI\_RECEIVE\_INTERRUPT

Modified bits of UCAXIE register.

**Returns:**

None

### 38.2.2.7 uint8\_t USCI\_A\_SPI\_getInterruptStatus (uint32\_t baseAddress, uint8\_t mask)

Gets the current SPI interrupt status.

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

**Code Metrics:**

Compiler	Optimization	Code Size
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	8
MSPGCC 4.8.0	Speed	8

**Parameters:**

**baseAddress** is the base address of the SPI module.

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

- USCI\_A\_SPI\_TRANSMIT\_INTERRUPT
- USCI\_A\_SPI\_RECEIVE\_INTERRUPT

**Returns:**

The current interrupt status as the mask of the set flags Return Logical OR of any of the following:

- **USCI\_A\_SPI\_TRANSMIT\_INTERRUPT**
  - **USCI\_A\_SPI\_RECEIVE\_INTERRUPT**
- indicating the status of the masked interrupts

### 38.2.2.8 USCI\_A\_SPI\_getReceiveBufferAddressForDMA

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
USCI_A_SPI_getReceiveBufferAddressForDMA(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 SPI module.

#### Returns:

the address of the RX Buffer

### 38.2.2.9 USCI\_A\_SPI\_getTransmitBufferAddressForDMA

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
USCI_A_SPI_getTransmitBufferAddressForDMA(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 SPI module.

**Returns:**

the address of the TX Buffer

### 38.2.2.10 USCI\_A\_SPI\_isBusy

Indicates whether or not the SPI bus is busy.

**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	34
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint8_t
USCI_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 SPI module.

**Returns:**

USCI\_A\_SPI\_BUSY if the SPI module transmitting or receiving is busy; otherwise, returns USCI\_A\_SPI\_NOT\_BUSY. Return one of the following:

- **USCI\_A\_SPI\_BUSY**
- **USCI\_A\_SPI\_NOT\_BUSY**  
indicating if the USCI\_A\_SPI is busy

### 38.2.2.11 USCI\_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	112
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

**Prototype:**

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

**Parameters:**

**baseAddress** is the base address of the I2C Master module.  
**clockSourceFrequency** is the frequency of the selected clock source  
**desiredSpiClock** is the desired clock rate for SPI communication

**Description:**

Modified bits of **UCAxBRW** register.

**Returns:**

None

### 38.2.2.12 USCI\_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	124
IAR 5.51.6	Size	96
IAR 5.51.6	Speed	96
MSPGCC 4.8.0	None	254
MSPGCC 4.8.0	Size	108
MSPGCC 4.8.0	Speed	108

**Prototype:**

```
bool
USCI_A_SPI_masterInit(uint32_t baseAddress,
                      uint8_t selectClockSource,
                      uint32_t clockSourceFrequency,
                      uint32_t desiredSpiClock,
                      uint8_t msbFirst,
                      uint8_t clockPhase,
                      uint8_t clockPolarity)
```

**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 [USCI\\_A\\_SPI\\_enable\(\)](#)

**Parameters:**

**baseAddress** is the base address of the I2C Master module.

**selectClockSource** selects Clock source. Valid values are:

- USCI\_A\_SPI\_CLOCKSOURCE\_ACLK
- USCI\_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:

- USCI\_A\_SPI\_MSB\_FIRST
- USCI\_A\_SPI\_LSB\_FIRST [Default]

**clockPhase** is clock phase select. Valid values are:

- USCI\_A\_SPI\_PHASE\_DATA\_CHANGED\_ONFIRST\_CAPTURED\_ON\_NEXT [Default]
- USCI\_A\_SPI\_PHASE\_DATA\_CAPTURED\_ONFIRST\_CHANGED\_ON\_NEXT

**clockPolarity** Valid values are:

- USCI\_A\_SPI\_CLOCKPOLARITY\_INACTIVITY\_HIGH
- USCI\_A\_SPI\_CLOCKPOLARITY\_INACTIVITY\_LOW [Default]

Modified bits are **UCCKPH**, **UCCKPL**, **UC7BIT** and **UCMSB** of **UCAxCTL0** register; bits **UCSSELx** and **UCSWRST** of **UCAxCTL1** register.

**Returns:**

STATUS\_SUCCESS

### 38.2.2.13 USCI\_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	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	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint8_t
USCI_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 SPI module.

**Returns:**

Returns the byte received from by the SPI module, cast as an uint8\_t.

### 38.2.2.14 USCI\_A\_SPI\_slaveInit

Initializes the SPI Slave block.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	64
CCS 4.2.1	Size	30
CCS 4.2.1	Speed	28
IAR 5.51.6	None	46
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	144
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

#### Prototype:

```
bool
USCI_A_SPI_slaveInit(uint32_t baseAddress,
                    uint8_t msbFirst,
                    uint8_t clockPhase,
                    uint8_t clockPolarity)
```

#### 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 [USCI\\_A\\_SPI\\_enable\(\)](#)

#### Parameters:

**baseAddress** is the base address of the SPI Slave module.

**msbFirst** controls the direction of the receive and transmit shift register. Valid values are:

- **USCI\_A\_SPI\_MSB\_FIRST**
- **USCI\_A\_SPI\_LSB\_FIRST** [Default]

**clockPhase** is clock phase select. Valid values are:

- **USCI\_A\_SPI\_PHASE\_DATA\_CHANGED\_ONFIRST\_CAPTURED\_ON\_NEXT** [Default]
- **USCI\_A\_SPI\_PHASE\_DATA\_CAPTURED\_ONFIRST\_CHANGED\_ON\_NEXT**

**clockPolarity** Valid values are:

- **USCI\_A\_SPI\_CLOCKPOLARITY\_INACTIVITY\_HIGH**
- **USCI\_A\_SPI\_CLOCKPOLARITY\_INACTIVITY\_LOW** [Default]

Modified bits are **UCMSB**, **UCMST**, **UC7BIT**, **UCCKPL**, **UCCKPH** and **UCMODE** of **UCAxCTL0** register; bits **UCSWRST** of **UCAxCTL1** register.

#### Returns:

STATUS\_SUCCESS

### 38.2.2.15 USCI\_A\_SPI\_transmitData

Transmits a byte from the SPI Module.

#### 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	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

**Prototype:**

```
void
USCI_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 SPI module.

**transmitData** data to be transmitted from the SPI module

**Returns:**

None

## 38.3 Programming Example

The following example shows how to use the USCI\_A\_SPI API to configure the USCI\_A\_SPI module as a master device, and how to do a simple send of data.

```
//Initialize Master
returnValue = USCI_A_SPI_masterInit(USCI_A0_BASE, SMCLK, CLK_getSMClk(),
                                     USCI_SPICLK, MSB_FIRST,
                                     CLOCK_POLARITY_INACTIVITYHIGH
                                     );

if (STATUS_FAIL == returnValue)
{
    return;
}

//Enable USCI_A_SPI module
USCI_A_SPI_enable(USCI_A0_BASE);

//Enable Receive interrupt
USCI_A_SPI_enableInterrupt(USCI_A0_BASE, UCRXIE);

//Configure port pins to reset slave

// Wait for slave to initialize
__delay_cycles(100);

// Initialize data values
transmitData = 0x00;

// USCI_A0 TX buffer ready?
while (!USCI_A_SPI_interruptStatus(USCI_A0_BASE, UCTXIFG));

//Transmit Data to slave
```

```
USCI_A_SPI_transmitData(USCI_A0_BASE, transmitData);

// CPU off, enable interrupts
__bis_SR_register(LPM0_bits + GIE);
}

//*****
//
// This is the USCI_B0 interrupt vector service routine.
//
//*****
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
    switch(__even_in_range(UCA0IV,4))
    {
        // Vector 2 - RXIFG
        case 2:
            // USCI_A0 TX buffer ready?
            while (!USCI_A_SPI_interruptStatus(USCI_A0_BASE, UCTXIFG));

            receiveData = USCI_A_SPI_receiveData(USCI_A0_BASE);

            // Increment data
            transmitData++;

            // Send next value
            USCI_A_SPI_transmitData(USCI_A0_BASE, transmitData);

            //Delay between transmissions for slave to process information
            __delay_cycles(40);

            break;
            default: break;
        }
    }
}
```

## 39 USCI Synchronous Peripheral Interface (USCI\_B\_SPI)

Introduction .....	490
API Functions .....	490
Programming Example .....	501

### 39.1 Introduction

The Serial Peripheral Interface Bus or USCI\_B\_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 3-wire USCI\_B\_SPI communication

The USCI\_B\_SPI module can be configured as either a master or a slave device.

The USCI\_B\_SPI module also includes a programmable bit rate clock divider and prescaler to generate the output serial clock derived from the SSI module's input clock.

This driver is contained in `usci_b_spi.c`, with `usci_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 with 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	250
CCS 4.2.1	Speed	248
IAR 5.51.6	None	366
IAR 5.51.6	Size	278
IAR 5.51.6	Speed	278
MSPGCC 4.8.0	None	1128
MSPGCC 4.8.0	Size	282
MSPGCC 4.8.0	Speed	282

### 39.2 API Functions

#### Functions

- void [USCI\\_B\\_SPI\\_changeClockPhasePolarity](#) (uint32\_t baseAddress, uint8\_t clockPhase, uint8\_t clockPolarity)
- void [USCI\\_B\\_SPI\\_clearInterruptFlag](#) (uint32\_t baseAddress, uint8\_t mask)
- void [USCI\\_B\\_SPI\\_disable](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_SPI\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- void [USCI\\_B\\_SPI\\_enable](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_SPI\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- uint8\_t [USCI\\_B\\_SPI\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t mask)
- uint32\_t [USCI\\_B\\_SPI\\_getReceiveBufferAddressForDMA](#) (uint32\_t baseAddress)
- uint32\_t [USCI\\_B\\_SPI\\_getTransmitBufferAddressForDMA](#) (uint32\_t baseAddress)
- uint8\_t [USCI\\_B\\_SPI\\_isBusy](#) (uint32\_t baseAddress)

- void [USCI\\_B\\_SPI\\_masterChangeClock](#) (uint32\_t baseAddress, uint32\_t clockSourceFrequency, uint32\_t desiredSpiClock)
- bool [USCI\\_B\\_SPI\\_masterInit](#) (uint32\_t baseAddress, uint8\_t selectClockSource, uint32\_t clockSourceFrequency, uint32\_t desiredSpiClock, uint8\_t msbFirst, uint8\_t clockPhase, uint8\_t clockPolarity)
- uint8\_t [USCI\\_B\\_SPI\\_receiveData](#) (uint32\_t baseAddress)
- bool [USCI\\_B\\_SPI\\_slaveInit](#) (uint32\_t baseAddress, uint8\_t msbFirst, uint8\_t clockPhase, uint8\_t clockPolarity)
- void [USCI\\_B\\_SPI\\_transmitData](#) (uint32\_t baseAddress, uint8\_t transmitData)

## 39.2.1 Detailed Description

To use the module as a master, the user must call [USCI\\_B\\_SPI\\_masterInit\(\)](#) to configure the USCI\_B\_SPI Master. This is followed by enabling the USCI\_B\_SPI module using [USCI\\_B\\_SPI\\_enable\(\)](#). The interrupts are then enabled (if needed). It is recommended to enable the USCI\_B\_SPI module before enabling the interrupts. A data transmit is then initiated using [USCI\\_B\\_SPI\\_transmitData\(\)](#) and then when the receive flag is set, the received data is read using [USCI\\_B\\_SPI\\_receiveData\(\)](#) and this indicates that an RX/TX operation is complete.

To use the module as a slave, initialization is done using [USCI\\_B\\_SPI\\_slaveInit\(\)](#) and this is followed by enabling the module using [USCI\\_B\\_SPI\\_enable\(\)](#). Following this, the interrupts may be enabled as needed. When the receive flag is set, data is first transmitted using [USCI\\_B\\_SPI\\_transmitData\(\)](#) and this is followed by a data reception by [USCI\\_B\\_SPI\\_receiveData\(\)](#)

The USCI\_B\_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 USCI\_B\_SPI module are managed by

- [USCI\\_B\\_SPI\\_masterInit\(\)](#)
- [USCI\\_B\\_SPI\\_slaveInit\(\)](#)
- [USCI\\_B\\_SPI\\_disable\(\)](#)
- [USCI\\_B\\_SPI\\_enable\(\)](#)
- [USCI\\_B\\_SPI\\_masterChangeClock\(\)](#)
- [USCI\\_B\\_SPI\\_isBusy\(\)](#)

Data handling is done by

- [USCI\\_B\\_SPI\\_transmitData\(\)](#)
- [USCI\\_B\\_SPI\\_receiveData\(\)](#)

Interrupts from the USCI\_B\_SPI module are managed using

- [USCI\\_B\\_SPI\\_disableInterrupt\(\)](#)
- [USCI\\_B\\_SPI\\_enableInterrupt\(\)](#)
- [USCI\\_B\\_SPI\\_getInterruptStatus\(\)](#)
- [USCI\\_B\\_SPI\\_clearInterruptFlag\(\)](#)

DMA related

- [USCI\\_B\\_SPI\\_getReceiveBufferAddressForDMA\(\)](#)
- [USCI\\_B\\_SPI\\_getTransmitBufferAddressForDMA\(\)](#)

## 39.2.2 Function Documentation

### 39.2.2.1 USCI\_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	64
CCS 4.2.1	Size	26
CCS 4.2.1	Speed	26
IAR 5.51.6	None	40
IAR 5.51.6	Size	30
IAR 5.51.6	Speed	30
MSPGCC 4.8.0	None	154
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

**Prototype:**

```
void
USCI_B_SPI_changeClockPhasePolarity(uint32_t baseAddress,
                                     uint8_t clockPhase,
                                     uint8_t clockPolarity)
```

**Parameters:**

**baseAddress** is the base address of the I2C Master module.

**clockPhase** is clock phase select. Valid values are:

- USCI\_B\_SPI\_PHASE\_DATA\_CHANGED\_ONFIRST\_CAPTURED\_ON\_NEXT [Default]
- USCI\_B\_SPI\_PHASE\_DATA\_CAPTURED\_ONFIRST\_CHANGED\_ON\_NEXT

**clockPolarity** Valid values are:

- USCI\_B\_SPI\_CLOCKPOLARITY\_INACTIVITY\_HIGH
- USCI\_B\_SPI\_CLOCKPOLARITY\_INACTIVITY\_LOW [Default]

**Description:**

Modified bits are **UCCKPL** and **UCCKPH** of **UCAxCTL0** register.

**Returns:**

None

### 39.2.2.2 USCI\_B\_SPI\_clearInterruptFlag

Clears the selected SPI interrupt status flag.

**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	72
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

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

**Parameters:**

**baseAddress** is the base address of the SPI module.

**mask** is the masked interrupt flag to be cleared. Valid values are:



- USCI\_B\_SPI\_TRANSMIT\_INTERRUPT
- USCI\_B\_SPI\_RECEIVE\_INTERRUPT

**Description:**

Modified bits of **UCBxIFG** register.

**Returns:**

None

### 39.2.2.3 USCI\_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	34
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_B_SPI_disable(uint32_t baseAddress)
```

**Description:**

This will disable operation of the SPI block.

**Parameters:**

***baseAddress*** is the base address of the USCI SPI module.

Modified bits are **UCSWRST** of **UCBxCTL1** register.

**Returns:**

None

### 39.2.2.4 USCI\_B\_SPI\_disableInterrupt

Disables individual SPI 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	72
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_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 SPI module.

**mask** is the bit mask of the interrupt sources to be disabled. Valid values are:

- USCI\_B\_SPI\_TRANSMIT\_INTERRUPT
- USCI\_B\_SPI\_RECEIVE\_INTERRUPT

Modified bits of **UCBxIE** register.

**Returns:**

None

### 39.2.2.5 USCI\_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	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
USCI_B_SPI_enable (uint32_t baseAddress)
```

**Description:**

This will enable operation of the SPI block.

**Parameters:**

**baseAddress** is the base address of the USCI SPI module.

Modified bits are **UCSWRST** of **UCBxCTL1** register.

**Returns:**

None

### 39.2.2.6 USCI\_B\_SPI\_enableInterrupt

Enables individual SPI 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	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	56
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_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 SPI module.

**mask** is the bit mask of the interrupt sources to be enabled. Valid values are:

- USCI\_B\_SPI\_TRANSMIT\_INTERRUPT
- USCI\_B\_SPI\_RECEIVE\_INTERRUPT

Modified bits of UCBxIE register.

**Returns:**

None

### 39.2.2.7 uint8\_t USCI\_B\_SPI\_getInterruptStatus (uint32\_t baseAddress, uint8\_t mask)

Gets the current SPI interrupt status.

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

**Code Metrics:**

Compiler	Optimization	Code Size
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	8
MSPGCC 4.8.0	Speed	8

**Parameters:**

**baseAddress** is the base address of the SPI module.

**mask** is the masked interrupt flag status to be returned. Valid values are:

- USCI\_B\_SPI\_TRANSMIT\_INTERRUPT
- USCI\_B\_SPI\_RECEIVE\_INTERRUPT

**Returns:**

The current interrupt status as the mask of the set flags Return Logical OR of any of the following:

- **USCI\_B\_SPI\_TRANSMIT\_INTERRUPT**
  - **USCI\_B\_SPI\_RECEIVE\_INTERRUPT**
- indicating the status of the masked interrupts

### 39.2.2.8 USCI\_B\_SPI\_getReceiveBufferAddressForDMA

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
USCI_B_SPI_getReceiveBufferAddressForDMA(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 SPI module.

#### Returns:

The address of the SPI RX buffer

### 39.2.2.9 USCI\_B\_SPI\_getTransmitBufferAddressForDMA

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
USCI_B_SPI_getTransmitBufferAddressForDMA(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 SPI module.

**Returns:**

The address of the SPI TX buffer

### 39.2.2.10 USCI\_B\_SPI\_isBusy

Indicates whether or not the SPI bus is busy.

**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	34
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint8_t
USCI_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 SPI module.

**Returns:**

USCI\_B\_SPI\_BUSY if the SPI module transmitting or receiving is busy; otherwise, returns USCI\_B\_SPI\_NOT\_BUSY. Return one of the following:

- **USCI\_B\_SPI\_BUSY**
- **USCI\_B\_SPI\_NOT\_BUSY**  
indicating if the USCI\_B\_SPI is busy

### 39.2.2.11 USCI\_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	112
MSPGCC 4.8.0	Size	46
MSPGCC 4.8.0	Speed	46

**Prototype:**

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

**Parameters:**

**baseAddress** is the base address of the I2C Master module.  
**clockSourceFrequency** is the frequency of the selected clock source  
**desiredSpiClock** is the desired clock rate for SPI communication

**Description:**

Modified bits of **UCAxBRW** register.

**Returns:**

None

### 39.2.2.12 USCI\_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	116
IAR 5.51.6	Size	88
IAR 5.51.6	Speed	88
MSPGCC 4.8.0	None	244
MSPGCC 4.8.0	Size	100
MSPGCC 4.8.0	Speed	100

**Prototype:**

```
bool
USCI_B_SPI_masterInit(uint32_t baseAddress,
                      uint8_t selectClockSource,
                      uint32_t clockSourceFrequency,
                      uint32_t desiredSpiClock,
                      uint8_t msbFirst,
                      uint8_t clockPhase,
                      uint8_t clockPolarity)
```

**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 [USCI\\_B\\_SPI\\_enable\(\)](#)

**Parameters:**

**baseAddress** is the base address of the I2C Master module.

**selectClockSource** selects Clock source. Valid values are:

- USCI\_B\_SPI\_CLOCKSOURCE\_ACLK
- USCI\_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:

- USCI\_B\_SPI\_MSB\_FIRST
- USCI\_B\_SPI\_LSB\_FIRST [Default]

**clockPhase** is clock phase select. Valid values are:

- USCI\_B\_SPI\_PHASE\_DATA\_CHANGED\_ONFIRST\_CAPTURED\_ON\_NEXT [Default]
- USCI\_B\_SPI\_PHASE\_DATA\_CAPTURED\_ONFIRST\_CHANGED\_ON\_NEXT

**clockPolarity** Valid values are:

- USCI\_B\_SPI\_CLOCKPOLARITY\_INACTIVITY\_HIGH
- USCI\_B\_SPI\_CLOCKPOLARITY\_INACTIVITY\_LOW [Default]

Modified bits are **UCSSSELx** and **UCSWRST** of **UCBxCTL1** register; bits **UCCKPH**, **UCCKPL**, **UC7BIT** and **UCMSB** of **UCBxCTL0** register.

**Returns:**

STATUS\_SUCCESS

### 39.2.2.13 USCI\_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	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	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint8_t
USCI_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 SPI module.

**Returns:**

Returns the byte received from by the SPI module, cast as an uint8\_t.

## 39.2.2.14 USCI\_B\_SPI\_slaveInit

Initializes the SPI Slave block.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	64
CCS 4.2.1	Size	30
CCS 4.2.1	Speed	28
IAR 5.51.6	None	46
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	144
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26

**Prototype:**

```
bool
USCI_B_SPI_slaveInit(uint32_t baseAddress,
                     uint8_t msbFirst,
                     uint8_t clockPhase,
                     uint8_t clockPolarity)
```

**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 [USCI\\_B\\_SPI\\_enable\(\)](#)

**Parameters:**

**baseAddress** is the base address of the SPI Slave module.

**msbFirst** controls the direction of the receive and transmit shift register. Valid values are:

- **USCI\_B\_SPI\_MSB\_FIRST**
- **USCI\_B\_SPI\_LSB\_FIRST** [Default]

**clockPhase** is clock phase select. Valid values are:

- **USCI\_B\_SPI\_PHASE\_DATA\_CHANGED\_ONFIRST\_CAPTURED\_ON\_NEXT** [Default]
- **USCI\_B\_SPI\_PHASE\_DATA\_CAPTURED\_ONFIRST\_CHANGED\_ON\_NEXT**

**clockPolarity** Valid values are:

- **USCI\_B\_SPI\_CLOCKPOLARITY\_INACTIVITY\_HIGH**
- **USCI\_B\_SPI\_CLOCKPOLARITY\_INACTIVITY\_LOW** [Default]

Modified bits are **UCSWRST** of **UCBxCTL1** register; bits **UCMSB**, **UCMST**, **UC7BIT**, **UCCKPL**, **UCCKPH** and **UCMODE** of **UCBxCTL0** register.

**Returns:**

STATUS\_SUCCESS

## 39.2.2.15 USCI\_B\_SPI\_transmitData

Transmits a byte from the SPI Module.

**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	12
IAR 5.51.6	Size	10
IAR 5.51.6	Speed	10
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

**Prototype:**

```
void
USCI_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 SPI module.

**transmitData** data to be transmitted from the SPI module

**Returns:**

None

## 39.3 Programming Example

The following example shows how to use the USCI\_B\_SPI API to configure the USCI\_B\_SPI module as a master device, and how to do a simple send of data.

```
//Initialize Master
returnValue = USCI_B_SPI_masterInit(USCI_A0_BASE, SMCLK, CLK_getSMClk(),
                                    USCI_SPICLK, MSB_FIRST,
                                    CLOCK_POLARITY_INACTIVITYHIGH
                                    );

if (STATUS_FAIL == returnValue)
{
    return;
}

//Enable USCI_B_SPI module
USCI_B_SPI_enable(USCI_A0_BASE);

//Enable Receive interrupt
USCI_B_SPI_enableInterrupt(USCI_A0_BASE, UCRXIE);

//Configure port pins to reset slave

// Wait for slave to initialize
__delay_cycles(100);

// Initialize data values
transmitData = 0x00;

// USCI_A0 TX buffer ready?
while (!USCI_B_SPI_interruptStatus(USCI_A0_BASE, UCTXIFG));

//Transmit Data to slave
```

```
USCI_B_SPI_transmitData(USCI_A0_BASE, transmitData);

// CPU off, enable interrupts
__bis_SR_register(LPM0_bits + GIE);
}

//*****
//
// This is the USCI_B0 interrupt vector service routine.
//
//*****
#pragma vector=USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
{
    switch(__even_in_range(UCA0IV,4))
    {
        // Vector 2 - RXIFG
        case 2:
            // USCI_A0 TX buffer ready?
            while (!USCI_B_SPI_interruptStatus(USCI_A0_BASE, UCTXIFG));

            receiveData = USCI_B_SPI_receiveData(USCI_A0_BASE);

            // Increment data
            transmitData++;

            // Send next value
            USCI_B_SPI_transmitData(USCI_A0_BASE, transmitData);

            //Delay between transmissions for slave to process information
            __delay_cycles(40);

            break;
            default: break;
        }
    }
}
```

## 40 USCI Inter-Integrated Circuit (USCI\_B\_I2C)

Introduction .....	503
API Functions .....	505
Programming Example .....	529

### 40.1 Introduction

The Inter-Integrated Circuit (USCI\_B\_I2C) API provides a set of functions for using the MSP430Ware USCI\_B\_I2C modules. Functions are provided to initialize the USCI\_B\_I2C modules, to send and receive data, obtain status, and to manage interrupts for the USCI\_B\_I2C modules.

The USCI\_B\_I2C module provide the ability to communicate to other IC devices over an USCI\_B\_I2C bus. The USCI\_B\_I2C bus is specified to support devices that can both transmit and receive (write and read) data. Also, devices on the USCI\_B\_I2C bus can be designated as either a master or a slave. The MSP430Ware USCI\_B\_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. Finally, the MSP430Ware USCI\_B\_I2C modules can operate at two speeds: Standard (100 kb/s) and Fast (400 kb/s).

USCI\_B\_I2C module can generate interrupts. The USCI\_B\_I2C module configured as a master will generate interrupts when a transmit or receive operation is completed (or aborted due to an error). The USCI\_B\_I2C module configured as a slave will generate interrupts when data has been sent or requested by a master.

#### 40.1.1 Master Operations

To drive the master module, the APIs need to be invoked in the following order

- [USCI\\_B\\_I2C\\_masterInit\(\)](#)
- [USCI\\_B\\_I2C\\_setSlaveAddress\(\)](#)
- [USCI\\_B\\_I2C\\_setMode\(\)](#)
- [USCI\\_B\\_I2C\\_enable\(\)](#)
- [USCI\\_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 USCI\_B\_I2C module and configure it as a master with a call to [USCI\\_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 using [USCI\\_B\\_I2C\\_setSlaveAddress](#). Then the mode of operation (transmit or receive) is chosen using [USCI\\_B\\_I2C\\_setMode](#). The USCI\_B\_I2C module may now be enabled using [USCI\\_B\\_I2C\\_enable](#). It is recommended to enable the USCI\_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. APIs that include a time-out can be used to avoid being stuck in an infinite loop if the device is stuck waiting for an IFG flag to be set.

Master Single Byte Transmission

- [USCI\\_B\\_I2C\\_masterSendSingleByte\(\)](#)

Master Multiple Byte Transmission

- [USCI\\_B\\_I2C\\_masterMultiByteSendStart\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteSendNext\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteSendFinish\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteSendStop\(\)](#)

Master Single Byte Reception

- [USCI\\_B\\_I2C\\_masterSingleReceiveStart\(\)](#)

- [USCI\\_B\\_I2C\\_masterSingleReceive\(\)](#)

Master Multiple Byte Reception

- [USCI\\_B\\_I2C\\_masterMultiByteReceiveStart\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteReceiveNext\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteReceiveFinish\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteReceiveStop\(\)](#)

Master Single Byte Transmission with Time-out

- [USCI\\_B\\_I2C\\_masterSendSingleByteWithTimeout\(\)](#)

Master Multiple Byte Transmission with Time-out

- [USCI\\_B\\_I2C\\_masterMultiByteSendStartWithTimeout\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteSendNextWithTimeout\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteReceiveFinishWithTimeout\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteSendStopWithTimeout\(\)](#)

Master Single Byte Reception with Time-out [USCI\\_B\\_I2C\\_masterSingleReceiveStartWithTimeout\(\)](#)

For the interrupt-driven transaction, the user must register an interrupt handler for the USCI\_B\_I2C devices and enable the USCI\_B\_I2C interrupt.

## 40.1.2 Slave Operations

To drive the slave module, the APIs need to be invoked in the following order

- [USCI\\_B\\_I2C\\_slaveInit\(\)](#)
- [USCI\\_B\\_I2C\\_setMode\(\)](#)
- [USCI\\_B\\_I2C\\_enable\(\)](#)
- [USCI\\_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 [USCI\\_B\\_I2C\\_slaveInit\(\)](#) to initialize the slave module in USCI\_B\_I2C mode and set the slave address. This is followed by a call to set the mode of operation ( transmit or receive ).The USCI\_B\_I2C module may now be enabled using [USCI\\_B\\_I2C\\_enable\(\)](#) It is recommended to enable the USCI\_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.

Slave Transmission API

- [USCI\\_B\\_I2C\\_slaveDataPut\(\)](#)

Slave Reception API

- [USCI\\_B\\_I2C\\_slaveDataGet\(\)](#)

For the interrupt-driven transaction, the user must register an interrupt handler for the USCI\_B\_I2C devices and enable the USCI\_B\_I2C interrupt.

This driver is contained in `usci_b_i2c.c`, with `usci_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	1998
CCS 4.2.1	Size	958
CCS 4.2.1	Speed	958
IAR 5.51.6	None	1432
IAR 5.51.6	Size	1016
IAR 5.51.6	Speed	1174
MSPGCC 4.8.0	None	3526
MSPGCC 4.8.0	Size	1314
MSPGCC 4.8.0	Speed	1376

## 40.2 API Functions

### Functions

- void [USCI\\_B\\_I2C\\_clearInterruptFlag](#) (uint32\_t baseAddress, uint8\_t mask)
- void [USCI\\_B\\_I2C\\_disable](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_I2C\\_disableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- void [USCI\\_B\\_I2C\\_enable](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_I2C\\_enableInterrupt](#) (uint32\_t baseAddress, uint8\_t mask)
- uint8\_t [USCI\\_B\\_I2C\\_getInterruptStatus](#) (uint32\_t baseAddress, uint8\_t mask)
- uint32\_t [USCI\\_B\\_I2C\\_getReceiveBufferAddressForDMA](#) (uint32\_t baseAddress)
- uint32\_t [USCI\\_B\\_I2C\\_getTransmitBufferAddressForDMA](#) (uint32\_t baseAddress)
- uint8\_t [USCI\\_B\\_I2C\\_isBusBusy](#) (uint32\_t baseAddress)
- uint8\_t [USCI\\_B\\_I2C\\_isBusy](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_I2C\\_masterInit](#) (uint32\_t baseAddress, uint8\_t selectClockSource, uint32\_t i2cClk, uint32\_t dataRate)
- uint8\_t [USCI\\_B\\_I2C\\_masterIsStartSent](#) (uint32\_t baseAddress)
- uint8\_t [USCI\\_B\\_I2C\\_masterIsStopSent](#) (uint32\_t baseAddress)
- uint8\_t [USCI\\_B\\_I2C\\_masterMultiByteReceiveFinish](#) (uint32\_t baseAddress)
- bool [USCI\\_B\\_I2C\\_masterMultiByteReceiveFinishWithTimeout](#) (uint32\_t baseAddress, uint8\_t \*rxData, uint32\_t timeout)
- uint8\_t [USCI\\_B\\_I2C\\_masterMultiByteReceiveNext](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_I2C\\_masterMultiByteReceiveStart](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_I2C\\_masterMultiByteReceiveStop](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_I2C\\_masterMultiByteSendFinish](#) (uint32\_t baseAddress, uint8\_t txData)
- bool [USCI\\_B\\_I2C\\_masterMultiByteSendFinishWithTimeout](#) (uint32\_t baseAddress, uint8\_t txData, uint32\_t timeout)
- void [USCI\\_B\\_I2C\\_masterMultiByteSendNext](#) (uint32\_t baseAddress, uint8\_t txData)
- bool [USCI\\_B\\_I2C\\_masterMultiByteSendNextWithTimeout](#) (uint32\_t baseAddress, uint8\_t txData, uint32\_t timeout)
- void [USCI\\_B\\_I2C\\_masterMultiByteSendStart](#) (uint32\_t baseAddress, uint8\_t txData)
- bool [USCI\\_B\\_I2C\\_masterMultiByteSendStartWithTimeout](#) (uint32\_t baseAddress, uint8\_t txData, uint32\_t timeout)
- void [USCI\\_B\\_I2C\\_masterMultiByteSendStop](#) (uint32\_t baseAddress)
- bool [USCI\\_B\\_I2C\\_masterMultiByteSendStopWithTimeout](#) (uint32\_t baseAddress, uint32\_t timeout)
- void [USCI\\_B\\_I2C\\_masterSendSingleByte](#) (uint32\_t baseAddress, uint8\_t txData)
- bool [USCI\\_B\\_I2C\\_masterSendSingleByteWithTimeout](#) (uint32\_t baseAddress, uint8\_t txData, uint32\_t timeout)
- void [USCI\\_B\\_I2C\\_masterSendStart](#) (uint32\_t baseAddress)
- uint8\_t [USCI\\_B\\_I2C\\_masterSingleReceive](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_I2C\\_masterSingleReceiveStart](#) (uint32\_t baseAddress)
- bool [USCI\\_B\\_I2C\\_masterSingleReceiveStartWithTimeout](#) (uint32\_t baseAddress, uint32\_t timeout)
- void [USCI\\_B\\_I2C\\_setMode](#) (uint32\_t baseAddress, uint8\_t mode)
- void [USCI\\_B\\_I2C\\_setSlaveAddress](#) (uint32\_t baseAddress, uint8\_t slaveAddress)
- uint8\_t [USCI\\_B\\_I2C\\_slaveDataGet](#) (uint32\_t baseAddress)
- void [USCI\\_B\\_I2C\\_slaveDataPut](#) (uint32\_t baseAddress, uint8\_t transmitData)
- void [USCI\\_B\\_I2C\\_slaveInit](#) (uint32\_t baseAddress, uint8\_t slaveAddress)

## 40.2.1 Detailed Description

The USCI\_B\_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 USCI\_B\_I2C master and slave interrupts and status are handled by

- [USCI\\_B\\_I2C\\_enableInterrupt\(\)](#)
- [USCI\\_B\\_I2C\\_disableInterrupt\(\)](#)
- [USCI\\_B\\_I2C\\_clearInterruptFlag\(\)](#)
- [USCI\\_B\\_I2C\\_getInterruptStatus\(\)](#)
- [USCI\\_B\\_I2C\\_masterIsStopSent\(\)](#)
- [USCI\\_B\\_I2C\\_masterIsStartSent\(\)](#)

Status and initialization functions for the USCI\_B\_I2C modules are

- [USCI\\_B\\_I2C\\_masterInit\(\)](#)
- [USCI\\_B\\_I2C\\_enable\(\)](#)
- [USCI\\_B\\_I2C\\_disable\(\)](#)
- [USCI\\_B\\_I2C\\_isBusBusy\(\)](#)
- [USCI\\_B\\_I2C\\_isBusy\(\)](#)
- [USCI\\_B\\_I2C\\_slaveInit\(\)](#)
- [USCI\\_B\\_I2C\\_interruptStatus\(\)](#)
- [USCI\\_B\\_I2C\\_setSlaveAddress\(\)](#)
- [USCI\\_B\\_I2C\\_setMode\(\)](#)

Sending and receiving data from the USCI\_B\_I2C slave module is handled by

- [USCI\\_B\\_I2C\\_slaveDataPut\(\)](#)
- [USCI\\_B\\_I2C\\_slaveDataGet\(\)](#)

Sending and receiving data from the USCI\_B\_I2C slave module is handled by

- [USCI\\_B\\_I2C\\_masterSendSingleByte\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteSendStart\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteSendNext\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteSendFinish\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteSendStop\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteReceiveStart\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteReceiveNext\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteReceiveFinish\(\)](#)
- [USCI\\_B\\_I2C\\_masterMultiByteReceiveStop\(\)](#)
- [USCI\\_B\\_I2C\\_masterSingleReceiveStart\(\)](#)
- [USCI\\_B\\_I2C\\_masterSingleReceive\(\)](#)
- [USCI\\_B\\_I2C\\_getReceiveBufferAddressForDMA\(\)](#)
- [USCI\\_B\\_I2C\\_getTransmitBufferAddressForDMA\(\)](#)

DMA related

- [USCI\\_B\\_I2C\\_getReceiveBufferAddressForDMA\(\)](#)
- [USCI\\_B\\_I2C\\_getTransmitBufferAddressForDMA\(\)](#)

## 40.2.2 Function Documentation

### 40.2.2.1 USCI\_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	72
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_B_I2C_clearInterruptFlag(uint32_t baseAddress,
                              uint8_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 Slave module.

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

- **USCI\_B\_I2C\_STOP\_INTERRUPT** - STOP condition interrupt
- **USCI\_B\_I2C\_START\_INTERRUPT** - START condition interrupt
- **USCI\_B\_I2C\_RECEIVE\_INTERRUPT** - Receive interrupt
- **USCI\_B\_I2C\_TRANSMIT\_INTERRUPT** - Transmit interrupt
- **USCI\_B\_I2C\_NAK\_INTERRUPT** - Not-acknowledge interrupt
- **USCI\_B\_I2C\_ARBITRATIONLOST\_INTERRUPT** - Arbitration lost interrupt

Modified bits of **UCBxIFG** register.

**Returns:**

None

### 40.2.2.2 USCI\_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	34
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_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 **UCBxCTL1** register.

**Returns:**

None

### 40.2.2.3 USCI\_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	72
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_B_I2C_disableInterrupt(uint32_t baseAddress,
                             uint8_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:

- **USCI\_B\_I2C\_STOP\_INTERRUPT** - STOP condition interrupt
- **USCI\_B\_I2C\_START\_INTERRUPT** - START condition interrupt
- **USCI\_B\_I2C\_RECEIVE\_INTERRUPT** - Receive interrupt
- **USCI\_B\_I2C\_TRANSMIT\_INTERRUPT** - Transmit interrupt
- **USCI\_B\_I2C\_NAK\_INTERRUPT** - Not-acknowledge interrupt
- **USCI\_B\_I2C\_ARBITRATIONLOST\_INTERRUPT** - Arbitration lost interrupt

Modified bits of **UCBxIE** register.

**Returns:**

None



#### 40.2.2.4 USCI\_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	36
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_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 **UCBxCTL1** register.

**Returns:**

None

#### 40.2.2.5 USCI\_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	56
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_B_I2C_enableInterrupt(uint32_t baseAddress,
                           uint8_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. Does not clear interrupt flags.

**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:

- **USCI\_B\_I2C\_STOP\_INTERRUPT** - STOP condition interrupt
- **USCI\_B\_I2C\_START\_INTERRUPT** - START condition interrupt
- **USCI\_B\_I2C\_RECEIVE\_INTERRUPT** - Receive interrupt
- **USCI\_B\_I2C\_TRANSMIT\_INTERRUPT** - Transmit interrupt
- **USCI\_B\_I2C\_NAK\_INTERRUPT** - Not-acknowledge interrupt
- **USCI\_B\_I2C\_ARBITRATIONLOST\_INTERRUPT** - Arbitration lost interrupt

Modified bits of **UCBxIE** register.

**Returns:**

None

### 40.2.2.6 USCI\_B\_I2C\_getInterruptStatus

Gets the current I2C 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	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint8_t
USCI_B_I2C_getInterruptStatus(uint32_t baseAddress,
                               uint8_t mask)
```

**Description:**

This returns the interrupt status for the I2C module based on which flag is passed. mask parameter can be logic OR of any of the following selection.

**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:

- **USCI\_B\_I2C\_STOP\_INTERRUPT** - STOP condition interrupt
- **USCI\_B\_I2C\_START\_INTERRUPT** - START condition interrupt
- **USCI\_B\_I2C\_RECEIVE\_INTERRUPT** - Receive interrupt
- **USCI\_B\_I2C\_TRANSMIT\_INTERRUPT** - Transmit interrupt
- **USCI\_B\_I2C\_NAK\_INTERRUPT** - Not-acknowledge interrupt
- **USCI\_B\_I2C\_ARBITRATIONLOST\_INTERRUPT** - Arbitration lost interrupt

**Returns:**

the masked status of the interrupt flag Return Logical OR of any of the following:

- **USCI\_B\_I2C\_STOP\_INTERRUPT** STOP condition interrupt
  - **USCI\_B\_I2C\_START\_INTERRUPT** START condition interrupt
  - **USCI\_B\_I2C\_RECEIVE\_INTERRUPT** Receive interrupt
  - **USCI\_B\_I2C\_TRANSMIT\_INTERRUPT** Transmit interrupt
  - **USCI\_B\_I2C\_NAK\_INTERRUPT** Not-acknowledge interrupt
  - **USCI\_B\_I2C\_ARBITRATIONLOST\_INTERRUPT** Arbitration lost interrupt
- indicating the status of the masked interrupts

#### 40.2.2.7 USCI\_B\_I2C\_getReceiveBufferAddressForDMA

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
USCI_B_I2C_getReceiveBufferAddressForDMA(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 RX Buffer

#### 40.2.2.8 USCI\_B\_I2C\_getTransmitBufferAddressForDMA

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
USCI_B_I2C_getTransmitBufferAddressForDMA(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 TX Buffer

### 40.2.2.9 USCI\_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	12
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	12
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	10
MSPGCC 4.8.0	Speed	10

#### Prototype:

```
uint8_t
USCI_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:

Returns USCI\_B\_I2C\_BUS\_BUSY if the I2C Master is busy; otherwise, returns USCI\_B\_I2C\_BUS\_NOT\_BUSY. Return one of the following:

- **USCI\_B\_I2C\_BUS\_BUSY**
- **USCI\_B\_I2C\_BUS\_NOT\_BUSY**  
indicating if the USCI\_B\_I2C is busy

### 40.2.2.10 USCI\_B\_I2C\_isBusy

DEPRECATED - Function may be removed in future release. Indicates whether or not the I2C module is busy.

#### Code Metrics:

Compiler	Optimization	Code Size
CCS 4.2.1	None	32
CCS 4.2.1	Size	18
CCS 4.2.1	Speed	18
IAR 5.51.6	None	28
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	22
MSPGCC 4.8.0	None	46
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	22

#### Prototype:

```
uint8_t
USCI_B_I2C_isBusy(uint32_t baseAddress)
```

#### Description:

This function returns an indication of whether or not the I2C module is busy transmitting or receiving data. This function checks if the Transmit or receive flag is set.

**Parameters:**

**baseAddress** is the base address of the I2C module.

**Returns:**

Returns USCI\_B\_I2C\_BUS\_BUSY if the I2C module is busy; otherwise, returns USCI\_B\_I2C\_BUS\_NOT\_BUSY.  
Return one of the following:

- **USCI\_B\_I2C\_BUS\_BUSY**
- **USCI\_B\_I2C\_BUS\_NOT\_BUSY**  
indicating if the USCI\_B\_I2C is busy

### 40.2.2.11 USCI\_B\_I2C\_masterInit

Initializes the I2C Master block.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	90
CCS 4.2.1	Size	48
CCS 4.2.1	Speed	48
IAR 5.51.6	None	74
IAR 5.51.6	Size	60
IAR 5.51.6	Speed	60
MSPGCC 4.8.0	None	112
MSPGCC 4.8.0	Size	72
MSPGCC 4.8.0	Speed	72

**Prototype:**

```
void
USCI_B_I2C_masterInit (uint32_t baseAddress,
                      uint8_t selectClockSource,
                      uint32_t i2cClk,
                      uint32_t dataRate)
```

**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 USCI\_B\_I2C\_enable is invoked. If the parameter *dataRate* is USCI\_B\_I2C\_SET\_DATA\_RATE\_400KBPS, then the master block will be set up to transfer data at 400 kbps; otherwise, it will be set up to transfer data at 100 kbps.

**Parameters:**

**baseAddress** is the base address of the I2C Master module.

**selectClockSource** is the clocksource. Valid values are:

- **USCI\_B\_I2C\_CLOCKSOURCE\_ACLK**
- **USCI\_B\_I2C\_CLOCKSOURCE\_SMCLK**

**i2cClk** is the rate of the clock supplied to the I2C module.

**dataRate** set up for selecting data transfer rate. Valid values are:

- **USCI\_B\_I2C\_SET\_DATA\_RATE\_400KBPS**
- **USCI\_B\_I2C\_SET\_DATA\_RATE\_100KBPS**

Modified bits are **UCBxBR0** of **UCBxBR1** register; bits **UCSSELx** and **UCSWRST** of **UCBxCTL1** register; bits **UCMST**, **UCMODE\_3** and **UCSYNC** of **UCBxCTL0** register.

**Returns:**

None

## 40.2.2.12 USCI\_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	30
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
uint8_t
USCI_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 module.

**Returns:**

Returns USCI\_B\_I2C\_START\_SEND\_COMPLETE if the I2C Master finished sending START; otherwise, returns USCI\_B\_I2C\_SENDING\_START. Return one of the following:

- USCI\_B\_I2C\_SENDING\_START
- USCI\_B\_I2C\_START\_SEND\_COMPLETE

## 40.2.2.13 USCI\_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	30
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
uint8_t
USCI_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 module.

**Returns:**

Returns USCI\_B\_I2C\_STOP\_SEND\_COMPLETE if the I2C Master finished sending STOP; otherwise, returns USCI\_B\_I2C\_SENDING\_STOP. Return one of the following:

- USCI\_B\_I2C\_SENDING\_STOP
- USCI\_B\_I2C\_STOP\_SEND\_COMPLETE

#### 40.2.2.14 USCI\_B\_I2C\_masterMultiByteReceiveFinish

Finishes multi-byte reception at the Master end.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	50
CCS 4.2.1	Size	24
CCS 4.2.1	Speed	24
IAR 5.51.6	None	38
IAR 5.51.6	Size	32
IAR 5.51.6	Speed	38
MSPGCC 4.8.0	None	90
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	34

**Prototype:**

```
uint8_t
USCI_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 does the following: - 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 **UCBxCTL1** register.

**Returns:**

Received byte at Master end.

#### 40.2.2.15 USCI\_B\_I2C\_masterMultiByteReceiveFinishWithTimeout

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

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	144
CCS 4.2.1	Size	80
CCS 4.2.1	Speed	80
IAR 5.51.6	None	106
IAR 5.51.6	Size	86
IAR 5.51.6	Speed	100
MSPGCC 4.8.0	None	218
MSPGCC 4.8.0	Size	118
MSPGCC 4.8.0	Speed	136

**Prototype:**

```
bool
USCI_B_I2C_masterMultiByteReceiveFinishWithTimeout (uint32_t baseAddress,
                                                    uint8_t *rxData,
                                                    uint32_t timeout)
```

**Description:**

This function is used by the Master module to initiate completion of a multi-byte reception. This function does the following: - Receives the current byte and initiates the STOP from Master to Slave

**Parameters:**

**baseAddress** is the base address of the I2C Master module.  
**rxData** 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 **UCBxCTL1** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of the transmission process.

#### 40.2.2.16 USCI\_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	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	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint8_t
USCI_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.

#### 40.2.2.17 USCI\_B\_I2C\_masterMultiByteReceiveStart

Starts multi-byte 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	56
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

**Prototype:**

```
void
USCI_B_I2C_masterMultiByteReceiveStart (uint32_t baseAddress)
```

**Description:**

This function is used by the Master module initiate reception of a single byte. This function does the following: - Sends START

**Parameters:**

**baseAddress** is the base address of the I2C Master module.

Modified bits are **UCTXSTT** of **UCBxCTL1** register.

**Returns:**

None

#### 40.2.2.18 USCI\_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	34
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_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 **UCBxCTL1** register.

**Returns:**

None

### 40.2.2.19 USCI\_B\_I2C\_masterMultiByteSendFinish

Finishes multi-byte transmission from Master to Slave.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	56
CCS 4.2.1	Size	28
CCS 4.2.1	Speed	28
IAR 5.51.6	None	54
IAR 5.51.6	Size	50
IAR 5.51.6	Speed	48
MSPGCC 4.8.0	None	124
MSPGCC 4.8.0	Size	44
MSPGCC 4.8.0	Speed	44

**Prototype:**

```
void
USCI_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 does the following: - Transmits the last data byte of a multi-byte transmission to the Slave; - Sends 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 **UCBxCTL1** register.

**Returns:**

None

### 40.2.2.20 USCI\_B\_I2C\_masterMultiByteSendFinishWithTimeout

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

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	144
CCS 4.2.1	Size	86
CCS 4.2.1	Speed	86
IAR 5.51.6	None	124
IAR 5.51.6	Size	110
IAR 5.51.6	Speed	118
MSPGCC 4.8.0	None	234
MSPGCC 4.8.0	Size	124
MSPGCC 4.8.0	Speed	132

**Prototype:**

```
bool
USCI_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 does the following: - Transmits the last data byte of a multi-byte transmission to the Slave; - Sends 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 **UCBxCTL1** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of the transmission process.

#### 40.2.2.21 USCI\_B\_I2C\_masterMultiByteSendNext

Continues multi-byte transmission from Master to Slave.

**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	36
IAR 5.51.6	Size	12
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	78
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

**Prototype:**

```
void
USCI_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 does the following: -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

#### 40.2.2.22 USCI\_B\_I2C\_masterMultiByteSendNextWithTimeout

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

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	86
CCS 4.2.1	Size	48
CCS 4.2.1	Speed	48
IAR 5.51.6	None	78
IAR 5.51.6	Size	50
IAR 5.51.6	Speed	74
MSPGCC 4.8.0	None	138
MSPGCC 4.8.0	Size	92
MSPGCC 4.8.0	Speed	104

**Prototype:**

```
bool
USCI_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 does the following: -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.

#### 40.2.2.23 USCI\_B\_I2C\_masterMultiByteSendStart

Starts multi-byte transmission from Master to Slave.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	78
CCS 4.2.1	Size	36
CCS 4.2.1	Speed	36
IAR 5.51.6	None	64
IAR 5.51.6	Size	30
IAR 5.51.6	Speed	50
MSPGCC 4.8.0	None	170
MSPGCC 4.8.0	Size	50
MSPGCC 4.8.0	Speed	50

**Prototype:**

```
void
USCI_B_I2C_masterMultiByteSendStart (uint32_t baseAddress,
                                      uint8_t txData)
```

**Description:**

This function is used by the Master module to send a single byte. This function does the following: - Sends START; - Transmits the first data byte of a multi-byte transmission to the Slave

**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 **UCBxIFG** register, bits of **UCBxCTL1** register and bits of **UCBxIE** register.

**Returns:**

None

#### 40.2.2.24 USCI\_B\_I2C\_masterMultiByteSendStartWithTimeout

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

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	122
CCS 4.2.1	Size	70
CCS 4.2.1	Speed	70
IAR 5.51.6	None	102
IAR 5.51.6	Size	68
IAR 5.51.6	Speed	86
MSPGCC 4.8.0	None	230
MSPGCC 4.8.0	Size	110
MSPGCC 4.8.0	Speed	128

**Prototype:**

```
bool
USCI_B_I2C_masterMultiByteSendStartWithTimeout (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 does the following: - Sends START; - Transmits the first data byte of a multi-byte transmission to the Slave

**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

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of the transmission process.

#### 40.2.2.25 USCI\_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	72
MSPGCC 4.8.0	Size	24
MSPGCC 4.8.0	Speed	24

**Prototype:**

```
void
USCI_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 does the following: - Sends a STOP after current transmission is complete

**Parameters:**

**baseAddress** is the base address of the I2C Master module.

Modified bits are **UCTXSTP** of **UCBxCTL1** register.

**Returns:**

None

#### 40.2.2.26 USCI\_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	126
MSPGCC 4.8.0	Size	56
MSPGCC 4.8.0	Speed	52

**Prototype:**

```
bool
USCI_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 does the following: - 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 **UCBxCTL1** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of the transmission process.

#### 40.2.2.27 USCI\_B\_I2C\_masterSendSingleByte

Does single byte transmission from Master to Slave.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	102
CCS 4.2.1	Size	50
CCS 4.2.1	Speed	50
IAR 5.51.6	None	92
IAR 5.51.6	Size	52
IAR 5.51.6	Speed	66
MSPGCC 4.8.0	None	250
MSPGCC 4.8.0	Size	70
MSPGCC 4.8.0	Speed	70

**Prototype:**

```
void
USCI_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 does the following: - Sends START; - Transmits the byte to the Slave; - Sends 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 **UCBxIFG** register, bits of **UCBxCTL1** register and bits of **UCBxIE** register.

**Returns:**

None

#### 40.2.2.28 USCI\_B\_I2C\_masterSendSingleByteWithTimeout

Does single byte transmission from Master to Slave with timeout.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	190
CCS 4.2.1	Size	108
CCS 4.2.1	Speed	108
IAR 5.51.6	None	158
IAR 5.51.6	Size	124
IAR 5.51.6	Speed	128
MSPGCC 4.8.0	None	364
MSPGCC 4.8.0	Size	148
MSPGCC 4.8.0	Speed	160

**Prototype:**

```
bool
USCI_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 does the following: - Sends START; - Transmits the byte to the Slave; - Sends 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 **UCBxIFG** register, bits of **UCBxCTL1** register and bits of **UCBxIE** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of the transmission process.

#### 40.2.2.29 USCI\_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	34
MSPGCC 4.8.0	Size	6
MSPGCC 4.8.0	Speed	6

**Prototype:**

```
void
USCI_B_I2C_masterSendStart(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.

**Returns:**

None

#### 40.2.2.30 USCI\_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	32
CCS 4.2.1	Size	18
CCS 4.2.1	Speed	18
IAR 5.51.6	None	32
IAR 5.51.6	Size	18
IAR 5.51.6	Speed	24
MSPGCC 4.8.0	None	66
MSPGCC 4.8.0	Size	26
MSPGCC 4.8.0	Speed	26



**Prototype:**

```
uint8_t
USCI_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 module.

**Returns:**

Returns the byte received from by the I2C module, cast as an uint8\_t.

### 40.2.2.31 USCI\_B\_I2C\_masterSingleReceiveStart

Initiates a single byte Reception at the Master End.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	62
CCS 4.2.1	Size	34
CCS 4.2.1	Speed	34
IAR 5.51.6	None	40
IAR 5.51.6	Size	22
IAR 5.51.6	Speed	32
MSPGCC 4.8.0	None	118
MSPGCC 4.8.0	Size	30
MSPGCC 4.8.0	Speed	30

**Prototype:**

```
void
USCI_B_I2C_masterSingleReceiveStart(uint32_t baseAddress)
```

**Description:**

This function sends a START and STOP immediately to indicate Single byte reception

**Parameters:**

**baseAddress** is the base address of the I2C Master module.

Modified bits are **GIE** of **SR** register; bits **UCTXSTT** and **UCTXSTP** of **UCBxCTL1** register.

**Returns:**

None

### 40.2.2.32 USCI\_B\_I2C\_masterSingleReceiveStartWithTimeout

Initiates a single byte Reception at the Master End with timeout.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	106
CCS 4.2.1	Size	60
CCS 4.2.1	Speed	60
IAR 5.51.6	None	80
IAR 5.51.6	Size	60
IAR 5.51.6	Speed	70
MSPGCC 4.8.0	None	186
MSPGCC 4.8.0	Size	74
MSPGCC 4.8.0	Speed	72

**Prototype:**

```
bool
USCI_B_I2C_masterSingleReceiveStartWithTimeout (uint32_t baseAddress,
                                                uint32_t timeout)
```

**Description:**

This function sends a START and STOP immediately to indicate Single byte reception

**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 **GIE** of **SR** register; bits **UCTXSTT** and **UCTXSTP** of **UCBxCTL1** register.

**Returns:**

STATUS\_SUCCESS or STATUS\_FAILURE of the transmission process.

### 40.2.2.33 USCI\_B\_I2C\_setMode

Sets the mode of the I2C device.

**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	8
IAR 5.51.6	Speed	8
MSPGCC 4.8.0	None	74
MSPGCC 4.8.0	Size	12
MSPGCC 4.8.0	Speed	12

**Prototype:**

```
void
USCI_B_I2C_setMode (uint32_t baseAddress,
                   uint8_t mode)
```

**Description:**

When the receive parameter is set to USCI\_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 I2C Master module.

**mode** indicates whether module is in transmit/receive mode Valid values are:

- USCI\_B\_I2C\_TRANSMIT\_MODE

## ■ USCI\_B\_I2C\_RECEIVE\_MODE [Default]

**Returns:**  
None

## 40.2.2.34 USCI\_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	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
USCI_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 I2C Master module.

**slaveAddress** 7-bit slave address

Modified bits of **UCBxI2CSA** register; bits **UCSWRST** of **UCBxCTL1** register.

**Returns:**

None

## 40.2.2.35 USCI\_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	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	4
MSPGCC 4.8.0	None	28
MSPGCC 4.8.0	Size	8
MSPGCC 4.8.0	Speed	8

**Prototype:**

```
uint8_t
USCI_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 module.

**Returns:**

Returns the byte received from by the I2C module, cast as an uint8\_t.

### 40.2.2.36 USCI\_B\_I2C\_slaveDataPut

Transmits a byte from the I2C Module.

**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	12
IAR 5.51.6	Size	2
IAR 5.51.6	Speed	2
MSPGCC 4.8.0	None	36
MSPGCC 4.8.0	Size	14
MSPGCC 4.8.0	Speed	14

**Prototype:**

```
void
USCI_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 Modified bit is UCBxTXBUF register

**Parameters:**

**baseAddress** is the base address of the I2C module.

**transmitData** data to be transmitted from the I2C module

Modified bits of **UCBxTXBUF** register.

**Returns:**

None

### 40.2.2.37 USCI\_B\_I2C\_slaveInit

Initializes the I2C Slave block.

**Code Metrics:**

Compiler	Optimization	Code Size
CCS 4.2.1	None	54
CCS 4.2.1	Size	26
CCS 4.2.1	Speed	26
IAR 5.51.6	None	38
IAR 5.51.6	Size	34
IAR 5.51.6	Speed	34
MSPGCC 4.8.0	None	104
MSPGCC 4.8.0	Size	32
MSPGCC 4.8.0	Speed	32

**Prototype:**

```
void
USCI_B_I2C_slaveInit(uint32_t baseAddress,
                    uint8_t slaveAddress)
```

**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 USCI\_B\_I2C\_enable is invoked.

**Parameters:**

**baseAddress** is the base address of the I2C Slave module.

**slaveAddress** 7-bit slave address

Modified bits of **UCBxI2COA** register; bits **UCSWRST** of **UCBxCTL1** register; bits **UCMODE\_3** and **UCSYNC** of **UCBxCTL0** register.

**Returns:**

None

## 40.3 Programming Example

The following example shows how to use the USCI\_B\_I2C API to send data as a master.

```
// Initialize Master
USCI_B_I2C_masterInit(USCI_B0_BASE, SMCLK, CLK_getSMCLK(), USCI_B_I2C_SET_DATA_RATE_400KBPS);

// Specify slave address
USCI_B_I2C_setSlaveAddress(USCI_B0_BASE, SLAVE_ADDRESS);

// Set in transmit mode
USCI_B_I2C_setMode(USCI_B0_BASE, USCI_B_I2C_TRANSMIT_MODE);

//Enable USCI_B_I2C Module to start operations
USCI_B_I2C_enable(USCI_B0_BASE);

while (1)
{
    // Send single byte data.
    USCI_B_I2C_masterSendSingleByte(USCI_B0_BASE, transmitData);

    // Delay until transmission completes
    while(USCI_B_I2C_busBusy(USCI_B0_BASE));

    // Increment transmit data counter
    transmitData++;
}
```

# 41 WatchDog Timer (WDT\_A)

Introduction .....	530
API Functions .....	530
Programming Example .....	534

## 41.1 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

## 41.2 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)

### 41.2.1 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\(\)](#)

## 41.2.2 Function Documentation

### 41.2.2.1 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

### 41.2.2.2 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 **WDTSEL** of **WDTCTL** register.

**clockDivider** is the divider of the clock source, in turn setting the watchdog timer interval. Valid values are:

- WDT\_A\_CLOCKDIVIDER\_2G
- WDT\_A\_CLOCKDIVIDER\_128M
- WDT\_A\_CLOCKDIVIDER\_8192K
- WDT\_A\_CLOCKDIVIDER\_512K
- WDT\_A\_CLOCKDIVIDER\_32K [Default]
- WDT\_A\_CLOCKDIVIDER\_8192
- WDT\_A\_CLOCKDIVIDER\_512
- WDT\_A\_CLOCKDIVIDER\_64

Modified bits are **WDTIS** and **WDTHOLD** of **WDTCTL** register.

**Returns:**

None

### 41.2.2.3 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



#### 41.2.2.4 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

#### 41.2.2.5 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 **WDTSEL** of **WDTCTL** register.

**clockDivider** is the divider of the clock source, in turn setting the watchdog timer interval. Valid values are:

- WDT\_A\_CLOCKDIVIDER\_2G
- WDT\_A\_CLOCKDIVIDER\_128M
- WDT\_A\_CLOCKDIVIDER\_8192K
- WDT\_A\_CLOCKDIVIDER\_512K
- WDT\_A\_CLOCKDIVIDER\_32K [Default]
- WDT\_A\_CLOCKDIVIDER\_8192
- WDT\_A\_CLOCKDIVIDER\_512
- WDT\_A\_CLOCKDIVIDER\_64

Modified bits are **WDTIS** and **WDTHOLD** of **WDTCTL** register.

**Returns:**

None

## 41.3 Programming Example

The following example shows how to initialize and use the WDT\_A API to interrupt about every 32 ms, toggling the LED in the ISR.

```
//Initialize WDT_A module in timer interval mode,
//with SMCLK as source at an interval of 32 ms.
WDT_A_intervalTimerInit(WDT_A_BASE,
    WDT_A_CLOCKSOURCE_SMCLK,
    WDT_A_CLOCKDIVIDER_32K);

//Enable Watchdog Interrupt
SFR_enableInterrupt(SFR_WATCHDOG_INTERVAL_TIMER_INTERRUPT);

//Set P1.0 to output direction
GPIO_setAsOutputPin(
    GPIO_PORT_P1,
    GPIO_PIN0
);

//Enter LPM0, enable interrupts
__bis_SR_register(LPM0_bits + GIE);
//For debugger
__no_operation();
```

---

# IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

## Products

Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DLP® Products	<a href="http://www.dlp.com">www.dlp.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>
RF/IF and ZigBee® Solutions	<a href="http://www.ti.com/lprf">www.ti.com/lprf</a>

## Applications

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Broadband	<a href="http://www.ti.com/broadband">www.ti.com/broadband</a>
Digital Control	<a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a>
Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Military	<a href="http://www.ti.com/military">www.ti.com/military</a>
Optical Networking	<a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Telephony	<a href="http://www.ti.com/telephony">www.ti.com/telephony</a>
Video & Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
Wireless	<a href="http://www.ti.com/wireless">www.ti.com/wireless</a>

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2014, Texas Instruments Incorporated