Here is a list of all modules:

<table>
<thead>
<tr>
<th>Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC_Analog_to_Digital_Converter_api</td>
</tr>
<tr>
<td>AES_Advanced_Encryption.Standard_api</td>
</tr>
<tr>
<td>Camera_api</td>
</tr>
<tr>
<td>CRC_Cyclic_Redundancy_Check_api</td>
</tr>
<tr>
<td>DES_Data_Encryption.Standard_api</td>
</tr>
<tr>
<td>Flash_api</td>
</tr>
<tr>
<td>GPIO_General_Purpose_Input_output_api</td>
</tr>
<tr>
<td>HwSpinLock_api</td>
</tr>
<tr>
<td>I2C_api</td>
</tr>
<tr>
<td>I2S_api</td>
</tr>
<tr>
<td>Interrupt_api</td>
</tr>
<tr>
<td>Pin_api</td>
</tr>
<tr>
<td>PRCM_Power_Reset_Clock_Module_api</td>
</tr>
<tr>
<td>Secure_Digital_Host_api</td>
</tr>
<tr>
<td>SHA_Secure_Hash_Algorithm_api</td>
</tr>
<tr>
<td>SPI_Serial_Peripheral_Interface_api</td>
</tr>
<tr>
<td>Systick_api</td>
</tr>
<tr>
<td>GPT_General_Purpose_Timer_api</td>
</tr>
<tr>
<td>UART_api</td>
</tr>
<tr>
<td>UDMA_Micro_Direct_Memory_Access_api</td>
</tr>
<tr>
<td>Main Page</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td><strong>ADC_Analog_to_Digital_Converter_api</strong></td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>ADCEnable</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void</td>
<td>ADCDisable</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void</td>
<td>ADCChannelEnable</td>
<td>(unsigned long ulBase, unsigned long ulChannel)</td>
</tr>
<tr>
<td>void</td>
<td>ADCChannelDisable</td>
<td>(unsigned long ulBase, unsigned long ulChannel)</td>
</tr>
<tr>
<td>void</td>
<td>ADCIntRegister</td>
<td>(unsigned long ulBase, unsigned long ulChannel, void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>void</td>
<td>ADCIntUnregister</td>
<td>(unsigned long ulBase, unsigned long ulChannel)</td>
</tr>
<tr>
<td>void</td>
<td>ADCIntEnable</td>
<td>(unsigned long ulBase, unsigned long ulChannel, unsigned long ulIntFlags)</td>
</tr>
<tr>
<td>void</td>
<td>ADCIntDisable</td>
<td>(unsigned long ulBase, unsigned long ulChannel, unsigned long ulIntFlags)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>ADCIntStatus</td>
<td>(unsigned long ulBase, unsigned long ulChannel)</td>
</tr>
<tr>
<td></td>
<td>ADCIntClear</td>
<td>(unsigned long ulBase, unsigned long ulChannel)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>void unsigned long ulChannel, unsigned long ullIntFlags</td>
<td>ADCDMAEnable (unsigned long ulBase, unsigned long ulChannel)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>ADCDMADisable (unsigned long ulBase, unsigned long ulChannel)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>ADCTimerConfig (unsigned long ulBase, unsigned long ulValue)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>ADCTimerReset (unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>ADCTimerEnable (unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>ADCTimerDisable (unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td>unsigned long</td>
<td>ADCTimerValueGet (unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>ADCFIFOLvlGet (unsigned long ulBase, unsigned long ulChannel)</td>
<td></td>
</tr>
<tr>
<td>unsigned long</td>
<td>ADCFIFORead (unsigned long ulBase, unsigned long ulChannel)</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

```c
void ADCChannelDisable ( unsigned long ulBase,
                        unsigned long ulChannel )
```

Disables specified ADC channel

**Parameters**
- **ulBase** is the base address of the ADC
- **ulChannel** is one of the valid ADC channels

This function disables specified ADC channel.

**Returns**
None.

```c
void ADCChannelEnable ( unsigned long ulBase,
                       unsigned long ulChannel )
```

Enables specified ADC channel

**Parameters**
- **ulBase** is the base address of the ADC
ulChannel is one of the valid ADC channels

This function enables specified ADC channel and configures the pin as analog pin.

**Returns**
None.

```c
void ADCDisable ( unsigned long ulBase )
```

Disable the ADC

**Parameters**
- **ulBase** is the base address of the ADC

This function clears the ADC global enable

**Returns**
None.

```c
void ADCDMADisable ( unsigned long ulBase, unsigned long ulChannel )
```

Disables the ADC DMA operation for specified channel

**Parameters**
- **ulBase** is the base address of the ADC
- **ulChannel** is one of the valid ADC channels

This function disables the DMA operation for specified ADC
The parameter `ulChannel` should be one of the following:

- `ADC_CH_0` for channel 0
- `ADC_CH_1` for channel 1
- `ADC_CH_2` for channel 2
- `ADC_CH_3` for channel 3

**Returns**
None.

```c
void ADCDMAEnable (unsigned long ulBase, unsigned long ulChannel )
```

Enables the ADC DMA operation for specified channel

**Parameters**
- `ulBase` is the base address of the ADC
- `ulChannel` is one of the valid ADC channels

This function enables the DMA operation for specified ADC channel

The parameter `ulChannel` should be one of the following:

- `ADC_CH_0` for channel 0
- `ADC_CH_1` for channel 1
- `ADC_CH_2` for channel 2
- `ADC_CH_3` for channel 3
Returns
None.

`void ADCEnable ( unsigned long ulBase )`

Enables the ADC

Parameters

`ulBase` is the base address of the ADC

This function sets the ADC global enable

Returns
None.

`unsigned char ADCFIFOLvlGet ( unsigned long ulBase, unsigned long ulChannel )`

Gets the current FIFO level for specified ADC channel

Parameters

`ulBase` is the base address of the ADC
`ulChannel` is one of the valid ADC channels.

This function returns the current FIFO level for specified ADC channel.

The parameter `ulChannel` should be one of the following
- ADC_CH_0 for channel 0
- ADC_CH_1 for channel 1
- ADC_CH_2 for channel 2
- ADC_CH_3 for channel 3

**Returns**
Return the current FIFO level for specified channel

```c
unsigned long ADCFIFORead ( unsigned long ulBase,
                            unsigned long ulChannel )
```

Reads FIFO for specified ADC channel

**Parameters**
- **ulBase** is the base address of the ADC
- **ulChannel** is one of the valid ADC channels.

This function returns one data sample from the channel fifo as specified by *ulChannel* parameter.

The parameter *ulChannel* should be one of the following

- ADC_CH_0 for channel 0
- ADC_CH_1 for channel 1
- ADC_CH_2 for channel 2
- ADC_CH_3 for channel 3

**Returns**
Return one data sample from the channel fifo.
void ADCIntClear ( unsigned long ulBase,
                 unsigned long ulChannel,
                 unsigned long ulIntFlags
)

Clears the current channel interrupt sources

Parameters
ulBase is the base address of the ADC
ulChannel is one of the valid ADC channels
ulIntFlags is the bit mask of the interrupt sources to be cleared.

This function clears individual interrupt source for the specified ADC channel.

The parameter ulChannel should be as explained in

See also
ADCIntEnable().

Returns
None.

void ADCIntDisable ( unsigned long ulBase,
                     unsigned long ulChannel,
                     unsigned long ulIntFlags
)

Disables individual interrupt sources for specified channel
Parameters

ulBase is the base address of the ADC.
ulChannel is one of the valid ADC channels
ulIntFlags is the bit mask of the interrupt sources to be enabled.

This function disables the indicated ADC interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The parameters ulIntFlags and ulChannel should be as explained in ADCIntEnable().

Returns

None.

void ADCIntEnable ( unsigned long ulBase,  
unsigned long ulChannel,  
unsigned long ulIntFlags  
)

Enables individual interrupt sources for specified channel

Parameters

ulBase is the base address of the ADC
ulChannel is one of the valid ADC channels
ulIntFlags is the bit mask of the interrupt sources to be enabled.

This function enables the indicated ADC interrupt sources.
Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The parameter \textit{ulChannel} should be one of the following:

- \texttt{ADC\_CH\_0} for channel 0
- \texttt{ADC\_CH\_1} for channel 1
- \texttt{ADC\_CH\_2} for channel 2
- \texttt{ADC\_CH\_3} for channel 3

The \textit{ullIntFlags} parameter is the logical OR of any of the following:

- \texttt{ADC\_DMA\_DONE} for DMA done
- \texttt{ADC\_FIFO\_OVERFLOW} for FIFO over flow
- \texttt{ADC\_FIFO\_UNDERFLOW} for FIFO under flow
- \texttt{ADC\_FIFO\_EMPTY} for FIFO empty
- \texttt{ADC\_FIFO\_FULL} for FIFO full

\textbf{Returns}

None.

\begin{verbatim}
void ADCIntRegister ( unsigned long ulBase, 
                     unsigned long ulChannel, 
                     void(*)(void) pfnHandler
)
\end{verbatim}

Enables and registers ADC interrupt handler for specified channel

\textbf{Parameters}
ulBase is the base address of the ADC
ulChannel is one of the valid ADC channels
pfnHandler is a pointer to the function to be called when the ADC channel interrupt occurs.

This function enables and registers ADC interrupt handler for specified channel. Individual interrupt for each channel should be enabled using

See also
ADCIntEnable(). It is the interrupt handler's responsibility to clear the interrupt source.

The parameter ulChannel should be one of the following

- ADC_CH_0 for channel 0
- ADC_CH_1 for channel 1
- ADC_CH_2 for channel 2
- ADC_CH_3 for channel 3

Returns
None.

unsigned long ADCIntStatus ( unsigned long ulBase, unsigned long ulChannel )

Gets the current channel interrupt status

Parameters
ulBase is the base address of the ADC
ulChannel is one of the valid ADC channels
This function returns the interrupt status of the specified ADC channel.

The parameter `ulChannel` should be as explained in

See also

`ADCIntEnable()`.

**Returns**

Return the ADC channel interrupt status, enumerated as a bit field of values described in `ADCIntEnable()`

```c
void ADCIntUnregister ( unsigned long ulBase,
                       unsigned long ulChannel )
```

Disables and unregisters ADC interrupt handler for specified channel

**Parameters**

- `ulBase` is the base address of the ADC
- `ulChannel` is one of the valid ADC channels

This function disables and unregisters ADC interrupt handler for specified channel. This function also masks off the interrupt in the interrupt controller so that the interrupt handler no longer is called.

The parameter `ulChannel` should be one of the following

- `ADC_CH_0` for channel 0
- `ADC_CH_1` for channel 1
• **ADC_CH_2** for channel 2  
• **ADC_CH_3** for channel 3

**Returns**

None.

```c
void ADCTimerConfig ( unsigned long ulBase,  
                      unsigned long ulValue  
)  
```

Configures the ADC internal timer

**Parameters**

- **ulBase** is the base address of the ADC
- **ulValue** is wrap around value of the timer

This function Configures the ADC internal timer. The ADC timer is a 17 bit used to timestamp the ADC data samples internally. User can read the timestamp along with the sample from the FIFO register(s). Each sample in the FIFO contains 14 bit actual data and 18 bit timestamp

The parameter **ulValue** can take any value between 0 - \(2^{17}\)

**Returns**

None.

```c
void ADCTimerDisable ( unsigned long ulBase )  
```

Disables ADC internal timer
Parameters

*ulBase* is the base address of the ADC

This function disables 17-bit ADC internal timer

**Returns**

None.

---

```c
void ADCTimerEnable ( unsigned long ulBase )
```

Enables ADC internal timer

**Parameters**

*ulBase* is the base address of the ADC

This function enables 17-bit ADC internal timer

**Returns**

None.

---

```c
void ADCTimerReset ( unsigned long ulBase )
```

Resets ADC internal timer

**Parameters**

*ulBase* is the base address of the ADC

This function resets 17-bit ADC internal timer

**Returns**

None.
unsigned long ADCTimerValueGet (unsigned long ulBase)

Gets the current value of ADC internal timer

Parameters

ulBase is the base address of the ADC

This function the current value of 17-bit ADC internal timer

Returns

Return the current value of ADC internal timer.
CC3200 Peripheral Driver Library User's Guide 1.2.0

 AES_Advanced_Encryption_Standard_api
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>void AESConfigSet (uint32_t ui32Base, uint32_t ui32Config)</strong></td>
<td>Set configuration parameters for AES configurations.</td>
</tr>
<tr>
<td>*<em>void AESKey1Set (uint32_t ui32Base, uint8_t <em>pui8Key, uint32_t ui32Keysize)</em></em></td>
<td>Set the first key for AES encryption.</td>
</tr>
<tr>
<td>*<em>void AESKey2Set (uint32_t ui32Base, uint8_t <em>pui8Key, uint32_t ui32Keysize)</em></em></td>
<td>Set the second key for AES encryption.</td>
</tr>
<tr>
<td>*<em>void AESKey3Set (uint32_t ui32Base, uint8_t <em>pui8Key)</em></em></td>
<td>Set the third key for AES encryption.</td>
</tr>
<tr>
<td>*<em>void AESIVSet (uint32_t ui32Base, uint8_t <em>pui8IVdata)</em></em></td>
<td>Set the initialization vector for AES encryption.</td>
</tr>
<tr>
<td>*<em>void AESIVGet (uint32_t ui32Base, uint8_t <em>pui8IVdata)</em></em></td>
<td>Get the initialization vector for AES encryption.</td>
</tr>
<tr>
<td>*<em>void AESTagRead (uint32_t ui32Base, uint8_t <em>pui8TagData)</em></em></td>
<td>Read the authentication tag for AES encryption.</td>
</tr>
<tr>
<td><strong>void AESDataLengthSet (uint32_t ui32Base, uint64_t ui64Length)</strong></td>
<td>Set the data length for AES encryption.</td>
</tr>
<tr>
<td><strong>void AESAuthDataLengthSet (uint32_t ui32Base, uint32_t ui32Length)</strong></td>
<td>Set the authentication data length for AES encryption.</td>
</tr>
<tr>
<td>*<em>bool AESDataReadNonBlocking (uint32_t ui32Base, uint8_t <em>pui8Dest, uint8_t ui8Length)</em></em></td>
<td>Read data non-blocking mode for AES encryption.</td>
</tr>
<tr>
<td><strong>void AESDataRead (uint32_t ui32Base, uint8_t</strong></td>
<td>Read data for AES encryption.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>AESDataWriteNonBlocking</code></td>
<td>(ui32_t ui32Base, uint8_t *pui8Src, uint8_t ui8Length)</td>
</tr>
<tr>
<td><code>AESDataWrite</code></td>
<td>(ui32_t ui32Base, uint8_t *pui8Src, uint8_t ui8Length)</td>
</tr>
<tr>
<td><code>AESDataProcess</code></td>
<td>(ui32_t ui32Base, uint8_t *pui8Src, uint8_t *pui8Dest, ui32_t ui32Length)</td>
</tr>
<tr>
<td><code>AESDataMAC</code></td>
<td>(ui32_t ui32Base, uint8_t *pui8Src, ui32_t ui32Length, uint8_t *pui8Tag)</td>
</tr>
<tr>
<td><code>AESDataProcessAE</code></td>
<td>(ui32_t ui32Base, uint8_t *pui8Src, uint8_t *pui8Dest, ui32_t ui32Length, ui8_t *pui8AuthSrc, ui32_t ui32AuthLength, ui8_t *pui8Tag)</td>
</tr>
<tr>
<td><code>AESIntStatus</code></td>
<td>(ui32_t ui32Base, bool bMasked)</td>
</tr>
<tr>
<td><code>AESIntEnable</code></td>
<td>(ui32_t ui32Base, ui32_t ui32IntFlags)</td>
</tr>
<tr>
<td><code>AESIntDisable</code></td>
<td>(ui32_t ui32Base, ui32_t ui32IntFlags)</td>
</tr>
<tr>
<td><code>AESIntClear</code></td>
<td>(ui32_t ui32Base, ui32_t ui32IntFlags)</td>
</tr>
<tr>
<td><code>AESIntRegister</code></td>
<td>(ui32_t ui32Base, void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>void AESIntUnregister</td>
<td>Unregisters an AES interrupt</td>
</tr>
<tr>
<td>void AESDMAEnable</td>
<td>Enables AES DMA</td>
</tr>
<tr>
<td>void AESDMADisable</td>
<td>Disables AES DMA</td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

void AESAuthDataLengthSet ( uint32_t ui32Base, uint32_t ui32Length )

Sets the optional additional authentication data (AAD) length.

Parameters

ui32Base is the base address of the AES module.
ui32Length is the length in bytes.

This function is only used to write the authentication data length in the combined modes (GCM or CCM) and XTS mode. Supported AAD lengths for CCM are from 0 to \((2^{16} - 28)\) bytes. For GCM, any value up to \((2^{32} - 1)\) can be used. For XTS mode, this register is used to load \(j\). Loading of \(j\) is only required if \(j \neq 0\). \(j\) represents the sequential number of the 128-bit blocks inside the data unit.
Consequently, \(j\) must be multiplied by 16 when passed to this function, thereby placing the block number in bits [31:4] of the register.

When this function is called, the engine is triggered to start using this context for GCM and CCM.
Returns
None

void AESConfigSet ( uint32_t ui32Base,
                    uint32_t ui32Config )

Configures the AES module.

Parameters
ui32Base is the base address of the AES module.
ui32Config is the configuration of the AES module.

This function configures the AES module based on the specified parameters. It does not change any DMA- or interrupt-related parameters.

The ui32Config parameter is a bit-wise OR of a number of configuration flags. The valid flags are grouped based on their function.

The direction of the operation is specified with only of following flags:

- AES_CFG_DIR_ENCRYPT - Encryption mode
- AES_CFG_DIR_Decrypt - Decryption mode

The key size is specified with only one of the following flags:

- AES_CFG_KEY_SIZE_128BIT - Key size of 128 bits
- AES_CFG_KEY_SIZE_192BIT - Key size of 192 bits
• **AES_CFG_KEY_SIZE_256BIT** - Key size of 256 bits

The mode of operation is specified with only one of the following flags.

• **AES_CFG_MODE_ECB** - Electronic codebook mode
• **AES_CFG_MODE_CBC** - Cipher-block chaining mode
• **AES_CFG_MODE_CFB** - Cipher feedback mode
• **AES_CFG_MODE_CTR** - Counter mode
• **AES_CFG_MODE_ICM** - Integer counter mode
• **AES_CFG_MODE_XTS** - Ciphertext stealing mode
• **AES_CFG_MODE_XTS_TWEAKJL** - XEX-based tweaked-codebook mode with ciphertext stealing with previous/intermediate tweak value and j loaded
• **AES_CFG_MODE_XTS_K2IJL** - XEX-based tweaked-codebook mode with ciphertext stealing with key2, i and j loaded
• **AES_CFG_MODE_XTS_K2ILJ0** - XEX-based tweaked-codebook mode with ciphertext stealing with key2 and i loaded, j = 0
• **AES_CFG_MODE_F8** - F8 mode
• **AES_CFG_MODE_F9** - F9 mode
• **AES_CFG_MODE_CBCMAC** - Cipher block chaining message authentication code mode
• **AES_CFG_MODE_GCM** - Galois/counter mode
• **AES_CFG_MODE_GCM_HLY0ZERO** - Galois/counter mode with GHASH with H loaded and Y0-encrypted forced to zero
• **AES_CFG_MODE_GCM_HLY0CALC** - Galois/counter mode with GHASH with H loaded and Y0-encrypted calculated internally
• **AES_CFG_MODE_GCM_HY0CALC** - Galois/Counter mode with autonomous GHASH (both H and Y0-
encrypted calculated internally)

- **AES_CFG_MODE_CCM** - Counter with CBC-MAC mode

The following defines are used to specify the counter width. It is only required to be defined when using CTR, CCM, or GCM modes, only one of the following defines must be used to specify the counter width length:

- **AES_CFG_CTR_WIDTH_32** - Counter is 32 bits
- **AES_CFG_CTR_WIDTH_64** - Counter is 64 bits
- **AES_CFG_CTR_WIDTH_96** - Counter is 96 bits
- **AES_CFG_CTR_WIDTH_128** - Counter is 128 bits

Only one of the following defines must be used to specify the length field for CCM operations (L):

- **AES_CFG_CCM_L_2** - 2 bytes
- **AES_CFG_CCM_L_4** - 4 bytes
- **AES_CFG_CCM_L_8** - 8 bytes

Only one of the following defines must be used to specify the length of the authentication field for CCM operations (M) through the `ui32Config` argument in the `AESConfigSet()` function:

- **AES_CFG_CCM_M_4** - 4 bytes
- **AES_CFG_CCM_M_6** - 6 bytes
- **AES_CFG_CCM_M_8** - 8 bytes
- **AES_CFG_CCM_M_10** - 10 bytes
- **AES_CFG_CCM_M_12** - 12 bytes
- **AES_CFG_CCM_M_14** - 14 bytes
- **AES_CFG_CCM_M_16** - 16 bytes
void AESDataLengthSet ( uint32_t ui32Base,  
                         uint64_t ui64Length )

Used to set the write crypto data length in the AES module.

Parameters

ui32Base is the base address of the AES module.
ui64Length is the crypto data length in bytes.

This function stores the cryptographic data length in blocks for all modes. Data lengths up to \((2^{61} - 1)\) bytes are allowed. For GCM, any value up to \((2^{36} - 2)\) bytes are allowed because a 32-bit block counter is used. For basic modes (ECB/CBC/CTR/ICM/CFB128), zero can be programmed into the length field, indicating that the length is infinite.

When this function is called, the engine is triggered to start using this context.

Note

This length does not include the authentication-only data used in some modes. Use the AESAuthLengthSet() function to specify the authentication data length.

Returns

None
bool AESDataMAC ( uint32_t ui32Base, 
    uint8_t * pui8Src, 
    uint32_t ui32Length, 
    uint8_t * pui8Tag 
)

Used to generate message authentication code (MAC) using CBC-MAC and F9 mode.

Parameters
    ui32Base is the base address of the AES module.
    pui8Src is a pointer to the memory location where the input data is stored.
    ui32Length is the length of the cryptographic data in bytes.
    pui8Tag is a pointer to a 4-word array where the hash tag is written.

This function processes data to produce a hash tag that can be used for authentication. Before calling this function, ensure that the AES module is properly configured the key, data size, mode, etc. Only CBC-MAC and F9 modes should be used.

Returns
    Returns true if data was processed successfully.
    Returns false if data processing failed.

bool AESDataProcess ( uint32_t ui32Base,
uint8_t * pui8Src,
uint8_t * pui8Dest,
uint32_t ui32Length
)

Used to process(transform) blocks of data, either encrypt or decrypt it.

Parameters

ui32Base is the base address of the AES module.
pui8Src is a pointer to the memory location where the input data is stored.
pui8Dest is a pointer to the memory location output is written.
ui32Length is the length of the cryptographic data in bytes.

This function iterates the encryption or decryption mechanism number over the data length. Before calling this function, ensure that the AES module is properly configured the key, data size, mode, etc. Only ECB, CBC, CTR, ICM, CFB, XTS and F8 operating modes should be used. The data is processed in 4-word (16-byte) blocks.

Note

This function only supports values of ui32Length less than $2^{32}$, because the memory size is restricted to between 0 to $2^{32}$ bytes.

Returns

Returns true if data was processed successfully.
Returns false if data processing failed.
bool AESDataProcessAE ( uint32_t ui32Base, 
    uint8_t * pui8Src,  
    uint8_t * pui8Dest,  
    uint32_t ui32Length,  
    uint8_t * pui8AuthSrc,  
    uint32_t ui32AuthLength,  
    uint8_t * pui8Tag )

Used for Authenticated encryption (AE) of the data. Processes and authenticates blocks of data, either encrypt the data or decrypt the data.

**Parameters**

- **ui32Base** is the base address of the AES module.
- **pui8Src** is a pointer to the memory location where the input data is stored. The data must be padded to the 16-byte boundary.
- **pui8Dest** is a pointer to the memory location output is written. The space for written data must be rounded up to the 16-byte boundary.
- **ui32Length** is the length of the cryptographic data in bytes.
- **pui8AuthSrc** is a pointer to the memory location where the additional authentication data is stored. The data must be padded to the 16-byte boundary.
ui32AuthLength is the length of the additional authentication data in bytes.

pui8Tag is a pointer to a 4-word array where the hash tag is written.

This function encrypts or decrypts blocks of data in addition to authentication data. A hash tag is also produced. Before calling this function, ensure that the AES module is properly configured the key, data size, mode, etc. Only CCM and GCM modes should be used.

Returns

Returns true if data was processed successfully.
Returns false if data processing failed.

void AESDataRead ( uint32_t ui32Base,
                   uint8_t * pui8Dest,
                   uint8_t ui8Length
                )

Reads plaintext/ciphertext from data registers with blocking. This api writes data in blocks

Parameters

ui32Base is the base address of the AES module.
pui8Dest is a pointer to an array of words.
ui8Length is the length of data in bytes to be read.
ui8Length can be from 1 to 16

This function reads a block of either plaintext or ciphertext out of the AES module. If the output is not ready, the
function waits until it is ready. A block is 16 bytes or 4 words.

Returns
None.

bool AESDataReadNonBlocking ( uint32_t ui32Base,
    uint8_t * pui8Dest,
    uint8_t ui8Length
)

Reads plaintext/ciphertext from data registers without blocking. This api writes data in blocks

Parameters
    ui32Base is the base address of the AES module.
    pui8Dest is a pointer to an array of words of data.
    ui8Length the length can be from 1 to 16

This function reads a block of either plaintext or ciphertext out of the AES module. If the output data is not ready, the function returns false. If the read completed successfully, the function returns true. A block is 16 bytes or 4 words.

Returns
true or false.

void AESDataWrite ( uint32_t ui32Base,
    uint8_t * pui8Src,
    uint8_t ui8Length
)
Wrote plaintext/ciphertext to data registers with blocking.

**Parameters**
- `ui32Base` is the base address of the AES module.
- `pui8Src` is a pointer to an array of bytes.
- `ui8Length` is the length can be from 1 to 16

This function writes a block of either plaintext or ciphertext into the AES module. If the input is not ready, the function waits until it is ready before performing the write.

**Returns**
None.

```c
bool AESDataWriteNonBlocking ( uint32_t ui32Base,
                                uint8_t * pui8Src,
                                uint8_t ui8Length
                            )
```

Wrote plaintext/ciphertext to data registers without blocking.

**Parameters**
- `ui32Base` is the base address of the AES module.
- `pui8Src` is a pointer to an array of words of data.
- `ui8Length` is the length can be from 1 to 16

This function writes a block of either plaintext or ciphertext into the AES module. If the input is not ready, the function
returns false If the write completed successfully, the function returns true.

**Returns**

True or false.

```c
void AESDMADisable ( uint32_t ui32Base,
                     uint32_t ui32Flags          
                 )
```

Disables uDMA requests for the AES module.

**Parameters**

- `ui32Base` is the base address of the AES module.
- `ui32Flags` is a bit mask of the uDMA requests to be disabled.

This function disables the uDMA request sources in the AES module. The `ui32Flags` parameter is the logical OR of any of the following:

- AES_DMA_DATA_IN
- AES_DMA_DATA_OUT
- AES_DMA_CONTEXT_IN
- AES_DMA_CONTEXT_OUT

**Returns**

None.

```c
void AESDMAEnable ( uint32_t ui32Base,
                    uint32_t ui32Flags          
               )
```
Enables uDMA requests for the AES module.

**Parameters**

- `ui32Base` is the base address of the AES module.
- `ui32Flags` is a bit mask of the uDMA requests to be enabled.

This function enables the uDMA request sources in the AES module. The `ui32Flags` parameter is the logical OR of any of the following:

- AES_DMA_DATA_IN
- AES_DMA_DATA_OUT
- AES_DMA_CONTEXT_IN
- AES_DMA_CONTEXT_OUT

**Returns**

None.

```c
void AESIntClear ( uint32_t ui32Base, uint32_t ui32IntFlags )
```

Clears AES module interrupts.

**Parameters**

- `ui32Base` is the base address of the AES module.
- `ui32IntFlags` is a bit mask of the interrupt sources to disable.
This function clears the interrupt sources in the AES module. The \textit{ui32IntFlags} parameter is the logical OR of any of the following:

- \textbf{AES\_INT\_DMA\_CONTEXT\_IN} - Context DMA done interrupt
- \textbf{AES\_INT\_DMA\_CONTEXT\_OUT} - Authentication tag (and IV) DMA done interrupt
- \textbf{AES\_INT\_DMA\_DATA\_IN} - Data input DMA done interrupt
- \textbf{AES\_INT\_DMA\_DATA\_OUT} - Data output DMA done interrupt

\textbf{Note}

Only the DMA done interrupts can be cleared. The remaining interrupts should be disabled with \texttt{AESIntDisable()}.

\textbf{Returns}

None.

\texttt{void AESIntDisable ( uint32\_t ui32Base,  
                           uint32\_t ui32IntFlags )}

Disables AES module interrupts.

\textbf{Parameters}

- \texttt{ui32Base} is the base address of the AES module.
- \texttt{ui32IntFlags} is a bit mask of the interrupt sources to disable.
This function disables the interrupt sources in the AES module. The `ui32IntFlags` parameter is the logical OR of any of the following:

- **AES_INT_CONTEXT_IN** - Context interrupt
- **AES_INT_CONTEXT_OUT** - Authentication tag (and IV) interrupt
- **AES_INT_DATA_IN** - Data input interrupt
- **AES_INT_DATA_OUT** - Data output interrupt
- **AES_INT_DMA_CONTEXT_IN** - Context DMA done interrupt
- **AES_INT_DMA_CONTEXT_OUT** - Authentication tag (and IV) DMA done interrupt
- **AES_INT_DMA_DATA_IN** - Data input DMA done interrupt
- **AES_INT_DMA_DATA_OUT** - Data output DMA done interrupt

**Note**

The DMA done interrupts are the only interrupts that can be cleared. The remaining interrupts can be disabled instead using `AESIntDisable()`.

**Returns**

None.

```c
void AESIntEnable ( uint32_t ui32Base, 
                   uint32_t ui32IntFlags )
```

Enables AES module interrupts.
Parameters

 ui32Base is the base address of the AES module.
 ui32IntFlags is a bit mask of the interrupt sources to enable.

This function enables the interrupts in the AES module. The ui32IntFlags parameter is the logical OR of any of the following:

- AES_INT_CONTEXT_IN - Context interrupt
- AES_INT_CONTEXT_OUT - Authentication tag (and IV) interrupt
- AES_INT_DATA_IN - Data input interrupt
- AES_INT_DATA_OUT - Data output interrupt
- AES_INT_DMA_CONTEXT_IN - Context DMA done interrupt
- AES_INT_DMA_CONTEXT_OUT - Authentication tag (and IV) DMA done interrupt
- AES_INT_DMA_DATA_IN - Data input DMA done interrupt
- AES_INT_DMA_DATA_OUT - Data output DMA done interrupt

Note

Interrupts that have been previously been enabled are not disabled when this function is called.

Returns

None.

void AESIntRegister (uint32_t ui32Base, 
                   void(*)(void) pfnHandler)
Registers an interrupt handler for the AES module.

**Parameters**

- **ui32Base** is the base address of the AES module.
- **pfnHandler** is a pointer to the function to be called when the enabled AES interrupts occur.

This function registers the interrupt handler in the interrupt vector table, and enables AES interrupts on the interrupt controller; specific AES interrupt sources must be enabled using `AESIntEnable()`. The interrupt handler being registered must clear the source of the interrupt using `AESIntClear()`.

If the application is using a static interrupt vector table stored in flash, then it is not necessary to register the interrupt handler this way. Instead, `IntEnable()` is used to enable AES interrupts on the interrupt controller.

**See also**

- `IntRegister()` for important information about registering interrupt handlers.

**Returns**

None.

```c
uint32_t AESIntStatus ( uint32_t ui32Base,
                        bool bMasked
)```
Returns the current AES module interrupt status.

**Parameters**

ui32Base is the base address of the AES module.

bMasked is **false** if the raw interrupt status is required and **true** if the masked interrupt status is required.

**Returns**

Returns a bit mask of the interrupt sources, which is a logical OR of any of the following:

- **AES_INT_CONTEXT_IN** - Context interrupt
- **AES_INT_CONTEXT_OUT** - Authentication tag (and IV) interrupt.
- **AES_INT_DATA_IN** - Data input interrupt
- **AES_INT_DATA_OUT** - Data output interrupt
- **AES_INT_DMA_CONTEXT_IN** - Context DMA done interrupt
- **AES_INT_DMA_CONTEXT_OUT** - Authentication tag (and IV) DMA done interrupt
- **AES_INT_DMA_DATA_IN** - Data input DMA done interrupt
- **AES_INT_DMA_DATA_OUT** - Data output DMA done interrupt

```c
void AESIntUnregister ( uint32_t ui32Base )
```

Unregisters an interrupt handler for the AES module.

**Parameters**

ui32Base is the base address of the AES module.
This function unregisters the previously registered interrupt handler and disables the interrupt in the interrupt controller.

See also

`IntRegister()` for important information about registering interrupt handlers.

Returns
None.

```c
void AESIVGet ( uint32_t ui32Base,
                  uint8_t * pui8IVdata
              )
```

Reads the Initial Vector (IV) register, needed in some of the AES Modes.

Parameters

- `ui32Base` is the base address of the AES module.
- `pui8IVdata` is pointer to an array of 16 bytes.

This functions reads the initial vector registers in the AES module.

Returns
None.

```c
void AESIVSet ( uint32_t ui32Base,
                  uint8_t * pui8IVdata
              )
```
Writes the Initial Vector (IV) register, needed in some of the AES Modes.

**Parameters**

- `ui32Base` is the base address of the AES module.
- `pui8IVdata` is an array of 16 bytes (128 bits), containing the IV value to be configured. The least significant word is in the 0th index.

This function writes the initial vector registers in the AES module.

**Returns**

None.

```c
void AESKey1Set ( uint32_t ui32Base,
                 uint8_t * pui8Key,
                 uint32_t ui32Keysize
                 )
```

Writes the key 1 configuration registers, which are used for encryption or decryption.

**Parameters**

- `ui32Base` is the base address for the AES module.
- `pui8Key` is an array of bytes, containing the key to be configured. The least significant word in the 0th index.
- `ui32Keysize` is the size of the key, which must be one
of the following values:
AES_CFG_KEY_SIZE_128,
AES_CFG_KEY_SIZE_192, or
AES_CFG_KEY_SIZE_256.

This function writes key 1 configuration registers based on
the key size. This function is used in all modes.

Returns
None.

void AESKey2Set (uint32_t ui32Base,
                 uint8_t * pui8Key,
                 uint32_t ui32Keysize)

Writes the key 2 configuration registers, which are used for
encryption or decryption.

Parameters
ui32Base is the base address for the AES module.
pui8Key is an array of bytes, containing the key to be configured. The least significant word
in the 0th index.
ui32Keysize is the size of the key, which must be one of the following values:
AES_CFG_KEY_SIZE_128,
AES_CFG_KEY_SIZE_192, or
AES_CFG_KEY_SIZE_256.

This function writes the key 2 configuration registers based
on the key size. This function is used in the F8, F9, XTS, CCM, and CBC-MAC modes.

**Returns**

None.

```c
void AESKey3Set (uint32_t ui32Base,
                 uint8_t * pui8Key)
```

Writes key 3 configuration registers, which are used for encryption or decryption.

**Parameters**

- **ui32Base** is the base address for the AES module.
- **pui8Key** is a pointer to an array bytes, containing the key to be configured. The least significant word is in the 0th index.

This function writes the key 2 configuration registers with key 3 data used in CBC-MAC and F8 modes. This key is always 128 bits.

**Returns**

None.

```c
void AESTagRead (uint32_t ui32Base,
                 uint8_t * pui8TagData)
```
Saves the tag registers to a user-defined location.

**Parameters**

- `ui32Base` is the base address of the AES module.
- `pui8TagData` is pointer to the location that stores the tag data.

This function stores the tag data for use authenticated encryption and decryption operations.

**Returns**

None.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Camera_api</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void CameraReset (ulBase)</td>
<td></td>
</tr>
<tr>
<td>void CameraParamsConfig (ulBase, ulHSPol, ulVSPol, ulFlags)</td>
<td></td>
</tr>
<tr>
<td>void CameraXClkConfig (ulBase, ulCamClkIn, ulXClk)</td>
<td></td>
</tr>
<tr>
<td>void CameraXClkSet (ulBase, bXClkFlags)</td>
<td></td>
</tr>
<tr>
<td>void CameraDMAEnable (ulBase)</td>
<td></td>
</tr>
<tr>
<td>void CameraDMADisable (ulBase)</td>
<td></td>
</tr>
<tr>
<td>void CameraThresholdSet (ulBase, ulThreshold)</td>
<td></td>
</tr>
<tr>
<td>void CameraIntRegister (ulBase, pfnHandler) (void)</td>
<td></td>
</tr>
<tr>
<td>void CameraIntUnregister (ulBase)</td>
<td></td>
</tr>
<tr>
<td>void CameraIntEnable (ulBase, ulIntFlags)</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>void CameraIntDisable()</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
</tr>
<tr>
<td>unsigned long CameraIntStatus()</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void CameraIntClear()</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
</tr>
<tr>
<td>void CameraCaptureStart()</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void CameraCaptureStop()</td>
<td>(unsigned long ulBase, tBoolean bImmediate)</td>
</tr>
<tr>
<td>void CameraBufferRead()</td>
<td>(unsigned long ulBase, unsigned long *pBuffer, unsigned char ucSize)</td>
</tr>
</tbody>
</table>
void CameraBufferRead ( unsigned long    ulBase,  
                      unsigned long * pBuffer, 
                      unsigned char    ucSize  
                    )

Reads the camera buffer (FIFO)

Parameters
  ulBase is the base address of the camera module.  
  pBuffer is the pointer to the read buffer  
  ucSize specifies the size to data to be read  

This function reads the camera buffer (FIFO).

Returns
  None.

void CameraCaptureStart ( unsigned long    ulBase  )

Starts image capture

Parameters
  ulBase is the base address of the camera module.
This function starts the image capture over the configured camera interface. This function should be called after configuring the camera module completely.

**Returns**
None.

```c
void CameraCaptureStop ( unsigned long ulBase, tBoolean blImmediate )
```

Stops image capture

**Parameters**
- `ulBase` is the base address of the camera module.
- `blImmediate` is `true` to stop capture immediately else `false`.

This function stops the image capture over the camera interface. The capture is stopped either immediately or at the end of current frame based on `blImmediate` parameter.

**Returns**
None.

```c
void CameraDMADisable ( unsigned long ulBase )
```

Disable camera DMA

**Parameters**
ulBase is the base address of the camera module.

This function masks transfer request to DMA from camera.

Returns
None.

void CameraDMAEnable ( unsigned long ulBase )

Enable camera DMA

Parameters
ulBase is the base address of the camera module.

This function enables transfer request to DMA from camera. DMA specific configuration has to be done separately.

Returns
None.

void CameraIntClear ( unsigned long ulBase, unsigned long ullIntFlags )

Clears individual camera interrupt sources.

Parameters
ulBase is the base address of the camera module.
ullIntFlags is the bit mask of the interrupt sources to be Clears.
This function clears individual camera interrupt sources.

The parameter *ullIntFlags* should be logical OR of one or more of the values as defined in **CameraIntEnable()**.

**Returns**
None.

```c
void CameraIntDisable ( unsigned long ulBase,
                         unsigned long ulIntFlags )
```

Disables individual camera interrupt sources.

**Parameters**
- **ulBase** is the base address of the camera module.
- **ullIntFlags** is the bit mask of the interrupt sources to be disabled.

This function disables individual camera interrupt sources.

The parameter *ullIntFlags* should be logical OR of one or more of the values as defined in **CameraIntEnable()**.

**Returns**
None.

```c
void CameraIntEnable ( unsigned long ulBase,
                         unsigned long ulIntFlags )
```
Enables individual camera interrupt sources.

Parameters

ulBase is the base address of the camera module.

ullIntFlags is the bit mask of the interrupt sources to be enabled.

This function enables individual camera interrupt sources.

the parameter ullIntFlags should be logical OR of one or more of the following:

- CAM_INT_DMA
- CAM_INT_FE
- CAM_INT_FSC_ERR
- CAM_INT_FIFO_NOEMPTY
- CAM_INT_FIFO_FULL
- CAM_INT_FIFO_THR
- CAM_INT_FIFO_OF
- CAN_INT_FIFO_UR

Returns

None.

```c
void CameraIntRegister( unsigned long ulBase,
                        void(*)(void) pfnHandler )
```

Register camera interrupt handler

Parameters
**ulBase** is the base address of the camera module.

**pfnHandler** hold pointer to interrupt handler

This function registers and enables global camera interrupt from the interrupt controller. Individual camera interrupts source should be enabled using

See also

**CameraIntEnable()**.

**Returns**

None.

---

```c
unsigned long CameraIntStatus ( unsigned long ulBase )
```

Returns the current interrupt status,

**Parameters**

- **ulBase** is the base address of the camera module.
- **ulBase** is the base address of the camera module.

This functions returns the current interrupt status for the camera.

**Returns**

Returns the current interrupt status, enumerated as a bit field of values described in **CameraIntEnable()**.

---

```c
void CameraIntUnregister ( unsigned long ulBase )
```
Un-Register camera interrupt handler

**Parameters**

- **ulBase** is the base address of the camera module.

This function unregisters and disables global camera interrupt from the interrupt controller.

**Returns**

None.

```c
void CameraParamsConfig ( unsigned long ulBase,  
                         unsigned long ulHSPol,  
                         unsigned long ulVSPol,  
                         unsigned long ulFlags  )
```

Configures camera parameters

**Parameters**

- **ulBase** is the base address of the camera module.
- **ulHSPol** sets the HSync polarity
- **ulVSPol** sets the VSync polarity
- **ulFlags** are configuration flags

This function sets different camera parameters.

The parameter **ulHSPol** should be on the following:

- **CAM_HS_POL_HI**
The parameter *ulVSPol* should be on the following:

- **CAM_HS_POL_LO**
- **CAM_VS_POL_HI**
- **CAM_VS_POL_LO**

The parameter *ulFlags* can be logical OR of one or more of the following or 0:

- **CAM_PCLK_RISE_EDGE**
- **CAM_PCLK_FALL_EDGE**
- **CAM_ORDERCAM_SWAP**
- **CAM_NOBT_SYNCHRO**
- **CAM_IF_SYNCHRO**

**Returns**
None.

```c
void CameraReset ( unsigned long ulBase )
```

Resets the Camera core

**Parameters**
*ulBase* is the base address of the camera module.

This function resets the camera core

**Returns**
None.
void CameraThresholdSet
(unsigned long ulThreshold
)

Sets the FIFO threshold for DMA transfer request

Parameters
  ulBase is the base address of the camera module.
  ulThreshold specifies the FIFO threshold

This function sets the FIFO threshold for DMA transfer request. Parameter *ulThreshold* can range from 1 - 64

Returns
  None.

void CameraXClkConfig (unsigned long ulBase,
  unsigned long ulCamClkIn,
  unsigned long ulXClk
)

Set the internal clock divider

Parameters
  ulBase is the base address of the camera module.
  ulCamClkIn is input to camera module
  ulXClk defines the output required

This function sets the internal clock divider based on
ulCamClkIn to generate XCLK as specified be ulXClk. Maximum suppoter division is 30

Returns
None.

void CameraXClkSet ( unsigned long ulBase,
                      unsigned char bXClkFlags
                    )

Sets the internal divide in specified mode

Parameters
  ulBase is the base address of the camera module.
  bXClkFlags decides the divide mode

This function sets the internal divide in specified mode.

The parameter bXClkFlags should be one of the following :

- CAM_XCLK_STABLE_LO
- CAM_XCLK_STABLE_HI
- CAM_XCLK_DIV_BYPASS

Returns
None.
### Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><strong>CRCCConfigSet</strong></td>
<td>(uint32_t ui32Base, uint32_t ui32CRCCConfig)</td>
</tr>
<tr>
<td>void</td>
<td><strong>CRCSeedSet</strong></td>
<td>(uint32_t ui32Base, uint32_t ui32Seed)</td>
</tr>
<tr>
<td>void</td>
<td><strong>CRCDataWrite</strong></td>
<td>(uint32_t ui32Base, uint32_t ui32Data)</td>
</tr>
<tr>
<td>uint32_t</td>
<td><strong>CRCResultRead</strong></td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td>uint32_t</td>
<td><strong>CRCDataProcess</strong></td>
<td>(uint32_t ui32Base, void *puiDataIn, uint32_t ui32DataLength, uint32_t ui32Config)</td>
</tr>
</tbody>
</table>
void CRCCConfigSet ( uint32_t ui32Base,  
                    uint32_t ui32CRCConfig  
                )

Set the configuration of CRC functionality with the EC module.

Parameters

ui32Base is the base address of the EC module.

ui32CRCConfig is the configuration of the CRC engine.

This function configures the operation of the CRC engine within the EC module. The configuration is specified with the ui32CRCConfig argument. It is the logical OR of any of the following options:

CRC Initialization Value

- **EC_CRC_CFG_INIT_SEED** - Initialize with seed value
- **EC_CRC_CFG_INIT_0** - Initialize to all '0s'
- **EC_CRC_CFG_INIT_1** - Initialize to all '1s'
Input Data Size

- **EC_CRC_CFG_SIZE_8BIT** - Input data size of 8 bits
- **EC_CRC_CFG_SIZE_32BIT** - Input data size of 32 bits

Post Process Reverse/Inverse

- **EC_CRC_CFG_RESINV** - Result inverse enable
- **EC_CRC_CFG_OBR** - Output reverse enable

Input Bit Reverse

- **EC_CRC_CFG_IBR** - Bit reverse enable

Endian Control

- **EC_CRC_CFG_ENDIAN_SBHW** - Swap byte in half-word
- **EC_CRC_CFG_ENDIAN_SHW** - Swap half-word

Operation Type

- **EC_CRC_CFG_TYPE_P8005** - Polynomial 0x8005
- **EC_CRC_CFG_TYPE_P1021** - Polynomial 0x1021
- **EC_CRC_CFG_TYPE_P4C11DB7** - Polynomial 0x4C11DB7
- **EC_CRC_CFG_TYPE_P1EDC6F41** - Polynomial 0x1EDC6F41
- **EC_CRC_CFG_TYPE_TCPCHKSUM** - TCP checksum

Returns

None.

```c
uint32_t CRCDataProcess ( uint32_t ui32Base,
```
void * puiDataIn,
uint32_t ui32DataLength,
uint32_t ui32Config
)

Process data to generate a CRC with the EC module.

Parameters

  ui32Base  is the base address of the EC module.
  puiDataIn is a pointer to an array of data that is processed.
  ui32DataLength is the number of data items that are processed to produce the CRC.
  ui32Config the config parameter to determine the CRC mode

This function processes an array of data to produce a CRC result. This function takes the CRC mode as the parameter.

The data in the array pointed to be pui32DataIn is either an array of bytes or an array or words depending on the selection of the input data size options

  EC_CRC_CFG_SIZE_8BIT and
  EC_CRC_CFG_SIZE_32BIT.

This function returns either the unmodified CRC result or the post-processed CRC result from the EC module. The post-processing options are selectable through

  EC_CRC_CFG_RESINV and EC_CRC_CFG_OBR parameters.
Returns
The CRC result.

void CRCDataWrite ( uint32_t ui32Base,
            uint32_t ui32Data
)

Write data into the EC module for CRC operations.

Parameters
  ui32Base is the base address of the EC module.
  ui32Data is the data to be written.

This function writes either 8 or 32 bits of data into the EC module for CRC operations. The distinction between 8 and 32 bits of data is made when the
EC_CRC_CFG_SIZE_8BIT or
EC_CRC_CFG_SIZE_32BIT flag is set using the
CRCCConfigSet() function.

When writing 8 bits of data, ensure the data is in the least significant byte position. The remaining bytes should be written with zero. For example, when writing 0xAB, ui32Data should be 0x000000AB.

Returns
None

uint32_t CRCResultRead ( uint32_t ui32Base )

Reads the result of a CRC operation in the EC module.
Parameters

\texttt{ui32Base} is the base address of the EC module.

This function reads either the unmodified CRC result or the post processed CRC result from the EC module. The post-processing options are selectable through \texttt{EC_CRC_CFG_RESINV} and \texttt{EC_CRC_CFG_OBR} parameters in the \texttt{CRCConfigSet()} function.

Returns

The CRC result.

\begin{verbatim}
void CRCSeedSet ( uint32_t ui32Base, 
                 uint32_t ui32Seed 
)
\end{verbatim}

Write the seed value for CRC operations in the EC module.

Parameters

\texttt{ui32Base} is the base address of the EC module. \texttt{ui32Seed} is the seed value.

This function writes the seed value for use with CRC operations in the EC module. This value is the start value for CRC operations. If this value is not written, then the residual seed from the previous operation is used as the starting value.

Note

The seed must be written only if \texttt{EC_CRC_CFG_INIT_SEED} is set with the \texttt{CRCConfigSet()} function.
## CC3200 Peripheral Driver Library User's Guide 1.2.0

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DES_Data_Encryption_Standard_api</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>void DESConfigSet (uint32_t ui32Base, uint32_t ui32Config)</th>
</tr>
</thead>
<tbody>
<tr>
<td>void DESKeySet (uint32_t ui32Base, uint8_t *pui8Key)</td>
</tr>
<tr>
<td>bool DESIVSet (uint32_t ui32Base, uint8_t *pui8IVdata)</td>
</tr>
<tr>
<td>void DESDataLengthSet (uint32_t ui32Base, uint32_t ui32Length)</td>
</tr>
<tr>
<td>bool DESDataReadNonBlocking (uint32_t ui32Base, uint8_t *pui8Dest, uint8_t ui8Length)</td>
</tr>
<tr>
<td>void DESDataRead (uint32_t ui32Base, uint8_t *pui8Dest, uint8_t ui8Length)</td>
</tr>
<tr>
<td>bool DESDataWriteNonBlocking (uint32_t ui32Base, uint8_t *pui8Src, uint8_t ui8Length)</td>
</tr>
<tr>
<td>void DESDataWrite (uint32_t ui32Base, uint8_t *pui8Src, uint8_t ui8Length)</td>
</tr>
<tr>
<td>bool DESDataProcess (uint32_t ui32Base, uint8_t *pui8Src, uint8_t *pui8Dest, uint32_t ui32Length)</td>
</tr>
<tr>
<td>uint32_t DESIntStatus (uint32_t ui32Base, bool bMasked)</td>
</tr>
<tr>
<td>void DESIntEnable (uint32_t ui32Base, uint32_t ui32IntFlags)</td>
</tr>
<tr>
<td>Function</td>
</tr>
<tr>
<td>-------------------------------</td>
</tr>
<tr>
<td><code>void DESIntDisable (uint32_t ui32Base, uint32_t ui32IntFlags)</code></td>
</tr>
<tr>
<td><code>void DESIntClear (uint32_t ui32Base, uint32_t ui32IntFlags)</code></td>
</tr>
<tr>
<td><code>void DESIntRegister (uint32_t ui32Base, void(*pfnHandler)(void))</code></td>
</tr>
<tr>
<td><code>void DESIntUnregister (uint32_t ui32Base)</code></td>
</tr>
<tr>
<td><code>void DESDMAEnable (uint32_t ui32Base, uint32_t ui32Flags)</code></td>
</tr>
<tr>
<td><code>void DESDMADisable (uint32_t ui32Base, uint32_t ui32Flags)</code></td>
</tr>
</tbody>
</table>
void DESConfigSet ( uint32_t ui32Base,
                   uint32_t ui32Config
                  )

Configures the DES module for operation.

Parameters

  ui32Base    is the base address of the DES module.
  ui32Config  is the configuration of the DES module.

This function configures the DES module for operation.

The ui32Config parameter is a bit-wise OR of a number of configuration flags. The valid flags are grouped below based on their function.

The direction of the operation is specified with one of the following two flags. Only one is permitted.

- DES_CFG_DIR_ENCRYPT - Encryption
- DES_CFG_DIR_DECRYPT - Decryption

The operational mode of the DES engine is specified with one of the following flags. Only one is permitted.
- **DES_CFG_MODE_ECB** - Electronic Codebook Mode
- **DES_CFG_MODE_CBC** - Cipher-Block Chaining Mode
- **DES_CFG_MODE_CFB** - Cipher Feedback Mode

The selection of single DES or triple DES is specified with one of the following two flags. Only one is permitted.

- **DES_CFG_SINGLE** - Single DES
- **DES_CFG_TRIPLE** - Triple DES

**Returns**
None.

```c
void DESDataLengthSet ( uint32_t ui32Base,
                        uint32_t ui32Length
                      )
```

Sets the cryptographic data length in the DES module.

**Parameters**
- **ui32Base** is the base address of the DES module.
- **ui32Length** is the length of the data in bytes.

This function writes the cryptographic data length into the DES module. When this register is written, the engine is triggered to start using this context.

**Note**
- Data lengths up to \(2^{32} - 1\) bytes are allowed.

**Returns**
None.
bool DESDataProcess ( uint32_t  ui32Base,
    uint8_t  *  pui8Src,
    uint8_t  *  pui8Dest,
    uint32_t  ui32Length
)  

Processes blocks of data through the DES module.

Parameters
    ui32Base is the base address of the DES module.
    pui8Src is a pointer to an array of words that contains the source data for processing.
    pui8Dest is a pointer to an array of words consisting of the processed data.
    ui32Length is the length of the cryptographic data in bytes. It must be a multiple of eight.

This function takes the data contained in the pui8Src array and processes it using the DES engine. The resulting data is stored in the pui8Dest array. The function blocks until all of the data has been processed. If processing is successful, the function returns true.

Note
    This functions assumes that the DES module has been configured, and initialization values and keys have been written.

Returns
    true or false.
void DESDataRead (uint32_t ui32Base,
                 uint8_t * pui8Dest,
                 uint8_t ui8Length
)

Reads plaintext/ciphertext from data registers with blocking.

Parameters

  ui32Base is the base address of the DES module.
  pui8Dest is a pointer to an array of bytes.
  ui8Length the length can be from 1 to 8

This function waits until the DES module is finished and encrypted or decrypted data is ready. The output data is then stored in the pui8Dest array.

Returns

  None

bool DESDataReadNonBlocking (uint32_t ui32Base,
                             uint8_t * pui8Dest,
                             uint8_t ui8Length
)

Reads plaintext/ciphertext from data registers without blocking

Parameters

  ui32Base is the base address of the DES module.
  pui8Dest is a pointer to an array of 2 words.
ui8Length the length can be from 1 to 8

This function returns true if the data was ready when the function was called. If the data was not ready, false is returned.

Returns
True or false.

void DESDataWrite ( uint32_t ui32Base,
                   uint8_t * pui8Src,
                   uint8_t ui8Length )

Writes plaintext/ciphertext to data registers without blocking

Parameters
ui32Base is the base address of the DES module.
pui8Src is a pointer to an array of bytes.
ui8Length the length can be from 1 to 8

This function waits until the DES module is ready before writing the data contained in the pui8Src array.

Returns
None.

bool DESDataWriteNonBlocking ( uint32_t ui32Base,
                               uint8_t * pui8Src,
                               uint8_t ui8Length )
 Writes plaintext/ciphertext to data registers without blocking

**Parameters**
- **ui32Base** is the base address of the DES module.
- **pui8Src** is a pointer to an array of 2 words.
- **ui8Length** the length can be from 1 to 8

This function returns false if the DES module is not ready to accept data. It returns true if the data was written successfully.

**Returns**
true or false.

```c
void DESDMADisable ( uint32_t ui32Base, 
                      uint32_t ui32Flags 
                   )
```

Disables DMA request sources in the DES module.

**Parameters**
- **ui32Base** is the base address of the DES module.
- **ui32Flags** is a bit mask of the DMA requests to be disabled.

This function disables DMA request sources in the DES module. The **ui32Flags** parameter should be the logical OR of any of the following:
void DESDMAEnable ( uint32_t ui32Base,
                   uint32_t ui32Flags
               )

Enables DMA request sources in the DES module.

Parameters

ui32Base is the base address of the DES module.
ui32Flags is a bit mask of the DMA requests to be enabled.

This function enables DMA request sources in the DES module. The ui32Flags parameter should be the logical OR of any of the following:

- **DES_DMA_CONTEXT_IN** - Context In
- **DES_DMA_DATA_OUT** - Data Out
- **DES_DMA_DATA_IN** - Data In

Returns

None.

void DESIntClear ( uint32_t ui32Base,
                   uint32_t ui32IntFlags
Clears interrupts in the DES module.

**Parameters**

ui32Base is the base address of the DES module.

ui32IntFlags is a bit mask of the interrupts to be disabled.

This function disables interrupt sources in the DES module. ui32IntFlags should be a logical OR of one or more of the following values:

- **DES_INT_DMA_CONTEXT_IN** - Context interrupt
- **DES_INT_DMA_DATA_IN** - Data input interrupt
- **DES_INT_DMA_DATA_OUT** - Data output interrupt

**Note**

The DMA done interrupts are the only interrupts that can be cleared. The remaining interrupts can be disabled instead using **DESIntDisable()**.

**Returns**

None.

```c
void DESIntDisable ( uint32_t ui32Base,
                     uint32_t ui32IntFlags )
```

Disables interrupts in the DES module.

**Parameters**
ui32Base is the base address of the DES module. ui32IntFlags is a bit mask of the interrupts to be disabled.

This function disables interrupt sources in the DES module. ui32IntFlags should be a logical OR of one or more of the following values:

- **DES_INT_CONTEXT_IN** - Context interrupt
- **DES_INT_DATA_IN** - Data input interrupt
- **DES_INT_DATA_OUT** - Data output interrupt
- **DES_INT_DMA_CONTEXT_IN** - Context DMA done interrupt
- **DES_INT_DMA_DATA_IN** - Data input DMA done interrupt
- **DES_INT_DMA_DATA_OUT** - Data output DMA done interrupt

**Returns**
None.

```c
void DESIntEnable ( uint32_t ui32Base,
                    uint32_t ui32IntFlags
                 )
```

Enables interrupts in the DES module.

**Parameters**
- **ui32Base** is the base address of the DES module.
- **ui32IntFlags** is a bit mask of the interrupts to be enabled.
ui32IntFlags should be a logical OR of one or more of the following values:

- **DES_INT_CONTEXT_IN** - Context interrupt
- **DES_INT_DATA_IN** - Data input interrupt
- **DES_INT_DATA_OUT** - Data output interrupt
- **DES_INT_DMA_CONTEXT_IN** - Context DMA done interrupt
- **DES_INT_DMA_DATA_IN** - Data input DMA done interrupt
- **DES_INT_DMA_DATA_OUT** - Data output DMA done interrupt

**Returns**

None.

```c
void DESIntRegister ( uint32_t ui32Base, 
                    void(*)(void) pfnHandler )
```

Registers an interrupt handler for the DES module.

**Parameters**

- **ui32Base** is the base address of the DES module.
- **pfnHandler** is a pointer to the function to be called when the enabled DES interrupts occur.

This function registers the interrupt handler in the interrupt vector table, and enables DES interrupts on the interrupt controller; specific DES interrupt sources must be enabled using **DESIntEnable()**. The interrupt handler being registered must clear the source of the interrupt using...
DESIntClear().

If the application is using a static interrupt vector table stored in flash, then it is not necessary to register the interrupt handler this way. Instead, IntEnable() should be used to enable DES interrupts on the interrupt controller.

See also
   IntRegister() for important information about registering interrupt handlers.

Returns
   None.

uint32_t DESIntStatus( uint32_t ui32Base,
                        bool bMasked
)

Returns the current interrupt status of the DES module.

Parameters
   ui32Base is the base address of the DES module.
   bMasked is false if the raw interrupt status is required
   and true if the masked interrupt status is required.

This function gets the current interrupt status of the DES module. The value returned is a logical OR of the following values:

- DES_INT_CONTEXT_IN - Context interrupt
- DES_INT_DATA_IN - Data input interrupt


- **DES_INT_DATA_OUT_INT** - Data output interrupt
- **DES_INT_DMA_CONTEXT_IN** - Context DMA done interrupt
- **DES_INT_DMA_DATA_IN** - Data input DMA done interrupt
- **DES_INT_DMA_DATA_OUT** - Data output DMA done interrupt

**Returns**

A bit mask of the current interrupt status.

```c
void DESIntUnregister ( uint32_t ui32Base )
```

Unregisters an interrupt handler for the DES module.

**Parameters**

- `ui32Base` is the base address of the DES module.

This function unregisters the previously registered interrupt handler and disables the interrupt in the interrupt controller.

**See also**

- `IntRegister()` for important information about registering interrupt handlers.

**Returns**

None.

```c
bool DESIVSet ( uint32_t ui32Base,
                uint8_t * pui8IVdata )
```
Sets the initialization vector in the DES module.

**Parameters**

- **ui32Base** is the base address of the DES module.
- **pui8IVdata** is a pointer to an array of 64 bits (2 words) of data to be written into the initialization vectors registers.

This function sets the initialization vector in the DES module. It returns true if the registers were successfully written. If the context registers cannot be written at the time the function was called, then false is returned.

**Returns**

True or false.

```c
void DESKeySet ( uint32_t ui32Base,
                 uint8_t * pui8Key
                )
```

Sets the key used for DES operations.

**Parameters**

- **ui32Base** is the base address of the DES module.
- **pui8Key** is a pointer to an array that holds the key

This function sets the key used for DES operations.

**pui8Key** should be 64 bits long (2 words) if single DES is being used or 192 bits (6 words) if triple DES is being used.

**Returns**
None.
Flash_api
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void FlashDisable()</td>
<td></td>
</tr>
<tr>
<td>long FlashErase(unsigned long ulAddress)</td>
<td></td>
</tr>
<tr>
<td>void FlashEraseNonBlocking(unsigned long ulAddress)</td>
<td></td>
</tr>
<tr>
<td>long FlashMassErase()</td>
<td></td>
</tr>
<tr>
<td>void FlashMassEraseNonBlocking()</td>
<td></td>
</tr>
<tr>
<td>long FlashProgram(unsigned long *pulData, unsigned long ulAddress, unsigned long ulCount)</td>
<td></td>
</tr>
<tr>
<td>long FlashProgramNonBlocking(unsigned long *pulData, unsigned long ulAddress, unsigned long ulCount)</td>
<td></td>
</tr>
<tr>
<td>tFlashProtection FlashProtectGet(unsigned long ulAddress)</td>
<td></td>
</tr>
<tr>
<td>void FlashIntRegister(void(*)(void))</td>
<td></td>
</tr>
<tr>
<td>void FlashIntUnregister()</td>
<td></td>
</tr>
<tr>
<td>void FlashIntEnable(unsigned long ullIntFlags)</td>
<td></td>
</tr>
</tbody>
</table>
void **FlashIntDisable** (unsigned long ullIntFlags)

unsigned long **FlashIntStatus** (tBoolean bMasked)

void **FlashIntClear** (unsigned long ullIntFlags)
void FlashDisable ( void )

Flash Disable

This function Disables the internal Flash.

Returns
   None.

long FlashErase ( unsigned long ulAddress )

Erases a block of flash.

Parameters
   ulAddress is the start address of the flash block to be erased.

This function will erase a 2 kB block of the on-chip flash. After erasing, the block will be filled with 0xFF bytes. Read-only and execute-only blocks cannot be erased.

This function will not return until the block has been erased.

Returns
Returns 0 on success, or -1 if an invalid block address was specified or the block is write-protected.

```c
void FlashEraseNonBlocking ( unsigned long ulAddress )
```

Erases a block of flash but does not wait for completion.

**Parameters**

- **ulAddress** is the start address of the flash block to be erased.

This function will erase a 2 kB block of the on-chip flash. After erasing, the block will be filled with 0xFF bytes. Read-only and execute-only blocks cannot be erased.

This function will return immediately after commanding the erase operation. Applications making use of the function can determine completion state by using a flash interrupt handler or by polling FlashIntStatus.

**Returns**

None.

```c
void FlashIntClear ( unsigned long ullIntFlags )
```

Clears flash controller interrupt sources.

**Parameters**

- **ullIntFlags** is the bit mask of the interrupt sources to be cleared. Can be any of the
The specified flash controller interrupt sources are cleared, so that they no longer assert. This must be done in the interrupt handler to keep it from being called again immediately upon exit.

**Note**

Because there is a write buffer in the Cortex-M3 processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

**Returns**

None.

```c
void FlashIntDisable ( unsigned long ulIntFlags )
```

Disables individual flash controller interrupt sources.

**Parameters**

`ulIntFlags` is a bit mask of the interrupt sources to be disabled. Can be any of the `FLASH_CTRL_PROGRAM` or `FLASH_CTRL_ACCESS` values.
Disables the indicated flash controller interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

**Returns**

None.

```c
void FlashIntEnable ( unsigned long ulIntFlags )
```

Enables individual flash controller interrupt sources.

**Parameters**

- `ulIntFlags` is a bit mask of the interrupt sources to be enabled. Can be any of the
  `FLASH_CTRL_PROGRAM` or `FLASH_CTRL_ACCESS` values.

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

**Returns**

None.

```c
void FlashIntRegister ( void(*)(void) pfnHandler )
```

Registers an interrupt handler for the flash interrupt.

**Parameters**
**pfnHandler** is a pointer to the function to be called when the flash interrupt occurs.

This sets the handler to be called when the flash interrupt occurs. The flash controller can generate an interrupt when an invalid flash access occurs, such as trying to program or erase a read-only block, or trying to read from an execute-only block. It can also generate an interrupt when a program or erase operation has completed. The interrupt will be automatically enabled when the handler is registered.

**See also**

- **IntRegister()** for important information about registering interrupt handlers.

**Returns**

None.

---

**unsigned long FlashIntStatus ( tBoolean  bMasked )**

Gets the current interrupt status.

**Parameters**

- **bMasked** is false if the raw interrupt status is required and true if the masked interrupt status is required.

This returns the interrupt status for the flash controller. Either the raw interrupt status or the status of interrupts that are allowed to reflect to the processor can be returned.
Returns
The current interrupt status, enumerated as a bit field of
FLASH_CTRL_PROGRAM and
FLASH_CTRL_ACCESS.

void FlashIntUnregister ( void )

Unregisters the interrupt handler for the flash interrupt.

This function will clear the handler to be called when the
flash interrupt occurs. This will also mask off the interrupt in
the interrupt controller so that the interrupt handler is no
longer called.

See also
    IntRegister() for important information about
    registering interrupt handlers.

Returns
    None.

long FlashMassErase ( void )

Erases a complete flash at shot.

This function erases a complete flash at shot

Returns
    Returns 0 on success, or -1 if the block is write-
    protected.
void FlashMassEraseNonBlocking ( void )

Erases a complete flash at shot but does not wait for completion.

This function will not return until the Flash has been erased.

Returns
None.

long FlashProgram ( unsigned long * pulData,
                     unsigned long  ulAddress,
                     unsigned long  ulCount )

Programs flash.

Parameters
pulData is a pointer to the data to be programmed.
ulAddress is the starting address in flash to be programmed. Must be a multiple of four.
ulCount is the number of bytes to be programmed. Must be a multiple of four.

This function will program a sequence of words into the on-chip flash. Each word in a page of flash can only be programmed one time between an erase of that page; programming a word multiple times will result in an unpredictable value in that word of flash.

Since the flash is programmed one word at a time, the
starting address and byte count must both be multiples of four. It is up to the caller to verify the programmed contents, if such verification is required.

This function will not return until the data has been programmed.

**Returns**

Returns 0 on success, or -1 if a programming error is encountered.

```c
long FlashProgramNon Blocking ( unsigned long * pulData, 
                                unsigned long ulAddress, 
                                unsigned long ulCount )
```

Programs flash but does not poll for completion.

**Parameters**

- `pulData` is a pointer to the data to be programmed.
- `ulAddress` is the starting address in flash to be programmed. Must be a multiple of four.
- `ulCount` is the number of bytes to be programmed. Must be a multiple of four.

This function will start programming one or more words into the on-chip flash and return immediately. The number of words that can be programmed in a single call depends on the part on which the function is running. For parts without support for a flash write buffer, only a single word may be
programmed on each call to this function (*ulCount* must be 1). If a write buffer is present, up to 32 words may be programmed on condition that the block being programmed does not straddle a 32 word address boundary. For example, whereas 32 words can be programmed if the address passed is 0x100 (a multiple of 128 bytes or 32 words), only 31 words could be programmed at 0x104 since attempting to write 32 would cross the 32 word boundary at 0x180.

Since the flash is programmed one word at a time, the starting address and byte count must both be multiples of four. It is up to the caller to verify the programmed contents, if such verification is required.

This function will return immediately after commanding the erase operation. Applications making use of the function can determine completion state by using a flash interrupt handler or by polling FlashIntStatus.

**Returns**

0 if the write was started successfully, -1 if there was an error.

tFlashProtection
FlashProtectGet (*unsigned long* *ulAddress*)

Gets the protection setting for a block of flash.

**Parameters**

*ulAddress* is the start address of the flash block to be queried.
This function gets the current protection for the specified 2-kB block of flash. Each block can be read/write, read-only, or execute-only. Read/write blocks can be read, executed, erased, and programmed. Read-only blocks can be read and executed. Execute-only blocks can only be executed; processor and debugger data reads are not allowed.

**Returns**

Returns the protection setting for this block. See FlashProtectSet() for possible values.
<table>
<thead>
<tr>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GPIO_General_Purpose_InputOutput_api</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void GPIODirModeSet(unsigned long ulPort, unsigned char ucPins, unsigned long ulPinIO)</code></td>
<td>Sets the direction mode for a specific port and pins.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>unsigned long GPIODirModeGet(unsigned long ulPort, unsigned char ucPin)</code></td>
<td>Gets the direction mode for a specific port and pin.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void GPIOIntTypeSet(unsigned long ulPort, unsigned char ucPins, unsigned long ullIntType)</code></td>
<td>Sets the interrupt type for a specific port and pins.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>unsigned long GPIOIntTypeGet(unsigned long ulPort, unsigned char ucPin)</code></td>
<td>Gets the interrupt type for a specific port and pin.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void GPIOIntEnable(unsigned long ulPort, unsigned long ullIntFlags)</code></td>
<td>Enables the interrupt for a specific port and flags.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void GPIOIntDisable(unsigned long ulPort, unsigned long ullIntFlags)</code></td>
<td>Disables the interrupt for a specific port and flags.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>long GPIOIntStatus(unsigned long ulPort, tBoolean bMasked)</code></td>
<td>Gets the status of the interrupt for a specific port and mask.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void GPIOIntClear(unsigned long ulPort, unsigned long ullIntFlags)</code></td>
<td>Clears the interrupt flags for a specific port.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void GPIOIntRegister(unsigned long ulPort, void(*pfnIntHandler)(void))</code></td>
<td>Registers an interrupt handler function for a specific port.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void GPIOIntUnregister</td>
<td>(unsigned long ulPort)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long GPIOPinRead</td>
<td>(unsigned long ulPort, unsigned char ucPins)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void GPIOPinWrite</td>
<td>(unsigned long ulPort, unsigned char ucPins, unsigned char ucVal)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void GPIODMATriggerEnable</td>
<td>(unsigned long ulPort)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void GPIODMATriggerDisable</td>
<td>(unsigned long ulPort)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

```
unsigned long GPIODirModeGet ( unsigned long ulPort,
                               unsigned char ucPin
                     )
```

Gets the direction and mode of a pin.

**Parameters**

- **ulPort** is the base address of the GPIO port.
- **ucPin** is the pin number.

This function gets the direction and control mode for a specified pin on the selected GPIO port. The pin can be configured as either an input or output under software control, or it can be under hardware control. The type of control and direction are returned as an enumerated data type.

**Returns**

Returns one of the enumerated data types described for `GPIODirModeSet()`.

```
void GPIODirModeSet ( unsigned long ulPort,
                      unsigned char ucPins,
                 )
```
unsigned long ulPinIO
)

Sets the direction and mode of the specified pin(s).

**Parameters**

- **ulPort** is the base address of the GPIO port
- **ucPins** is the bit-packed representation of the pin(s).
- **ulPinIO** is the pin direction and/or mode.

This function will set the specified pin(s) on the selected GPIO port as either an input or output under software control, or it will set the pin to be under hardware control.

The parameter *ulPinIO* is an enumerated data type that can be one of the following values:

- **GPIO_DIR_MODE_IN**
- **GPIO_DIR_MODE_OUT**

where **GPIO_DIR_MODE_IN** specifies that the pin will be programmed as a software controlled input,

**GPIO_DIR_MODE_OUT** specifies that the pin will be programmed as a software controlled output.

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

**Note**

**GPIOPadConfigSet()** must also be used to configure the corresponding pad(s) in order for them to propagate
the signal to/from the GPIO.

**Returns**
None.

```c
void GPIODMATriggerDisable ( unsigned long ulPort )
```

Disables a GPIO port as a trigger to start a DMA transaction.

**Parameters**
- `ulPort` is the base address of the GPIO port.

This function disables a GPIO port to be used as a trigger to start a uDMA transaction. This function can be used to disable this feature if it was enabled via a call to `GPIODMATriggerEnable()`.

**Returns**
None.

```c
void GPIODMATriggerEnable ( unsigned long ulPort )
```

Enables a GPIO port as a trigger to start a DMA transaction.

**Parameters**
- `ulPort` is the base address of the GPIO port.

This function enables a GPIO port to be used as a trigger to start a uDMA transaction. The GPIO pin will still generate
interrupts if the interrupt is enabled for the selected pin.

**Returns**
None.

```c
void GPIOIntClear ( unsigned long ulPort,
                   unsigned long ullIntFlags )
```

Clears the interrupt for the specified pin(s).

**Parameters**
- `ulPort` is the base address of the GPIO port.
- `ullIntFlags` is a bit mask of the interrupt sources to be cleared.

The `ullIntFlags` parameter has the same definition as the `ullIntFlags` parameter to `GPIOIntEnable()`.

**Returns**
None.

```c
void GPIOIntDisable ( unsigned long ulPort,
                     unsigned long ullIntFlags )
```

Disables the specified GPIO interrupts.

**Parameters**
**ulPort** is the base address of the GPIO port. **ullIntFlags** is the bit mask of the interrupt sources to disable.

This function disables the indicated GPIO interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The **ullIntFlags** parameter is the logical OR of any of the following:

- **GPIO_INT_DMA** - interrupt due to GPIO triggered DMA
- **Done**
- **GPIO_INT_PIN_0** - interrupt due to activity on Pin 0.
- **GPIO_INT_PIN_1** - interrupt due to activity on Pin 1.
- **GPIO_INT_PIN_2** - interrupt due to activity on Pin 2.
- **GPIO_INT_PIN_3** - interrupt due to activity on Pin 3.
- **GPIO_INT_PIN_4** - interrupt due to activity on Pin 4.
- **GPIO_INT_PIN_5** - interrupt due to activity on Pin 5.
- **GPIO_INT_PIN_6** - interrupt due to activity on Pin 6.
- **GPIO_INT_PIN_7** - interrupt due to activity on Pin 7.

**Returns**

None.

```c
void GPIOIntEnable ( unsigned long ulPort, unsigned long ullIntFlags )
```

Enables the specified GPIO interrupts.
Parameters

ulPort is the base address of the GPIO port.
ullIntFlags is the bit mask of the interrupt sources to enable.

This function enables the indicated GPIO interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The ullIntFlags parameter is the logical OR of any of the following:

- **GPIO_INT_DMA** - interrupt due to GPIO triggered DMA Done
- **GPIO_INT_PIN_0** - interrupt due to activity on Pin 0.
- **GPIO_INT_PIN_1** - interrupt due to activity on Pin 1.
- **GPIO_INT_PIN_2** - interrupt due to activity on Pin 2.
- **GPIO_INT_PIN_3** - interrupt due to activity on Pin 3.
- **GPIO_INT_PIN_4** - interrupt due to activity on Pin 4.
- **GPIO_INT_PIN_5** - interrupt due to activity on Pin 5.
- **GPIO_INT_PIN_6** - interrupt due to activity on Pin 6.
- **GPIO_INT_PIN_7** - interrupt due to activity on Pin 7.

Returns

None.

```c
void GPIOIntRegister ( unsigned long ulPort,
    void(*)(void) pfnIntHandler
)
```

Registers an interrupt handler for a GPIO port.
Parameters

`ulPort` is the base address of the GPIO port.

`pfnIntHandler` is a pointer to the GPIO port interrupt handling function.

This function will ensure that the interrupt handler specified by `pfnIntHandler` is called when an interrupt is detected from the selected GPIO port. This function will also enable the corresponding GPIO interrupt in the interrupt controller; individual pin interrupts and interrupt sources must be enabled with `GPIOIntEnable()`.

See also

`IntRegister()` for important information about registering interrupt handlers.

Returns

None.

```c
long GPIOIntStatus ( unsigned long ulPort,
                    tBoolean bMasked )
```

Gets interrupt status for the specified GPIO port.

Parameters

`ulPort` is the base address of the GPIO port.

`bMasked` specifies whether masked or raw interrupt status is returned.

If `bMasked` is set as `true`, then the masked interrupt status is returned; otherwise, the raw interrupt status will be
Returns

Returns the current interrupt status, enumerated as a bit field of values described in `GPIOIntEnable()`.

```c
unsigned long GPIOIntTypeGet (unsigned long ulPort,
                               unsigned char ucPin)
```

Gets the interrupt type for a pin.

Parameters

- `ulPort` is the base address of the GPIO port.
- `ucPin` is the pin number.

This function gets the interrupt type for a specified pin on the selected GPIO port. The pin can be configured as a falling edge, rising edge, or both edge detected interrupt, or it can be configured as a low level or high level detected interrupt. The type of interrupt detection mechanism is returned as an enumerated data type.

Returns

Returns one of the enumerated data types described for `GPIOIntTypeSet()`.

```c
void GPIOIntTypeSet (unsigned long ulPort,
                     unsigned char ucPins,
                     unsigned long ulIntType)
```
Sets the interrupt type for the specified pin(s).

**Parameters**

- **ulPort** is the base address of the GPIO port.
- **ucPins** is the bit-packed representation of the pin(s).
- **ullIntType** specifies the type of interrupt trigger mechanism.

This function sets up the various interrupt trigger mechanisms for the specified pin(s) on the selected GPIO port.

The parameter *ullIntType* is an enumerated data type that can be one of the following values:

- **GPIO_FALLING_EDGE**
- **GPIO_RISING_EDGE**
- **GPIO_BOTH_EDGES**
- **GPIO_LOW_LEVEL**
- **GPIO_HIGH_LEVEL**

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

**Note**

In order to avoid any spurious interrupts, the user must ensure that the GPIO inputs remain stable for the duration of this function.

**Returns**
void GPIOIntUnregister ( unsigned long ulPort )

Removes an interrupt handler for a GPIO port.

Parameters

ulPort is the base address of the GPIO port.

This function will unregister the interrupt handler for the specified GPIO port. This function will also disable the corresponding GPIO port interrupt in the interrupt controller; individual GPIO interrupts and interrupt sources must be disabled with GPIOIntDisable().

See also

IntRegister() for important information about registering interrupt handlers.

Returns

None.

long GPIOPinRead ( unsigned long ulPort, unsigned char ucPins )

Reads the values present of the specified pin(s).

Parameters

ulPort is the base address of the GPIO port.

ucPins is the bit-packed representation of the pin(s).
The values at the specified pin(s) are read, as specified by \textit{ucPins}. Values are returned for both input and output pin(s), and the value for pin(s) that are not specified by \textit{ucPins} are set to 0.

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

**Returns**

Returns a bit-packed byte providing the state of the specified pin, where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on. Any bit that is not specified by \textit{ucPins} is returned as a 0. Bits 31:8 should be ignored.

```c
void GPIOPinWrite ( unsigned long ulPort,
                    unsigned char ucPins,
                    unsigned char ucVal )
```

Writes a value to the specified pin(s).

**Parameters**

- \textbf{ulPort} is the base address of the GPIO port.
- \textbf{ucPins} is the bit-packed representation of the pin(s).
- \textbf{ucVal} is the value to write to the pin(s).

Writes the corresponding bit values to the output pin(s) specified by \textit{ucPins}. Writing to a pin configured as an input
pin has no effect.

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

**Returns**

None.
HwSpinLock_api
### Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><strong>HwSpinLockAcquire</strong></td>
<td>(uint32_t ui32LockID)</td>
</tr>
<tr>
<td>int32_t</td>
<td><strong>HwSpinLockTryAcquire</strong></td>
<td>(uint32_t ui32LockID, uint32_t ui32Retry)</td>
</tr>
<tr>
<td>void</td>
<td><strong>HwSpinLockRelease</strong></td>
<td>(uint32_t ui32LockID)</td>
</tr>
<tr>
<td>uint32_t</td>
<td><strong>HwSpinLockTest</strong></td>
<td>(uint32_t ui32LockID, bool bCurrentStatus)</td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

**void HwSpinLockAcquire ( uint32_t ui32LockID )**

Acquire specified spin lock.

**Parameters**

ui32LockID is one of the valid spin lock.

This function acquires specified spin lock and will not return until the specified lock is acquired.

The parameter ui32LockID should be **HWSPINLOCK_MCSPIS0**.

return None.

**void HwSpinLockRelease ( uint32_t ui32LockID )**

Release a previously owned spin lock.

**Parameters**

ui32LockID is one of the valid spin lock.

This function releases previously owned spin lock.
uint32_t HwSpinLockTest ( uint32_t ui32LockID, 
    bool bCurrentStatus 
) 

Get the current or previous ownership status.

Parameters
ui32LockID is one of the valid spin lock.
bCurrentStatus is true for current status, false otherwise

This function gets the current or previous ownership status of the specified spin lock based on bCurrentStatus parameter.

Returns
Returns HWSPINLOCK_OWNER_APPS, HWSPINLOCK_OWNER_NWP or HWSPINLOCK_OWNER_NONE.

int32_t HwSpinLockTryAcquire ( uint32_t ui32LockID, 
    uint32_t ui32Retry 
) 

Try to acquire specified spin lock.

Parameters
ui32LockID is one of the valid spin lock.
ui32Retry is the number of retries.

This function tries acquire specified spin lock in ui32Retry retries.

The parameter ui32Retry can be any value between 0 and 2^32.

return Returns 0 on success, -1 otherwise.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>I2C_api</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>void</th>
<th>I2CMasterInitExpClk (uint32_t ui32Base, uint32_t ui32I2CClk, bool bFast)</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>I2CSlaveInit (uint32_t ui32Base, uint8_t ui8SlaveAddr)</td>
</tr>
<tr>
<td>void</td>
<td>I2CSlaveAddressSet (uint32_t ui32Base, uint8_t ui8AddrNum, uint8_t ui8SlaveAddr)</td>
</tr>
<tr>
<td>void</td>
<td>I2CMasterEnable (uint32_t ui32Base)</td>
</tr>
<tr>
<td>void</td>
<td>I2CSlaveEnable (uint32_t ui32Base)</td>
</tr>
<tr>
<td>void</td>
<td>I2CMasterDisable (uint32_t ui32Base)</td>
</tr>
<tr>
<td>void</td>
<td>I2CSlaveDisable (uint32_t ui32Base)</td>
</tr>
<tr>
<td>void</td>
<td>I2CIntRegister (uint32_t ui32Base, void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>void</td>
<td>I2CIntUnregister (uint32_t ui32Base)</td>
</tr>
<tr>
<td>void</td>
<td>I2CMasterIntEnable (uint32_t ui32Base)</td>
</tr>
<tr>
<td>void</td>
<td>I2CMasterIntEnableEx (uint32_t ui32Base, uint32_t ui32IntFlags)</td>
</tr>
<tr>
<td>void</td>
<td>I2CSlaveIntEnable (uint32_t ui32Base)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>I2CSlaveIntEnableEx</code></td>
<td>Enable slave interrupt flag enable</td>
</tr>
<tr>
<td><code>I2CMasterIntDisable</code></td>
<td>Disable master interrupt</td>
</tr>
<tr>
<td><code>I2CMasterIntDisableEx</code></td>
<td>Disable master interrupt with flag enable</td>
</tr>
<tr>
<td><code>I2CSlaveIntDisable</code></td>
<td>Disable slave interrupt</td>
</tr>
<tr>
<td><code>I2CSlaveIntDisableEx</code></td>
<td>Disable slave interrupt with flag enable</td>
</tr>
<tr>
<td><code>I2CMasterIntStatus</code></td>
<td>Check master interrupt status with mask</td>
</tr>
<tr>
<td><code>I2CMasterIntStatusEx</code></td>
<td>Check master interrupt status with mask and flag enable</td>
</tr>
<tr>
<td><code>I2CSlaveIntStatus</code></td>
<td>Check slave interrupt status with mask</td>
</tr>
<tr>
<td><code>I2CSlaveIntStatusEx</code></td>
<td>Check slave interrupt status with mask and flag enable</td>
</tr>
<tr>
<td><code>I2CMasterIntClear</code></td>
<td>Clear master interrupt</td>
</tr>
<tr>
<td><code>I2CMasterIntClearEx</code></td>
<td>Clear master interrupt with flag enable</td>
</tr>
<tr>
<td><code>I2CSlaveIntClear</code></td>
<td>Clear slave interrupt</td>
</tr>
</tbody>
</table>

**Parameters:**
- `uint32_t ui32Base`: Base address of the I2C module.
- `uint32_t ui32IntFlags`: Interrupt flags for enabling or disabling.
- `bool bMasked`: Mask for checking or clearing the interrupt status.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void I2CSlaveIntClearEx</code></td>
<td>(uint32_t ui32Base, uint32_t ui32IntFlags)</td>
</tr>
<tr>
<td><code>void I2CMasterSlaveAddrSet</code></td>
<td>(uint32_t ui32Base, uint8_t ui8SlaveAddr, bool bReceive)</td>
</tr>
<tr>
<td><code>uint32_t I2CMasterLineStateGet</code></td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td><code>bool I2CMasterBusy</code></td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td><code>bool I2CMasterBusBusy</code></td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td><code>void I2CMasterControl</code></td>
<td>(uint32_t ui32Base, uint32_t ui32Cmd)</td>
</tr>
<tr>
<td><code>uint32_t I2CMasterErr</code></td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td><code>void I2CMasterDataPut</code></td>
<td>(uint32_t ui32Base, uint8_t ui8Data)</td>
</tr>
<tr>
<td><code>uint32_t I2CMasterDataGet</code></td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td><code>void I2CMasterTimeoutSet</code></td>
<td>(uint32_t ui32Base, uint32_t ui32Value)</td>
</tr>
<tr>
<td><code>void I2CSlaveACKOverride</code></td>
<td>(uint32_t ui32Base, bool bEnable)</td>
</tr>
<tr>
<td><code>void I2CSlaveACKValueSet</code></td>
<td>(uint32_t ui32Base, bool bACK)</td>
</tr>
<tr>
<td><code>uint32_t I2CSlaveStatus</code></td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>void I2CSlaveDataPut(uint32_t ui32Base, uint8_t ui8Data)</code></td>
<td></td>
</tr>
<tr>
<td><code>uint32_t I2CSlaveDataGet(uint32_t ui32Base)</code></td>
<td></td>
</tr>
<tr>
<td><code>void I2CTxFIFOConfigSet(uint32_t ui32Base, uint32_t ui32Config)</code></td>
<td></td>
</tr>
<tr>
<td><code>void I2CTxFIFOFlush(uint32_t ui32Base)</code></td>
<td></td>
</tr>
<tr>
<td><code>void I2CRxFIFOConfigSet(uint32_t ui32Base, uint32_t ui32Config)</code></td>
<td></td>
</tr>
<tr>
<td><code>void I2CRxFIFOFlush(uint32_t ui32Base)</code></td>
<td></td>
</tr>
<tr>
<td><code>uint32_t I2CFIFOStatus(uint32_t ui32Base)</code></td>
<td></td>
</tr>
<tr>
<td><code>void I2CFIFODataPut(uint32_t ui32Base, uint8_t ui8Data)</code></td>
<td></td>
</tr>
<tr>
<td><code>uint32_t I2CFIFODataPutNonBlocking(uint32_t ui32Base, uint8_t ui8Data)</code></td>
<td></td>
</tr>
<tr>
<td><code>uint32_t I2CFIFODataGet(uint32_t ui32Base)</code></td>
<td></td>
</tr>
<tr>
<td><code>uint32_t I2CFIFODataGetNonBlocking(uint32_t ui32Base, uint8_t *pui8Data)</code></td>
<td></td>
</tr>
<tr>
<td><code>void I2CMasterBurstLengthSet(uint32_t ui32Base, uint8_t ui8Length)</code></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><code>uint32_t I2CMasterBurstCountGet (uint32_t ui32Base)</code></td>
<td>Get the burst count for the I2C master.</td>
</tr>
<tr>
<td><code>void I2CMasterGlitchFilterConfigSet (uint32_t ui32Base, uint32_t ui32Config)</code></td>
<td>Set glitch filter configuration for the I2C master.</td>
</tr>
<tr>
<td><code>void I2CSlaveFIFOEnable (uint32_t ui32Base, uint32_t ui32Config)</code></td>
<td>Enable FIFO for the I2C slave.</td>
</tr>
<tr>
<td><code>void I2CSlaveFIFODisable (uint32_t ui32Base)</code></td>
<td>Disable FIFO for the I2C slave.</td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

**uint32_t I2CFIFODataGet ( uint32_t ui32Base )**

Reads a byte from the I2C receive FIFO.

**Parameters**

- **ui32Base** is the base address of the I2C Master or Slave module.

This function reads a byte of data from I2C receive FIFO and places it in the location specified by the *pui8Data* parameter. If there is no data available, this function waits until data is received before returning.

**Returns**

- The data byte.

**uint32_t I2CFIFODataGetNonBlocking ( uint32_t ui32Base, uint8_t * pui8Data )**

Reads a byte from the I2C receive FIFO.
Parameters

**ui32Base** is the base address of the I2C Master or Slave module.

**pui8Data** is a pointer where the read data is stored.

This function reads a byte of data from I2C receive FIFO and places it in the location specified by the **pui8Data** parameter. If there is no data available, this function returns 0.

Returns

The number of elements read from the I2C receive FIFO.

```c
void I2CFIFODataPut ( uint32_t ui32Base,
                      uint8_t ui8Data
                    )
```

Writes a data byte to the I2C transmit FIFO.

Parameters

**ui32Base** is the base address of the I2C Master or Slave module.

**ui8Data** is the data to be placed into the transmit FIFO.

This function adds a byte of data to the I2C transmit FIFO. If there is no space available in the FIFO, this function waits for space to become available before returning.

Returns

None.
uint32_t I2CFIFODataPutNonBlocking ( uint32_t ui32Base,
uint8_t ui8Data )

Writes a data byte to the I2C transmit FIFO.

Parameters
ui32Base is the base address of the I2C Master or Slave module.
ui8Data is the data to be placed into the transmit FIFO.

This function adds a byte of data to the I2C transmit FIFO. If there is no space available in the FIFO, this function returns a zero.

Returns
The number of elements added to the I2C transmit FIFO.

uint32_t I2CFIFOStatus ( uint32_t ui32Base )

Gets the current FIFO status.

Parameters
ui32Base is the base address of the I2C Master or Slave module.

This function retrieves the status for both the transmit (TX)
and receive (RX) FIFOs. The trigger level for the transmit FIFO is set using `I2CTxFIFOConfigSet()` and for the receive FIFO using `I2CTxFIFOConfigSet()`.

**Returns**

Returns the FIFO status, enumerated as a bit field containing `I2C_FIFO_RX_BELOW_TRIG_LEVEL`, `I2C_FIFO_RX_FULL`, `I2C_FIFO_RX_EMPTY`, `I2C_FIFO_TX_BELOW_TRIG_LEVEL`, `I2C_FIFO_TX_FULL`, and `I2C_FIFO_TX_EMPTY`.

```c
void I2CIntRegister ( uint32_t ui32Base,
                      void(*)(void) pfnHandler )
```

Registers an interrupt handler for the I2C module.

**Parameters**

- `ui32Base` is the base address of the I2C Master module.
- `pfnHandler` is a pointer to the function to be called when the I2C interrupt occurs.

This function sets the handler to be called when an I2C interrupt occurs. This function enables the global interrupt in the interrupt controller; specific I2C interrupts must be enabled via `I2CMasterIntEnable()` and `I2CSlaveIntEnable()`. If necessary, it is the interrupt handler's responsibility to clear the interrupt source via `I2CMasterIntClear()` and `I2CSlaveIntClear()`.

**See also**
IntRegister() for important information about registering interrupt handlers.

**Returns**
None.

```c
void I2CIntUnregister ( uint32_t ui32Base )
```

Unregisters an interrupt handler for the I2C module.

**Parameters**
- `ui32Base` is the base address of the I2C Master module.

This function clears the handler to be called when an I2C interrupt occurs. This function also masks off the interrupt in the interrupt controller so that the interrupt handler no longer is called.

**See also**
IntRegister() for important information about registering interrupt handlers.

**Returns**
None.

```c
uint32_t I2CMasterBurstCountGet ( uint32_t ui32Base )
```

Returns the current value of the burst transfer counter.

**Parameters**
ui32Base is the base address of the I2C Master module.

This function returns the current value of the burst transfer counter that is used by the FIFO mechanism. Software can use this value to determine how many bytes remain in a transfer, or where in the transfer the burst operation was if an error has occurred.

Returns
None.

void I2CMasterBurstLengthSet ( uint32_t ui32Base,
                                  uint8_t ui8Length )

Set the burst length for a I2C master FIFO operation.

Parameters
ui32Base is the base address of the I2C Master module.
ui8Length is the length of the burst transfer.

This function configures the burst length for a I2C Master FIFO operation. The burst field is limited to 8 bits or 256 bytes. The burst length applies to a single I2CMCS BURST operation meaning that it specifies the burst length for only the current operation (can be TX or RX). Each burst operation must configure the burst length prior to writing the BURST bit in the I2CMCS using I2CMasterControl().

Returns
bool I2CMasterBusBusy ( uint32_t ui32Base )

Indicates whether or not the I2C bus is busy.

Parameters

ui32Base is the base address of the I2C Master module.

This function returns an indication of whether or not the I2C bus is busy. This function can be used in a multi-master environment to determine if another master is currently using the bus.

Returns

Returns true if the I2C bus is busy; otherwise, returns false.

bool I2CMasterBusy ( uint32_t ui32Base )

Indicates whether or not the I2C Master is busy.

Parameters

ui32Base is the base address of the I2C Master module.

This function returns an indication of whether or not the I2C Master is busy transmitting or receiving data.

Returns
Returns **true** if the I2C Master is busy; otherwise, returns **false**.

```c
void I2CMasterControl ( uint32_t ui32Base,  
                        uint32_t ui32Cmd  
                      )
```

Controls the state of the I2C Master module.

**Parameters**

- **ui32Base** is the base address of the I2C Master module.
- **ui32Cmd** command to be issued to the I2C Master module.

This function is used to control the state of the Master module and receive operations. The **ui8Cmd** parameter can be one of the following values:

- **I2C_MASTER_CMD_SINGLE_SEND**
- **I2C_MASTER_CMD_SINGLE_RECEIVE**
- **I2C_MASTER_CMD_BURST_SEND_START**
- **I2C_MASTER_CMD_BURST_SEND_CONT**
- **I2C_MASTER_CMD_BURST_SEND_FINISH**
- **I2C_MASTER_CMD_BURST_SEND_ERROR_STOP**
- **I2C_MASTER_CMD_BURST_RECEIVE_START**
- **I2C_MASTER_CMD_BURST_RECEIVE_CONT**
- **I2C_MASTER_CMD_BURST_RECEIVE_FINISH**
- **I2C_MASTER_CMD_BURST_RECEIVE_ERROR_STOP**
- **I2C_MASTER_CMD_QUICK_COMMAND**
- **I2C_MASTER_CMD_HS_MASTER_CODE_SEND**
- **I2C_MASTER_CMD_FIFO_SINGLE_SEND**
- **I2C_MASTER_CMD_FIFO_SINGLE_RECEIVE**
- **I2C_MASTER_CMD_FIFO_BURST_SEND_START**
Returns
None.

```c
uint32_t I2CMasterDataGet ( uint32_t ui32Base )
```

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

**Parameters**
- `ui32Base` is the base address of the I2C Master module.

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

**Returns**
Returns the byte received from by the I2C Master, cast as an `uint32_t`.

```c
void I2CMasterDataPut ( uint32_t ui32Base,
                        uint8_t ui8Data )
```

Transmits a byte from the I2C Master.
Parameters

ui32Base is the base address of the I2C Master module.

ui8Data data to be transmitted from the I2C Master.

This function places the supplied data into I2C Master Data Register.

Returns
None.

void I2CMasterDisable (uint32_t ui32Base)

Disables the I2C master block.

Parameters

ui32Base is the base address of the I2C Master module.

This function disables operation of the I2C master block.

Returns
None.

void I2CMasterEnable (uint32_t ui32Base)

Enables the I2C Master block.

Parameters

ui32Base is the base address of the I2C Master module.
This function enables operation of the I2C Master block.

**Returns**
None.

`uint32_t I2CMasterErr ( uint32_t ui32Base )`

Gets the error status of the I2C Master module.

**Parameters**
`ui32Base` is the base address of the I2C Master module.

This function is used to obtain the error status of the Master module send and receive operations.

**Returns**
Returns the error status, as one of
I2C_MASTER_ERR_NONE,
I2C_MASTER_ERR_ADDR_ACK,
I2C_MASTER_ERR_DATA_ACK, or
I2C_MASTER_ERR_ARB_LOST.

`void I2CMasterGlitchFilterConfigSet ( uint32_t ui32Base, uint32_t ui32Config )`

Configures the I2C Master glitch filter.

**Parameters**
**ui32Base** is the base address of the I2C Master module.

**ui32Config** is the glitch filter configuration.

This function configures the I2C Master glitch filter. The value passed in to **ui32Config** determines the sampling range of the glitch filter, which is configurable between 1 and 32 system clock cycles. The default configuration of the glitch filter is 0 system clock cycles, which means that it's disabled.

The **ui32Config** field should be any of the following values:

- I2C_MASTER_GLITCH_FILTER_DISABLED
- I2C_MASTER_GLITCH_FILTER_1
- I2C_MASTER_GLITCH_FILTER_2
- I2C_MASTER_GLITCH_FILTER_3
- I2C_MASTER_GLITCH_FILTER_4
- I2C_MASTER_GLITCH_FILTER_8
- I2C_MASTER_GLITCH_FILTER_16
- I2C_MASTER_GLITCH_FILTER_32

**Returns**

None.

```c
void I2CMasterInitExpClk ( uint32_t ui32Base,
                           uint32_t ui32I2CClk,
                           bool bFast
                     )
```

Initializes the I2C Master block.
Parameters

- **ui32Base** is the base address of the I2C Master module.
- **ui32I2CClk** is the rate of the clock supplied to the I2C module.
- **bFast** set up for fast data transfers.

This function initializes operation of the I2C Master block by configuring the bus speed for the master and enabling the I2C Master block.

If the parameter **bFast** is **true**, then the master block is set up to transfer data at 400 Kbps; otherwise, it is set up to transfer data at 100 Kbps. If Fast Mode Plus (1 Mbps) is desired, software should manually write the I2CMTPR after calling this function. For High Speed (3.4 Mbps) mode, a specific command is used to switch to the faster clocks after the initial communication with the slave is done at either 100 Kbps or 400 Kbps.

The peripheral clock is the same as the processor clock. This value is returned by SysCtlClockGet(), or it can be explicitly hard coded if it is constant and known (to save the code/execution overhead of a call to SysCtlClockGet()).

**Returns**

None.

```c
void I2CMasterIntClear ( uint32_t ui32Base )
```

Clears I2C Master interrupt sources.
Parameters

ui32Base is the base address of the I2C Master module.

The I2C Master interrupt source is cleared, so that it no longer asserts. This function must be called in the interrupt handler to keep the interrupt from being triggered again immediately upon exit.

Note

Because there is a write buffer in the Cortex-M processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

Returns

None.

void I2CMasterIntClearEx ( uint32_t ui32Base,  
                         uint32_t ui32IntFlags )

Clears I2C Master interrupt sources.

Parameters

ui32Base is the base address of the I2C Master
module.

\textbf{ui32IntFlags} is a bit mask of the interrupt sources to be cleared.

The specified I2C Master interrupt sources are cleared, so that they no longer assert. This function must be called in the interrupt handler to keep the interrupt from being triggered again immediately upon exit.

The \textit{ui32IntFlags} parameter has the same definition as the \textit{ui32IntFlags} parameter to \textbf{I2CMasterIntEnableEx()}.

\textbf{Note}

Because there is a write buffer in the Cortex-M processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

\textbf{Returns}

None.

\textbf{void I2CMasterIntDisable ( uint32_t ui32Base )}

Disables the I2C Master interrupt.

\textbf{Parameters}
ui32Base is the base address of the I2C Master module.

This function disables the I2C Master interrupt source.

Returns
None.

void I2CMasterIntDisableEx ( uint32_t ui32Base, uint32_t ui32IntFlags )

Disables individual I2C Master interrupt sources.

Parameters
ui32Base is the base address of the I2C Master module.
ui32IntFlags is the bit mask of the interrupt sources to be disabled.

This function disables the indicated I2C Master interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The ui32IntFlags parameter has the same definition as the ui32IntFlags parameter to I2CMasterIntEnableEx().

Returns
None.
void I2CMasterIntEnable ( uint32_t ui32Base )

Enables the I2C Master interrupt.

Parameters
ui32Base is the base address of the I2C Master module.

This function enables the I2C Master interrupt source.

Returns
None.

void I2CMasterIntEnableEx ( uint32_t ui32Base,
                          uint32_t ui32IntFlags )

Enables individual I2C Master interrupt sources.

Parameters
ui32Base is the base address of the I2C Master module.

ui32IntFlags is the bit mask of the interrupt sources to be enabled.

This function enables the indicated I2C Master interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The ui32IntFlags parameter is the logical OR of any of the
following:

- **I2C_MASTER_INT_RX_FIFO_FULL** - RX FIFO Full interrupt
- **I2C_MASTER_INT_TX_FIFO_EMPTY** - TX FIFO Empty interrupt
- **I2C_MASTER_INT_RX_FIFO_REQ** - RX FIFO Request interrupt
- **I2C_MASTER_INT_TX_FIFO_REQ** - TX FIFO Request interrupt
- **I2C_MASTER_INT_ARB_LOST** - Arbitration Lost interrupt
- **I2C_MASTER_INT_STOP** - Stop Condition interrupt
- **I2C_MASTER_INT_START** - Start Condition interrupt
- **I2C_MASTER_INT_NACK** - Address/Data NACK interrupt
- **I2C_MASTER_INT_TX_DMA_DONE** - TX DMA Complete interrupt
- **I2C_MASTER_INT_RX_DMA_DONE** - RX DMA Complete interrupt
- **I2C_MASTER_INT_TIMEOUT** - Clock Timeout interrupt
- **I2C_MASTER_INT_DATA** - Data interrupt

Returns
None.

```c
bool I2CMasterIntStatus ( uint32_t ui32Base,
                           bool       bMasked
                     )
```

Gets the current I2C Master interrupt status.
Parameters

ui32Base is the base address of the I2C Master module.

bMasked is false if the raw interrupt status is requested and true if the masked interrupt status is requested.

This function returns the interrupt status for the I2C Master module. Either the raw interrupt status or the status of interrupts that are allowed to reflect to the processor can be returned.

Returns
The current interrupt status, returned as true if active or false if not active.

uint32_t I2CMasterIntStatusEx ( uint32_t ui32Base, bool bMasked )

Gets the current I2C Master interrupt status.

Parameters

ui32Base is the base address of the I2C Master module.

bMasked is false if the raw interrupt status is requested and true if the masked interrupt status is requested.

This function returns the interrupt status for the I2C Master module. Either the raw interrupt status or the status of
interrupts that are allowed to reflect to the processor can be returned.

Returns
Returns the current interrupt status, enumerated as a bit field of values described in I2CMasterIntEnableEx().

```c
uint32_t I2CMasterLineStateGet ( uint32_t ui32Base )
```

Reads the state of the SDA and SCL pins.

Parameters
ui32Base is the base address of the I2C Master module.

This function returns the state of the I2C bus by providing the real time values of the SDA and SCL pins.

Returns
Returns the state of the bus with SDA in bit position 1 and SCL in bit position 0.

```c
void I2CMasterSlaveAddrSet ( uint32_t ui32Base, uint8_t ui8SlaveAddr, bool bReceive )
```

Sets the address that the I2C Master places on the bus.

Parameters
ui32Base is the base address of the I2C Master module.

ui8SlaveAddr 7-bit slave address

bReceive flag indicating the type of communication with the slave

This function configures the address that the I2C Master places on the bus when initiating a transaction. When the bReceive parameter is set to true, the address indicates that the I2C Master is initiating a read from the slave; otherwise the address indicates that the I2C Master is initiating a write to the slave.

Returns
None.

void I2CMasterTimeoutSet (uint32_t ui32Base, uint32_t ui32Value)

Sets the Master clock timeout value.

Parameters
ui32Base is the base address of the I2C Master module.

ui32Value is the number of I2C clocks before the timeout is asserted.

This function enables and configures the clock low timeout feature in the I2C peripheral. This feature is implemented as a 12-bit counter, with the upper 8-bits being
programmable. For example, to program a timeout of 20ms with a 100kHz SCL frequency, \texttt{ui32Value} would be 0x7d.

Returns
None.

\begin{verbatim}
void I2CRxFIFOConfigSet ( uint32_t ui32Base,
                       uint32_t ui32Config )
\end{verbatim}

Configures the I2C receive (RX) FIFO.

Parameters
\begin{itemize}
\item \texttt{ui32Base} is the base address of the I2C Master or Slave module.
\item \texttt{ui32Config} is the configuration of the FIFO using specified macros.
\end{itemize}

This configures the I2C peripheral's receive FIFO. The receive FIFO can be used by the master or slave, but not both. The following macros are used to configure the RX FIFO behavior for master or slave, with or without DMA:

\begin{verbatim}
I2C_FIFO_CFG_RX_MASTER,
I2C_FIFO_CFG_RX_SLAVE,
I2C_FIFO_CFG_RX_MASTER_DMA,
I2C_FIFO_CFG_RX_SLAVE_DMA
\end{verbatim}

To select the trigger level, one of the following macros should be used:

\begin{verbatim}
I2C_FIFO_CFG_RX_TRIG_1,
\end{verbatim}
I2C_FIFO_CFG_RX_TRIG_2,
I2C_FIFO_CFG_RX_TRIG_3,
I2C_FIFO_CFG_RX_TRIG_4,
I2C_FIFO_CFG_RX_TRIG_5,
I2C_FIFO_CFG_RX_TRIG_6,
I2C_FIFO_CFG_RX_TRIG_7,
I2C_FIFO_CFG_RX_TRIG_8

Returns
None.

```c
void I2CRxFIFOFlush ( uint32_t ui32Base )
```

Flushes the receive (RX) FIFO.

**Parameters**

- `ui32Base` is the base address of the I2C Master or Slave module.

This function flushes the I2C receive FIFO.

**Returns**
None.

```c
void I2CSlaveACKOverride ( uint32_t ui32Base, bool bEnable )
```

Configures ACK override behavior of the I2C Slave.

**Parameters**
**ui32Base** is the base address of the I2C Slave module.

**bEnable** enables or disables ACK override.

This function enables or disables ACK override, allowing the user application to drive the value on SDA during the ACK cycle.

**Returns**

None.

```c
void I2CSlaveACKValueSet ( uint32_t ui32Base, bool bACK )
```

Writes the ACK value.

**Parameters**

- **ui32Base** is the base address of the I2C Slave module.
- **bACK** chooses whether to ACK (true) or NACK (false) the transfer.

This function puts the desired ACK value on SDA during the ACK cycle. The value written is only valid when ACK override is enabled using **I2CSlaveACKOverride()**.

**Returns**

None.

```c
void I2CSlaveAddressSet ( uint32_t ui32Base,
```
uint8_t ui8AddrNum,
uint8_t ui8SlaveAddr
)

Sets the I2C slave address.

Parameters
ui32Base is the base address of the I2C Slave module.
ui8AddrNum determines which slave address is set.
ui8SlaveAddr is the 7-bit slave address

This function writes the specified slave address. The ui32AddrNum field dictates which slave address is configured. For example, a value of 0 configures the primary address and a value of 1 configures the secondary.

Returns
None.

uint32_t I2CSlaveDataGet ( uint32_t ui32Base )

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

Parameters
ui32Base is the base address of the I2C Slave module.

This function reads a byte of data from the I2C Slave Data Register.
**Returns**
Returns the byte received from by the I2C Slave, cast as an uint32_t.

```c
void I2CSlaveDataPut ( uint32_t ui32Base, uint8_t ui8Data )
```

Transmits a byte from the I2C Slave.

**Parameters**
- `ui32Base` is the base address of the I2C Slave module.
- `ui8Data` is the data to be transmitted from the I2C Slave

This function places the supplied data into I2C Slave Data Register.

**Returns**
None.

```c
void I2CSlaveDisable ( uint32_t ui32Base )
```

Disables the I2C slave block.

**Parameters**
- `ui32Base` is the base address of the I2C Slave module.

This function disables operation of the I2C slave block.
Returns
None.

void I2CSlaveEnable ( uint32_t ui32Base )

Enables the I2C Slave block.

Parameters
ui32Base is the base address of the I2C Slave module.

This function enables operation of the I2C Slave block.

Returns
None.

void I2CSlaveFIFODisable ( uint32_t ui32Base )

Disable FIFO usage for the I2C Slave module.

Parameters
ui32Base is the base address of the I2C Slave module.

This function disables the FIFOs for the I2C Slave. After calling this function, the FIFOs are disabled, but the Slave remains active.

Returns
None.
void I2CSlaveFIFOEnable ( uint32_t ui32Base,
    uint32_t ui32Config
  )

Enables FIFO usage for the I2C Slave module.

Parameters
  ui32Base is the base address of the I2C Slave module.
  ui32Config is the desired FIFO configuration of the I2C Slave.

This function configures the I2C Slave module to use the FIFO(s). This function should be used in combination with I2CTxFIFOConfigSet() and/or I2CRxFIFOConfigSet(), which configure the FIFO trigger level and tell the FIFO hardware whether to interact with the I2C Master or Slave. The application appropriate combination of I2C_SLAVE_TX_FIFO_ENABLE and I2C_SLAVE_RX_FIFO_ENABLE should be passed in to the ui32Config field.

The Slave I2CSCSR register is write-only, so any call to I2CSlaveEnable(), I2CSlaveDisable or I2CSlaveFIFOEnable() overwrites the slave configuration. Therefore, application software should call I2CSlaveEnable() followed by I2CSlaveFIFOEnable() with the desired FIFO configuration.

Returns
  None.
void I2CSlaveInit ( uint32_t ui32Base,
               uint8_t ui8SlaveAddr
               )

Initializes the I2C Slave block.

Parameters
ui32Base is the base address of the I2C Slave module.
ui8SlaveAddr 7-bit slave address

This function initializes operation of the I2C Slave block by configuring the slave address and enabling the I2C Slave block.

The parameter ui8SlaveAddr is the value that is compared against the slave address sent by an I2C master.

Returns
None.

void I2CSlaveIntClear ( uint32_t ui32Base )

Clears I2C Slave interrupt sources.

Parameters
ui32Base is the base address of the I2C Slave module.

The I2C Slave interrupt source is cleared, so that it no longer asserts. This function must be called in the interrupt
handler to keep the interrupt from being triggered again immediately upon exit.

**Note**

Because there is a write buffer in the Cortex-M processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

**Returns**

None.

```c
void I2CSlaveIntClearEx ( uint32_t ui32Base,
                           uint32_t ui32IntFlags
                     )
```

Clears I2C Slave interrupt sources.

**Parameters**

- `ui32Base` is the base address of the I2C Slave module.
- `ui32IntFlags` is a bit mask of the interrupt sources to be cleared.

The specified I2C Slave interrupt sources are cleared, so that they no longer assert. This function must be called in
the interrupt handler to keep the interrupt from being triggered again immediately upon exit.

The `ui32IntFlags` parameter has the same definition as the `ui32IntFlags` parameter to `I2CSlaveIntEnableEx()`.

**Note**
Because there is a write buffer in the Cortex-M processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

**Returns**
None.

```c
void I2CSlaveIntDisable ( uint32_t ui32Base )
```

Disables the I2C Slave interrupt.

**Parameters**
- `ui32Base` is the base address of the I2C Slave module.

This function disables the I2C Slave interrupt source.

**Returns**
None.
void I2CSlaveIntDisableEx ( uint32_t ui32Base,  
    uint32_t ui32IntFlags  
)  

Disables individual I2C Slave interrupt sources.

Parameters

    ui32Base is the base address of the I2C Slave module.

    ui32IntFlags is the bit mask of the interrupt sources to be disabled.

This function disables the indicated I2C Slave interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The ui32IntFlags parameter has the same definition as the ui32IntFlags parameter to I2CSlaveIntEnableEx().

Returns

None.

void I2CSlaveIntEnable ( uint32_t ui32Base )

Enables the I2C Slave interrupt.

Parameters

    ui32Base is the base address of the I2C Slave module.
This function enables the I2C Slave interrupt source.

**Returns**
None.

```c
void I2CSlaveIntEnableEx ( uint32_t ui32Base,  
                          uint32_t ui32IntFlags )
```

Enables individual I2C Slave interrupt sources.

**Parameters**
- `ui32Base` is the base address of the I2C Slave module.
- `ui32IntFlags` is the bit mask of the interrupt sources to be enabled.

This function enables the indicated I2C Slave interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The `ui32IntFlags` parameter is the logical OR of any of the following:

- **I2C_SLAVE_INT_RX_FIFO_FULL** - RX FIFO Full interrupt
- **I2C_SLAVE_INT_TX_FIFO_EMPTY** - TX FIFO Empty interrupt
- **I2C_SLAVE_INT_RX_FIFO_REQ** - RX FIFO Request interrupt
- **I2C_SLAVE_INT_TX_FIFO_REQ** - TX FIFO Request interrupt
interrupt
• **I2C_SLAVE_INT_TX_DMA_DONE** - TX DMA Complete interrupt
• **I2C_SLAVE_INT_RX_DMA_DONE** - RX DMA Complete interrupt
• **I2C_SLAVE_INT_STOP** - Stop condition detected interrupt
• **I2C_SLAVE_INT_START** - Start condition detected interrupt
• **I2C_SLAVE_INT_DATA** - Data interrupt

Returns
None.

```cpp
bool I2CSlaveIntStatus ( uint32_t ui32Base, 
                        bool bMasked
)```

Gets the current I2C Slave interrupt status.

Parameters
- **ui32Base** is the base address of the I2C Slave module.
- **bMasked** is false if the raw interrupt status is requested and true if the masked interrupt status is requested.

This function returns the interrupt status for the I2C Slave module. Either the raw interrupt status or the status of interrupts that are allowed to reflect to the processor can be returned.
Returns
The current interrupt status, returned as `true` if active or `false` if not active.

```
uint32_t I2CSlaveIntStatusEx ( uint32_t ui32Base, bool bMasked )
```

Gets the current I2C Slave interrupt status.

**Parameters**
- `ui32Base` is the base address of the I2C Slave module.
- `bMasked` is false if the raw interrupt status is requested and true if the masked interrupt status is requested.

This function returns the interrupt status for the I2C Slave module. Either the raw interrupt status or the status of interrupts that are allowed to reflect to the processor can be returned.

Returns
Returns the current interrupt status, enumerated as a bit field of values described in `I2CSlaveIntEnableEx()`.

```
uint32_t I2CSlaveStatus ( uint32_t ui32Base )
```

Gets the I2C Slave module status

**Parameters**
ui32Base is the base address of the I2C Slave module.

This function returns the action requested from a master, if any. Possible values are:

- I2C_SLAVE_ACT_NONE
- I2C_SLAVE_ACT_RREQ
- I2C_SLAVE_ACT_TREQ
- I2C_SLAVE_ACT_RREQ_FBR
- I2C_SLAVE_ACT_OWN2SEL
- I2C_SLAVE_ACT_QCMD
- I2C_SLAVE_ACT_QCMD_DATA

Note
Not all devices support the second I2C slave's own address or the quick command function. Please consult the device data sheet to determine if these features are supported.

Returns
Returns I2C_SLAVE_ACT_NONE to indicate that no action has been requested of the I2C Slave module, I2C_SLAVE_ACT_RREQ to indicate that an I2C master has sent data to the I2C Slave module, I2C_SLAVE_ACT_TREQ to indicate that an I2C master has requested that the I2C Slave module send data, I2C_SLAVE_ACT_RREQ_FBR to indicate that an I2C master has sent data to the I2C slave and the first byte following the slave's own address has been received, I2C_SLAVE_ACT_OWN2SEL to indicate that the second I2C slave address was matched, I2C_SLAVE_ACT_QCMD to indicate that a quick
command was received, and
\texttt{I2C\_SLAVE\_ACT\_QCMD\_DATA} to indicate that the
data bit was set when the quick command was
received.

\begin{verbatim}
void I2CTxFIFOConfigSet ( uint32_t ui32Base,
                        uint32_t ui32Config )
\end{verbatim}

Configures the I2C transmit (TX) FIFO.

**Parameters**

- \texttt{ui32Base} is the base address of the I2C Master or
  Slave module.
- \texttt{ui32Config} is the configuration of the FIFO using
  specified macros.

This configures the I2C peripheral's transmit FIFO. The
transmit FIFO can be used by the master or slave, but not
both. The following macros are used to configure the TX
FIFO behavior for master or slave, with or without DMA:

\begin{verbatim}
I2C\_FIFO\_CFG\_TX\_MASTER,
I2C\_FIFO\_CFG\_TX\_SLAVE,
I2C\_FIFO\_CFG\_TX\_MASTER\_DMA,
I2C\_FIFO\_CFG\_TX\_SLAVE\_DMA
\end{verbatim}

To select the trigger level, one of the following macros
should be used:

\begin{verbatim}
I2C\_FIFO\_CFG\_TX\_TRIG\_1,
I2C\_FIFO\_CFG\_TX\_TRIG\_2,
\end{verbatim}
I2C_FIFO_CFG_TX_TRIG_3, I2C_FIFO_CFG_TX_TRIG_4, I2C_FIFO_CFG_TX_TRIG_5, I2C_FIFO_CFG_TX_TRIG_6, I2C_FIFO_CFG_TX_TRIG_7, I2C_FIFO_CFG_TX_TRIG_8

Returns
None.

void I2CTxFIFOFlush ( uint32_t ui32Base )

Flushes the transmit (TX) FIFO.

Parameters
ui32Base is the base address of the I2C Master or Slave module.

This function flushes the I2C transmit FIFO.

Returns
None.
CC3200 Peripheral Driver Library User's Guide 1.2.0

I2S_api
## Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><code>I2SEnable</code></td>
<td>(unsigned long ulBase, unsigned long ulMode)</td>
</tr>
<tr>
<td>void</td>
<td><code>I2SDisable</code></td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void</td>
<td><code>I2SDataPut</code></td>
<td>(unsigned long ulBase, unsigned long ulDataLine, unsigned long ulData)</td>
</tr>
<tr>
<td>long</td>
<td><code>I2SDataPutNonBlocking</code></td>
<td>(unsigned long ulBase, unsigned long ulDataLine, unsigned long ulData)</td>
</tr>
<tr>
<td>void</td>
<td><code>I2SDataGet</code></td>
<td>(unsigned long ulBase, unsigned long ulDataLine, unsigned long *pulData)</td>
</tr>
<tr>
<td>long</td>
<td><code>I2SDataGetNonBlocking</code></td>
<td>(unsigned long ulBase, unsigned long ulDataLine, unsigned long *pulData)</td>
</tr>
<tr>
<td>void</td>
<td><code>I2SConfigSetExpClk</code></td>
<td>(unsigned long ulBase, unsigned long ulI2SClk, unsigned long ulBitClk, unsigned long ulConfig)</td>
</tr>
<tr>
<td>void</td>
<td><code>I2STxFIFOEnable</code></td>
<td>(unsigned long ulBase, unsigned long ulTxLevel, unsigned long ulWordsPerTransfer)</td>
</tr>
<tr>
<td>void</td>
<td><code>I2STxFIFODisable</code></td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><code>I2SRxFIFOEnable</code></td>
<td>(unsigned long ulBase, unsigned long ulRxLevel, unsigned long ulWordsPerTransfer)</td>
<td></td>
</tr>
<tr>
<td><code>I2SRxFIFODisable</code></td>
<td>(unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td><code>I2STxFIFOSTatusGet</code></td>
<td>(unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td><code>I2SRxFIFOSTatusGet</code></td>
<td>(unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td><code>I2SSerializerConfig</code></td>
<td>(unsigned long ulBase, unsigned long ulDataLine, unsigned long ulSerMode, unsigned long ulInActState)</td>
<td></td>
</tr>
<tr>
<td><code>I2SIntEnable</code></td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
<td></td>
</tr>
<tr>
<td><code>I2SIntDisable</code></td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
<td></td>
</tr>
<tr>
<td><code>I2SIntStatus</code></td>
<td>(unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td><code>I2SIntClear</code></td>
<td>(unsigned long ulBase, unsigned long ullStatFlags)</td>
<td></td>
</tr>
<tr>
<td><code>I2SIntRegister</code></td>
<td>(unsigned long ulBase, void(*pfnHandler)(void))</td>
<td></td>
</tr>
<tr>
<td><code>I2SIntUnregister</code></td>
<td>(unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>void I2STxActiveSlotSet</td>
<td>(unsigned long ulBase, unsigned long ulActSlot)</td>
<td></td>
</tr>
<tr>
<td>void I2SRxActiveSlotSet</td>
<td>(unsigned long ulBase, unsigned long ulActSlot)</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

```c
void I2SConfigSetExpClk ( unsigned long ulBase,
                          unsigned long ul2SClk,
                          unsigned long ulBitClk,
                          unsigned long ulConfig )
```

Sets the configuration of the I2S module.

**Parameters**

- **ulBase** is the base address of the I2S module.
- **ul2SClk** is the rate of the clock supplied to the I2S module.
- **ulBitClk** is the desired bit rate.
- **ulConfig** is the data format.

This function configures the I2S for operation in the specified data format. The bit rate is provided in the **ulBitClk** parameter and the data format in the **ulConfig** parameter.

The **ulConfig** parameter is the logical OR of three values: the slot size the data read/write port select, Master or Slave mode

Follwoing selects the Master-Slave mode -
I2S_MODE_MASTER - I2S_MODE_SLAVE

Following selects the slot size: -I2S_SLOT_SIZE_24 - I2S_SLOT_SIZE_16

Following selects the data read/write port: -I2S_PORT_DMA - I2S_PORT_CPU

Returns
None.

void I2SDataGet ( unsigned long ulBase,
                 unsigned long ulDataLine,
                 unsigned long * pulData )

Waits for data from the specified data line.

Parameters
ulBase is the base address of the I2S module.
ulDataLine is one of the valid data lines.
pulData is pointer to receive data variable.

This function gets data from the receive register for the specified data line. If there are no data available, this function waits until a receive before returning.

Returns
None.

long
I2SDataGetNonBlocking (unsigned long ulBase, unsigned long ulDataLine, unsigned long *pulData)

Receives data from the specified data line.

**Parameters**
- `ulBase` is the base address of the I2S module.
- `ulDataLine` is one of the valid data lines.
- `pulData` is pointer to receive data variable.

This function gets data from the receive register for the specified data line.

**Returns**
- Returns 0 on success, -1 otherwise.

void I2SDataPut (unsigned long ulBase, unsigned long ulDataLine, unsigned long ulData)

Waits to send data over the specified data line

**Parameters**
- `ulBase` is the base address of the I2S module.
- `ulDataLine` is one of the valid data lines.
- `ulData` is the data to be transmitted.
This function sends the \textit{ucData} to the transmit register for the specified data line. If there is no space available, this function waits until there is space available before returning.

**Returns**

None.

\begin{verbatim}
long I2SDataPutNonBlocking ( unsigned long ulBase, unsigned long ulDataLine, unsigned long ulData )
\end{verbatim}

Sends data over the specified data line

**Parameters**

- \textbf{ulBase} is the base address of the I2S module.
- \textbf{ulDataLine} is one of the valid data lines.
- \textbf{ulData} is the data to be transmitted.

This function writes the \textit{ucData} to the transmit register for the specified data line. This function does not block, so if there is no space available, then \textbf{-1} is returned, and the application must retry the function later.

**Returns**

Returns 0 on success, -1 otherwise.

\begin{verbatim}
void I2SDisable ( unsigned long ulBase )
\end{verbatim}
Disables transmit and/or receive.

**Parameters**

ulBase is the base address of the I2S module.

This function disables transmit and/or receive from I2S module.

**Returns**

None.

```c
void I2SEnable ( unsigned long ulBase,  
                unsigned long ulMode  
)  
```

Enables transmit and/or receive.

**Parameters**

ulBase is the base address of the I2S module.

ulMode is one of the valid modes.

This function enables the I2S module in specified mode. The parameter *ulMode* should be one of the following

- I2S_MODE_TX_ONLY - I2S_MODE_TX_RX_SYNC

**Returns**

None.

```c
void I2SIntClear ( unsigned long ulBase,  
                   unsigned long ulStatFlags 
)  
```
Clears I2S interrupt sources.

**Parameters**

- `ulBase` is the base address of the I2S module.
- `ulStatFlags` is a bit mask of the interrupt sources to be cleared.

The specified I2S interrupt sources are cleared, so that they no longer assert. This function must be called in the interrupt handler to keep the interrupt from being recognized again immediately upon exit.

The `ullIntFlags` parameter is the logical OR of any of the value describe in `I2SIntStatus()`.

**Returns**

None.

```c
void I2SIntDisable ( unsigned long ulBase,
                    unsigned long ullIntFlags
                  )
```

Disables individual I2S interrupt sources.

**Parameters**

- `ulBase` is the base address of the I2S module.
- `ullIntFlags` is the bit mask of the interrupt sources to be disabled.
This function disables the indicated I2S interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The `ullIntFlags` parameter has the same definition as the `ullIntFlags` parameter to `I2SIntEnable()`.

**Returns**

None.

```c
void I2SIntEnable ( unsigned long ulBase,
                        unsigned long ullIntFlags )
```

Enables individual I2S interrupt sources.

**Parameters**

- `ulBase` is the base address of the I2S module.
- `ullIntFlags` is the bit mask of the interrupt sources to be enabled.

This function enables the indicated I2S interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The `ullIntFlags` parameter is the logical OR of any of the following:

- `I2S_INT_XUNDRN`
- `I2S_INT_XSYNCERR`
- `I2S_INT_XLAST`
- `I2S_INT_XDATA`
- `I2S_INT_XSTAFRM`
void I2SIntRegister ( unsigned long ulBase,  
                  void(*)(void)  pfnHandler  
               )

Registers an interrupt handler for a I2S interrupt.

Parameters
  ulBase      is the base address of the I2S module.
  pfnHandler is a pointer to the function to be called when the I2S interrupt occurs.

This function does the actual registering of the interrupt handler. This function enables the global interrupt in the interrupt controller; specific I2S interrupts must be enabled via I2SIntEnable(). It is the interrupt handler's responsibility to clear the interrupt source.

See also
  IntRegister() for important information about registering interrupt handlers.

Returns
  None.
unsigned long I2SIntStatus (unsigned long ulBase )

Gets the current interrupt status.

Parameters

  ulBase is the base address of the I2S module.

This function returns the raw interrupt status for I2S enumerated as a bit field of values:

- I2S_STS_XERR
- I2S_STS_XDMAERR
- I2S_STS_XSTAFRM
- I2S_STS_XDATA
- I2S_STS_XLAST
- I2S_STS_XSYNCERR
- I2S_STS_XUNDRN
- I2S_STS_XDMA
- I2S_STS_RERR
- I2S_STS_RDMAERR
- I2S_STS_RSTAFRM
- I2S_STS_RDATA
- I2S_STS_RLAST
- I2S_STS_RSYNCERR
- I2S_STS_ROVERN
- I2S_STS_RDMA

Returns

  Returns the current interrupt status, enumerated as a bit field of values described above.

void I2SIntUnregister (unsigned long ulBase )

Unregisters an interrupt handler for a I2S interrupt.

Parameters

  ulBase is the base address of the I2S module.

This function does the actual unregistering of the interrupt handler. It clears the handler to be called when a I2S interrupt occurs. This function also masks off the interrupt in the interrupt controller so that the interrupt handler no
longer is called.

See also
   IntRegister() for important information about
   registering interrupt handlers.

Returns
   None.

void I2SRxActiveSlotSet ( unsigned long ulBase,
                           unsigned long ulActSlot
                     )

Set the active slots for Receiver

Parameters
   ulBase  is the base address of the I2S module.
   ulActSlot is the bit-mask of active slots

This function sets the active slots for the receiver. By
default both the slots are active. The parameter ulActSlot is
logical OR following values:
- I2S_ACT_SLOT_EVEN
- I2S_ACT_SLOT_ODD

Returns
   None.

void I2SRxFIFODisable ( unsigned long ulBase )

Disables receive FIFO.
Parameters

ulBase is the base address of the I2S module.

This function disables the I2S receive FIFO.

Returns

None.

void
I2SRxFIFOEnable( unsigned long ulBase,
                 unsigned long ulRxLevel,
                 unsigned long ulWordsPerTransfer )

Configure and enable receive FIFO.

Parameters

ulBase is the base address of the I2S module.

ulRxLevel is the receive FIFO DMA request level.

ulWordsPerTransfer is the number of words transferred from the FIFO.

This function configures and enable I2S receive FIFO.

The parameter ulRxLevel sets the level at which receive DMA requests are generated. This should be non-zero integer multiple of number of serializers enabled as receivers.

The parameter ulWordsPerTransfer sets the number of
words that are transferred to the receive FIFO from the data line(s). This value must equal the number of serializers used as receivers.

**Returns**
None.

```c
unsigned long I2SRxFIFOStatusGet (unsigned long ulBase)
```

Get the receive FIFO status.

**Parameters**
- `ulBase` is the base address of the I2S module.

This function gets the number of 32-bit words currently in the receive FIFO.

**Returns**
Returns receive FIFO status.

```c
void I2SSerializerConfig (unsigned long ulBase,
                          unsigned long ulDataLine,
                          unsigned long ulSerMode,
                          unsigned long ulInActState)
```

Configure the serializer in specified mode.

**Parameters**
- `ulBase` is the base address of the I2S module.
ulDataLine is the data line (serializer) to be configured.
ulSerMode is the required serializer mode.
ullInActState sets the inactive state of the data line.

This function configure and enable the serializer associated with the given data line in specified mode.

The parameter ulDataLine selects to data line to be configured and can be one of the following: - I2S_DATA_LINE_0 - I2S_DATA_LINE_1

The parameter ulSerMode can be one of the following: - I2S_SER_MODE_TX - I2S_SER_MODE_RX - I2S_SER_MODE_DISABLE

The parameter ullInActState can be one of the following - I2S_INACT_TRI_STATE - I2S_INACT_LOW_LEVEL - I2S_INACT_LOW_HIGH

Returns
Returns receive FIFO status.

void I2STxActiveSlotSet (unsigned long ulBase,
                           unsigned long ulActSlot)

Set the active slots for Trasmitter

Parameters
ulBase is the base address of the I2S module.
ulActSlot is the bit-mask of activ slots
This function sets the active slots for the transmitter. By default both the slots are active. The parameter \textit{ulActSlot} is logical OR following values: -\texttt{I2S\_ACT\_SLOT\_EVEN} - \texttt{I2S\_ACT\_SLOT\_ODD}

\textbf{Returns}

None.

\textbf{void I2STxFIFODisable ( unsigned long \textit{ulBase} )}

Disables transmit FIFO.

\textbf{Parameters}

\texttt{ulBase} is the base address of the I2S module.

This function disables the I2S transmit FIFO.

\textbf{Returns}

None.

\textbf{void I2STxFIFOEnable ( unsigned long \textit{ulBase},
    unsigned long \textit{ulTxLevel},
    unsigned long \textit{ulWordsPerTransfer})}

Configure and enable transmit FIFO.

\textbf{Parameters}

\texttt{ulBase} is the base address of the I2S module.
**ulTxLevel** is the transmit FIFO DMA request level.

**ulWordsPerTransfer** is the number of words transferred from the FIFO.

This function configures and enables I2S transmit FIFO.

The parameter *ulTxLevel* sets the level at which transmit DMA requests are generated. This should be non-zero integer multiple of number of serializers enabled as transmitters.

The parameter *ulWordsPerTransfer* sets the number of words that are transferred from the transmit FIFO to the data line(s). This value must equal the number of serializers used as transmitters.

**Returns**

None.

```c
unsigned long I2STxFIFOStatusGet (unsigned long ulBase)
```

Get the transmit FIFO status.

**Parameters**

*ulBase* is the base address of the I2S module.

This function gets the number of 32-bit words currently in the transmit FIFO.

**Returns**
Returns transmit FIFO status.
## Interrupt_api

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
</table>

This section provides information on the Interrupt_api module, including details on how to use it in the CC3200 Peripheral Driver Library User's Guide.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tBoolean IntMasterEnable</td>
<td>(void)</td>
</tr>
<tr>
<td>tBoolean IntMasterDisable</td>
<td>(void)</td>
</tr>
<tr>
<td>void IntVTableBaseSet</td>
<td>(unsigned long ulVtableBase)</td>
</tr>
<tr>
<td>void IntRegister</td>
<td>(unsigned long ullInterrupt, void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>void IntUnregister</td>
<td>(unsigned long ullInterrupt)</td>
</tr>
<tr>
<td>void IntPriorityGroupingSet</td>
<td>(unsigned long ulBits)</td>
</tr>
<tr>
<td>unsigned long IntPriorityGroupingGet</td>
<td>(void)</td>
</tr>
<tr>
<td>void IntPrioritySet</td>
<td>(unsigned long ullInterrupt, unsigned char ucPriority)</td>
</tr>
<tr>
<td>long IntPriorityGet</td>
<td>(unsigned long ullInterrupt)</td>
</tr>
<tr>
<td>void IntEnable</td>
<td>(unsigned long ullInterrupt)</td>
</tr>
<tr>
<td>void IntDisable</td>
<td>(unsigned long ullInterrupt)</td>
</tr>
<tr>
<td>void IntPendSet</td>
<td>(unsigned long ullInterrupt)</td>
</tr>
<tr>
<td>void IntPendClear</td>
<td>(unsigned long ullInterrupt)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>void <code>IntPriorityMaskSet</code> (unsigned long ulPriorityMask)</td>
<td></td>
</tr>
<tr>
<td>unsigned long <code>IntPriorityMaskGet</code> (void)</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

void IntDisable ( unsigned long ullInterrupt )

Disables an interrupt.

Parameters

ullInterrupt specifies the interrupt to be disabled.

The specified interrupt is disabled in the interrupt controller. Other enables for the interrupt (such as at the peripheral level) are unaffected by this function.

Returns

None.

void IntEnable ( unsigned long ullInterrupt )

Enables an interrupt.

Parameters

ullInterrupt specifies the interrupt to be enabled.

The specified interrupt is enabled in the interrupt controller. Other enables for the interrupt (such as at the peripheral level) are unaffected by this function.
level) are unaffected by this function.

**Returns**
None.

**tBoolean IntMasterDisable (void )**

Disables the processor interrupt.

Prevents the processor from receiving interrupts. This does not affect the set of interrupts enabled in the interrupt controller; it just gates the single interrupt from the controller to the processor.

**Note**
Previously, this function had no return value. As such, it was possible to include `interrupt.h` and call this function without having included `hw_types.h`. Now that the return is a `tBoolean`, a compiler error will occur in this case. The solution is to include `hw_types.h` before including `interrupt.h`.

**Returns**
Returns **true** if interrupts were already disabled when the function was called or **false** if they were initially enabled.

**tBoolean IntMasterEnable (void )**

Enables the processor interrupt.

Allows the processor to respond to interrupts. This does not
affect the set of interrupts enabled in the interrupt controller; it just gates the single interrupt from the controller to the processor.

**Note**
Previously, this function had no return value. As such, it was possible to include `interrupt.h` and call this function without having included `hw_types.h`. Now that the return is a `tBoolean`, a compiler error will occur in this case. The solution is to include `hw_types.h` before including `interrupt.h`.

**Returns**
Returns **true** if interrupts were disabled when the function was called or **false** if they were initially enabled.

```c
void IntPendClear (unsigned long ullInterrupt)
```

Unpends an interrupt.

**Parameters**

- `ullInterrupt` specifies the interrupt to be unpended.

The specified interrupt is unpended in the interrupt controller. This will cause any previously generated interrupts that have not been handled yet (due to higher priority interrupts or the interrupt no having been enabled yet) to be discarded.

**Returns**
None.
void IntPendSet( unsigned long ulInterrupt )

Pends an interrupt.

Parameters

ulInterrupt specifies the interrupt to be pended.

The specified interrupt is pended in the interrupt controller. This will cause the interrupt controller to execute the corresponding interrupt handler at the next available time, based on the current interrupt state priorities. For example, if called by a higher priority interrupt handler, the specified interrupt handler will not be called until after the current interrupt handler has completed execution. The interrupt must have been enabled for it to be called.

Returns

None.

long IntPriorityGet( unsigned long ulInterrupt )

Gets the priority of an interrupt.

Parameters

ulInterrupt specifies the interrupt in question.

This function gets the priority of an interrupt. See IntPrioritySet() for a definition of the priority value.

Returns

Returns the interrupt priority, or -1 if an invalid interrupt
was specified.

`unsigned long IntPriorityGroupingGet ( void )`

Gets the priority grouping of the interrupt controller.

This function returns the split between preemptable priority levels and subpriority levels in the interrupt priority specification.

**Returns**

The number of bits of preemptable priority.

`void IntPriorityGroupingSet ( unsigned long ulBits )`

Sets the priority grouping of the interrupt controller.

**Parameters**

`ulBits` specifies the number of bits of preemptable priority.

This function specifies the split between preemptable priority levels and subpriority levels in the interrupt priority specification. The range of the grouping values are dependent upon the hardware implementation; on the CC3200, three bits are available for hardware interrupt prioritization and therefore priority grouping values of three through seven have the same effect.

**Returns**

None.
unsigned long IntPriorityMaskGet ( void )

Gets the priority masking level

This function gets the current setting of the interrupt priority masking level. The value returned is the priority level such that all interrupts of that and lesser priority are masked. A value of 0 means that priority masking is disabled.

Smaller numbers correspond to higher interrupt priorities. So for example a priority level mask of 4 will allow interrupts of priority level 0-3, and interrupts with a numerical priority of 4 and greater will be blocked.

The hardware priority mechanism will only look at the upper N bits of the priority level (where N is 3), so any prioritization must be performed in those bits.

Returns
Returns the value of the interrupt priority level mask.

void IntPriorityMaskSet ( unsigned long ulPriorityMask )

Sets the priority masking level

Parameters
ulPriorityMask is the priority level that will be masked.

This function sets the interrupt priority masking level so that all interrupts at the specified or lesser priority level is masked. This can be used to globally disable a set of
Interrupts with priority below a predetermined threshold. A value of 0 disables priority masking.

Smaller numbers correspond to higher interrupt priorities. So for example a priority level mask of 4 will allow interrupts of priority level 0-3, and interrupts with a numerical priority of 4 and greater will be blocked.

The hardware priority mechanism will only look at the upper N bits of the priority level (where N is 3), so any prioritization must be performed in those bits.

**Returns**

None.

```c
void IntPrioritySet ( unsigned long ullInterrupt,
                     unsigned char ucPriority )
```

Sets the priority of an interrupt.

**Parameters**

- `ullInterrupt` specifies the interrupt in question.
- `ucPriority` specifies the priority of the interrupt.

This function is used to set the priority of an interrupt. When multiple interrupts are asserted simultaneously, the ones with the highest priority are processed before the lower priority interrupts. Smaller numbers correspond to higher interrupt priorities; priority 0 is the highest interrupt priority.

The hardware priority mechanism will only look at the upper
N bits of the priority level (where N is 3), so any prioritization must be performed in those bits. The remaining bits can be used to sub-prioritize the interrupt sources, and may be used by the hardware priority mechanism on a future part. This arrangement allows priorities to migrate to different NVIC implementations without changing the gross prioritization of the interrupts.

The parameter `ucPriority` can be any one of the following:

- INT_PRIORITY_LVL_0
- INT_PRIORITY_LVL_1
- INT_PRIORITY_LVL_2
- INT_PRIORITY_LVL_3
- INT_PRIORITY_LVL_4
- INT_PRIORITY_LVL_5
- INT_PRIORITY_LVL_6
- INT_PRIORITY_LVL_7

**Returns**

None.

```c
void IntRegister ( unsigned long ullInterrupt, 
                void(*)(void) pfnHandler 
                )
```

Registers a function to be called when an interrupt occurs.

**Parameters**

- `ullInterrupt` specifies the interrupt in question.
- `pfnHandler` is a pointer to the function to be called.

This function is used to specify the handler function to be called when the given interrupt is asserted to the processor. When the interrupt occurs, if it is enabled (via `IntEnable()`), the handler function will be called in interrupt context. Since the handler function can preempt other code, care must be
taken to protect memory or peripherals that are accessed by the handler and other non-handler code.

**Returns**

None.

```c
void IntUnregister ( unsigned long ullInterrupt )
```

Unregisters the function to be called when an interrupt occurs.

**Parameters**

- `ullInterrupt` specifies the interrupt in question.

This function is used to indicate that no handler should be called when the given interrupt is asserted to the processor. The interrupt source will be automatically disabled (via `IntDisable()`) if necessary.

**See also**

- `IntRegister()` for important information about registering interrupt handlers.

**Returns**

None.

```c
void IntVTableBaseSet ( unsigned long ulVtableBase )
```

Sets the NVIC VTable base.

**Parameters**
ulVtableBase specifies the new base address of VTable

This function is used to specify a new base address for the VTable. This function must be called before using IntRegister() for registering any interrupt handler.

Returns
None.
<table>
<thead>
<tr>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin_api</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Function Name</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><strong>PinModeSet</strong></td>
<td>(unsigned long <code>ulPin</code>, unsigned long <code>ulPinMode</code>)</td>
</tr>
<tr>
<td>unsigned long</td>
<td><strong>PinModeGet</strong></td>
<td>(unsigned long <code>ulPin</code>)</td>
</tr>
<tr>
<td>void</td>
<td><strong>PinDirModeSet</strong></td>
<td>(unsigned long <code>ulPin</code>, unsigned long <code>ulPinIO</code>)</td>
</tr>
<tr>
<td>unsigned long</td>
<td><strong>PinDirModeGet</strong></td>
<td>(unsigned long <code>ulPin</code>)</td>
</tr>
<tr>
<td>void</td>
<td><strong>PinConfigGet</strong></td>
<td>(unsigned long <code>ulPin</code>, unsigned long *pulPinStrength, unsigned long *pulPinType)</td>
</tr>
<tr>
<td>void</td>
<td><strong>PinConfigSet</strong></td>
<td>(unsigned long <code>ulPin</code>, unsigned long <code>ulPinStrength</code>, unsigned long <code>ulPinType</code>)</td>
</tr>
<tr>
<td>void</td>
<td><strong>PinTypeUART</strong></td>
<td>(unsigned long <code>ulPin</code>, unsigned long <code>ulPinMode</code>)</td>
</tr>
<tr>
<td>void</td>
<td><strong>PinTypeI2C</strong></td>
<td>(unsigned long <code>ulPin</code>, unsigned long <code>ulPinMode</code>)</td>
</tr>
<tr>
<td>void</td>
<td><strong>PinTypeSPI</strong></td>
<td>(unsigned long <code>ulPin</code>, unsigned long <code>ulPinMode</code>)</td>
</tr>
<tr>
<td>void</td>
<td><strong>PinTypeI2S</strong></td>
<td>(unsigned long <code>ulPin</code>, unsigned long <code>ulPinMode</code>)</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>void PinTypeTimer</td>
<td>(unsigned long ulPin, unsigned long ulPinMode)</td>
<td></td>
</tr>
<tr>
<td>void PinTypeCamera</td>
<td>(unsigned long ulPin, unsigned long ulPinMode)</td>
<td></td>
</tr>
<tr>
<td>void PinTypeGPIO</td>
<td>(unsigned long ulPin, unsigned long ulPinMode, tBoolean bOpenDrain)</td>
<td></td>
</tr>
<tr>
<td>void PinTypeADC</td>
<td>(unsigned long ulPin, unsigned long ulPinMode)</td>
<td></td>
</tr>
<tr>
<td>void PinTypeSDHost</td>
<td>(unsigned long ulPin, unsigned long ulPinMode)</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

void PinConfigGet ( unsigned long ulPin,  
    unsigned long * pulPinStrength,  
    unsigned long * pulPinType  
  )

Gets Pin output drive strength and Type

Parameters

  ulPin is one of the valid pin
  pulPinStrength is pointer to storage for output drive strength
  pulPinType is pointer to storage for pin type

This function gets the pin type and output drive strength for the pin specified by ulPin parameter. Parameters pulPinStrength and pulPinType corresponds to the values used in PinConfigSet().

Returns

  None.

void PinConfigSet ( unsigned long ulPin,  
    unsigned long ulPinStrength,  
    unsigned long ulPinType  
  )

Gets Pin output drive strength and Type

Parameters

  ulPin is one of the valid pin
  pulPinStrength is pointer to storage for output drive strength
  pulPinType is pointer to storage for pin type

This function gets the pin type and output drive strength for the pin specified by ulPin parameter. Parameters pulPinStrength and pulPinType corresponds to the values used in PinConfigSet().

Returns

  None.
Configure Pin output drive strength and Type

**Parameters**

- `ulPin` is one of the valid pin
- `ulPinStrength` is logical OR of valid output drive strengths.
- `ulPinType` is one of the valid pin type.

This function sets the pin type and strength for the pin specified by `ulPin` parameter.

The parameter `ulPinStrength` should be one of the following

- `PIN_STRENGTH_2MA`
- `PIN_STRENGTH_4MA`
- `PIN_STRENGTH_6MA`

The parameter `ulPinType` should be one of the following

For standard type

- `PIN_TYPE_STD`
- `PIN_TYPE_STD_PU`
- `PIN_TYPE_STD_PD`

And for Open drain type

- `PIN_TYPE_OD`
- `PIN_TYPE_OD_PU`
- `PIN_TYPE_OD_PD`
Returns
None.

unsigned long PinDirModeGet ( unsigned long ulPin )

Gets the direction of a pin.

Parameters
ulPin is one of the valid pin.

This function gets the direction and control mode for a specified pin on the selected GPIO port. The pin can be configured as either an input only or output only, or it can be under hardware control. The type of control and direction are returned as an enumerated data type.

Returns
Returns one of the enumerated data types described for GPIODirModeSet().

void PinDirModeSet ( unsigned long ulPin, unsigned long ulPinIO )

Sets the direction of the specified pin(s).

Parameters
ulPin is one of the valid pin.
ulPinIO is the pin direction and/or mode.

This function configures the specified pin(s) as either input
only or output only or it configures the pin to be under hardware control.

The parameter \textit{ulPinIO} is an enumerated data type that can be one of the following values:

- PIN\_DIR\_MODE\_IN
- PIN\_DIR\_MODE\_OUT
- PIN\_DIR\_MODE\_HW

where PIN\_DIR\_MODE\_IN specifies that the pin is programmed as a input only, PIN\_DIR\_MODE\_OUT specifies that the pin is programmed output only, and PIN\_DIR\_MODE\_HW specifies that the pin is placed under hardware control.

**Retruns**

None.

\begin{verbatim}
unsigned long PinModeGet ( unsigned long ulPin )
\end{verbatim}

Gets current pin mux configuration of specified pin.

**Parameters**

\textbf{ulPin} is a valid pin.

This function get the current configuration of the pin mux.

**Returns**

Returns current pin mode if \textit{ulPin} is valid, 0xFF otherwise.
void PinModeSet (unsigned long ulPin,
             unsigned long ulPinMode)

Configures pin mux for the specified pin.

**Parameters**

- **ulPin** is a valid pin.
- **ulPinMode** is one of the valid mode

This function configures the pin mux that selects the peripheral function associated with a particular SOC pin. Only one peripheral function at a time can be associated with a pin, and each peripheral function should only be associated with a single pin at a time.

**Returns**

- none

void PinTypeADC (unsigned long ulPin,
                 unsigned long ulPinMode)

Sets the pin mode and configures the pin for use by ADC.

**Parameters**

- **ulPin** is one of the valid pin.
- **ulPinMode** is one of the valid pin mode.

The ADC pins must be properly configured for the peripheral to function correctly. This function provides a
typical configuration for those pin.

**Note**
This function cannot be used to turn any pin into a ADC pin; it only sets the pin mode and configures it for proper ADC operation.

**Returns**
None.

```c
void PinTypeCamera ( unsigned long ulPin,  
                    unsigned long ulPinMode  )
```

Sets the pin mode and configures the pin for use by Camera peripheral

**Parameters**
- **ulPin** is one of the valid pin.
- **ulPinMode** is one of the valid pin mode.

The Camera pins must be properly configured for the peripheral to function correctly. This function provides a typical configuration for those pin.

**Note**
This function cannot be used to turn any pin into a Camera pin; it only sets the pin mode and configures it for proper Camera operation.

**Returns**
None.
void PinTypeGPIO ( unsigned long ulPin, 
    unsigned long ulPinMode, 
    tBoolean bOpenDrain 
)

Sets the pin mode and configures the pin for use by GPIO peripheral

Parameters
    ulPin    is one of the valid pin.
    ulPinMode is one of the valid pin mode.
    bOpenDrain is one to decide either OpenDrain or STD

The GPIO pins must be properly configured for the peripheral to function correctly. This function provides a typical configuration for those pin.

Returns
    None.

void PinTypeI2C ( unsigned long ulPin, 
                unsigned long ulPinMode 
)

Sets the pin mode and configures the pin for use by I2C peripheral

Parameters
    ulPin    is one of the valid pin.
**ulPinMode** is one of the valid pin mode.

The I2C pins must be properly configured for the peripheral to function correctly. This function provides a typical configuration for the pin.

**Note**
This function cannot be used to turn any pin into a I2C pin; it only sets the pin mode and configures it for proper I2C operation.

**Returns**
None.

```c
void PinTypeI2S ( unsigned long ulPin,
                 unsigned long ulPinMode )
```

Sets the pin mode and configures the pin for use by I2S peripheral

**Parameters**
- **ulPin** is one of the valid pin.
- **ulPinMode** is one of the valid pin mode.

The I2S pins must be properly configured for the peripheral to function correctly. This function provides a typical configuration for those pin.

**Note**
This function cannot be used to turn any pin into a I2S pin; it only sets the pin mode and configures it for
proper I2S operation.

**Returns**

None.

```c
void PinTypeSDHost ( unsigned long ulPin,
                    unsigned long ulPinMode
                )
```

Sets the pin mode and configures the pin for use by SD Host peripheral

**Parameters**

- `ulPin` is one of the valid pin.
- `ulPinMode` is one of the valid pin mode.

The MMC pins must be properly configured for the peripheral to function correctly. This function provides a typical configuration for those pin.

**Note**

This function cannot be used to turn any pin into a SD Host pin; it only sets the pin mode and configures it for proper SD Host operation.

**Returns**

None.

```c
void PinTypeSPI ( unsigned long ulPin,
                 unsigned long ulPinMode
             )
```
Sets the pin mode and configures the pin for use by SPI peripheral

**Parameters**

- `ulPin` is one of the valid pin.
- `ulPinMode` is one of the valid pin mode.

The SPI pins must be properly configured for the peripheral to function correctly. This function provides a typical configuration for those pin.

**Note**

This function cannot be used to turn any pin into a SPI pin; it only sets the pin mode and configures it for proper SPI operation.

**Returns**

None.

```c
void PinTypeTimer ( unsigned long  ulPin,
                    unsigned long ulPinMode )
```

Sets the pin mode and configures the pin for use by Timer peripheral

**Parameters**

- `ulPin` is one of the valid pin.
- `ulPinMode` is one of the valid pin mode.

The timer PWM pins must be properly configured for the
Timer peripheral to function correctly. This function provides a typical configuration for those pin; other configurations may work as well depending upon the board setup (for example, using the on-chip pull-ups).

**Note**

This function cannot be used to turn any pin into a timer PWM pin; it only sets the pin mode and configures it for proper timer PWM operation.

**Returns**

None.

```c
void PinTypeUART (
    unsigned long ulPin,
    unsigned long ulPinMode
);
```

Sets the pin mode and configures the pin for use by UART peripheral.

**Parameters**

- **ulPin** is one of the valid pin.
- **ulPinMode** is one of the valid pin mode.

The UART pins must be properly configured for the peripheral to function correctly. This function provides a typical configuration for those pin(s); other configurations may work as well depending upon the board setup (for example, using the on-chip pull-ups).

**Note**

This function cannot be used to turn any pin into a...
UART pin; it only sets the pin mode and configures it for proper UART operation.

Returns
None.
### PRCM_Power_Reset_Clock_Module_api

**Module:** PRCM

**Description:**

This module provides APIs for managing power and clock resources on the CC3200 microcontroller.

**Functions:**

- `prcm_power_reset_clock_module_init()`: Initializes the PRCM module.
- `prcm_power_reset_clock_module_start()`: Starts the PRCM module.
- `prcm_power_reset_clock_module_stop()`: Stops the PRCM module.
- `prcm_power_reset_clock_module_reset()`: Resets the PRCM module.

**Usage Example:**

```c
PRCM_Power_Reset_Clock_Module_api
```
## Functions

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><code>PRCMMCUReset</code></td>
<td>(tBoolean blIncludeSubsystem)</td>
</tr>
<tr>
<td>unsigned long</td>
<td><code>PRCMSysResetCauseGet</code></td>
<td>()</td>
</tr>
<tr>
<td>void</td>
<td><code>PRCMPeripheralClkEnable</code></td>
<td>(unsigned long ulPeripheral, unsigned long ulClkFlags)</td>
</tr>
<tr>
<td>void</td>
<td><code>PRCMPeripheralClkDisable</code></td>
<td>(unsigned long ulPeripheral, unsigned long ulClkFlags)</td>
</tr>
<tr>
<td>unsigned long</td>
<td><code>PRCMPeripheralClockGet</code></td>
<td>(unsigned long ulPeripheral)</td>
</tr>
<tr>
<td>void</td>
<td><code>PRCMPeripheralReset</code></td>
<td>(unsigned long ulPeripheral)</td>
</tr>
<tr>
<td>tBoolean</td>
<td><code>PRCMPeripheralStatusGet</code></td>
<td>(unsigned long ulPeripheral)</td>
</tr>
<tr>
<td>void</td>
<td><code>PRCMI2SClockFreqSet</code></td>
<td>(unsigned long ulI2CClkFreq)</td>
</tr>
<tr>
<td>void</td>
<td><code>PRCMLPDSRestoreInfoSet</code></td>
<td>(unsigned long ulStackPtr, unsigned long ulProgCntr)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>PRCMLPDSEnter()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMLPDSEnterKeepDebugIf()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMLPDSWakeupSourceEnable(ulLpdsWakeupSrc)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMLPDSWakeupSourceDisable(ulLpdsWakeupSrc)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMLPDSWakeupCauseGet()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMLPDSIntervalSet(ulTicks)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMLPDSWakeUpGPIOSelect(ulGPIOPin, ulType)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMSleepEnter()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMSRAMRetentionEnable(ulSramColSel, ulModeFlags)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMSRAMRetentionDisable(ulSramColSel, ulFlags)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRCMHibernateWakeupSourceEnable(ulHIBWakupSrc)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Prototype</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PRCMHibernateWakeupSourceDisable</td>
<td>void PRCMHibernateWakeupSourceDisable(unsigned long ulHIBWakupSrc)</td>
<td>Disable the wakeup source.</td>
</tr>
<tr>
<td>PRCMHibernateWakeupCauseGet</td>
<td>unsigned long PRCMHibernateWakeupCauseGet()</td>
<td>Get the cause of the wakeup.</td>
</tr>
<tr>
<td>PRCMHibernateIntervalSet</td>
<td>void PRCMHibernateIntervalSet(unsigned long ullTicks)</td>
<td>Set the interval for the wakeup.</td>
</tr>
<tr>
<td>PRCMHibernateWakeUpGPIOSelect</td>
<td>void PRCMHibernateWakeUpGPIOSelect(unsigned long ulGPIOBitMap, unsigned long ulType)</td>
<td>Select the GPIO for the wakeup.</td>
</tr>
<tr>
<td>PRCMHibernateEnter</td>
<td>void PRCMHibernateEnter()</td>
<td>Enter the Hibernate state.</td>
</tr>
<tr>
<td>PRCMSlowClkCtrGet</td>
<td>unsigned long PRCMSlowClkCtrGet()</td>
<td>Get the slow clock counter.</td>
</tr>
<tr>
<td>PRCMSlowClkCtrFastGet</td>
<td>unsigned long PRCMSlowClkCtrFastGet(void)</td>
<td>Get the fast clock counter.</td>
</tr>
<tr>
<td>PRCMSlowClkCtrMatchSet</td>
<td>void PRCMSlowClkCtrMatchSet(unsigned long ullValue)</td>
<td>Match the clock counter.</td>
</tr>
<tr>
<td>PRCMSlowClkCtrMatchGet</td>
<td>unsigned long PRCMSlowClkCtrMatchGet()</td>
<td>Get the matched value of the clock counter.</td>
</tr>
<tr>
<td>PRCMOCRRegisterWrite</td>
<td>void PRCMOCRRegisterWrite(unsigned char uclIndex, unsigned long ulRegValue)</td>
<td>Write to the OCR register.</td>
</tr>
<tr>
<td>PRCMOCRRegisterRead</td>
<td>unsigned long PRCMOCRRegisterRead(unsigned char uclIndex)</td>
<td>Read from the OCR register.</td>
</tr>
<tr>
<td>PRCMIntRegister</td>
<td>void PRCMIntRegister(void(*pfnHandler)(void))</td>
<td>Register interrupt handler.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>void PRCMIntUnregister()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void PRCMIntEnable(unsigned long ulIntFlags)</td>
<td>Enables an interrupt with the specified flags.</td>
<td></td>
</tr>
<tr>
<td>void PRCMIntDisable(unsigned long ulIntFlags)</td>
<td>Disables an interrupt with the specified flags.</td>
<td></td>
</tr>
<tr>
<td>unsigned long PRCMIntStatus()</td>
<td>Returns the status of interrupts.</td>
<td></td>
</tr>
<tr>
<td>void PRCMRTCInUseSet()</td>
<td>Sets the RTC in use flag.</td>
<td></td>
</tr>
<tr>
<td>tBoolean PRCMRTCInUseGet()</td>
<td>Gets the RTC in use flag.</td>
<td></td>
</tr>
<tr>
<td>void PRCMRTCSets(unsigned long ulSecs, unsigned short usMsec)</td>
<td>Sets the RTC to the specified seconds and microseconds.</td>
<td></td>
</tr>
<tr>
<td>void PRCMRTCGets(unsigned long *ulSecs, unsigned short *usMsec)</td>
<td>Gets the RTC to the specified seconds and microseconds.</td>
<td></td>
</tr>
<tr>
<td>void PRCMRTCMatchSets(unsigned long ulSecs, unsigned short usMsec)</td>
<td>Sets the RTC match to the specified seconds and microseconds.</td>
<td></td>
</tr>
<tr>
<td>void PRCMRTCMatchGets(unsigned long *ulSecs, unsigned short *usMsec)</td>
<td>Gets the RTC match to the specified seconds and microseconds.</td>
<td></td>
</tr>
<tr>
<td>void PRCMCC3200MCUInit()</td>
<td>Initializes the CC3200 MCU.</td>
<td></td>
</tr>
<tr>
<td>unsigned long PRCMHIBRegRead(unsigned long ulRegAddr)</td>
<td>Reads the HIB Register with the specified address.</td>
<td></td>
</tr>
<tr>
<td>PRCMHIBRegWrite()</td>
<td>Writes to the HIB Register.</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>void ulRegAddr, unsigned long ulValue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned long PRCMCameraFreqSet</td>
<td>(unsigned char ulDivider, unsigned char ulWidth)</td>
<td></td>
</tr>
</tbody>
</table>
unsigned long PRCM_CameraFreqSet( unsigned char ulDivider, unsigned char ulWidth )

Parameters

ulDivider is clock frequency divider value
ulWidth is the width of the high pulse

This function sets the input frequency for camera module.

The frequency is calculated as follows:

\[
f_{out} = \frac{240MHz}{ulDivider};
\]

The parameter \( ulWidth \) sets the width of the high pulse.

For e.g.:

ulDivider = 4;
ulWidth = 2;

f_out = 30 MHz and 50% duty cycle
And,

\[
\begin{align*}
ulDivider &= 4; \\
ulWidth &= 1; \\
f\_out &= 30 \text{ MHz and } 25\% \text{ duty cycle}
\end{align*}
\]

**Returns**

- 0 on success, 1 on error

---

```c
void PRCMCC3200MCUInit (void )
```

MCU Initialization Routine

This function sets mandatory configurations for the MCU

**Returns**

- None

---

```c
void PRCMHibernateEnter (void )
```

Puts the system into Hibernate

This function puts the system into Hibernate. The device enters HIB immediately and on exit from HIB device core starts its execution from reset thus the function never returns.

**Returns**

- None.
void
PRCMHibernateIntervalSet ( unsigned long long ullTicks )

Sets Hibernate wakeup Timer

Parameters
    ullTicks is number of 32.768 KHz clocks

This function sets internal hibernate wakeup timer running at 32.768 KHz.

Returns
    Returns true on success, false otherwise.

unsigned long
PRCMHibernateWakeupCauseGet ( void )

Get hibernate wakeup cause

This function gets the hibernate wakeup cause.

Returns
    Returns
        PRCM_HIB_WAKEUP_CAUSE_SLOW_CLOCK or
        PRCM_HIB_WAKEUP_CAUSE_GPIO

void
PRCMHibernateWakeUpGPIOSelect ( unsigned long ulGPIOBitMap,
                                 unsigned long ulType )
Selects the GPIO(s) for hibernate wake up.

**Parameters**

- `ulGPIOBitMap` is the bit-map of valid hibernate wakeup GPIO.
- `ulType` is the wakeup trigger type.

This function selects the wakeup GPIO for hibernate and can be used to select any combination of 7 pre-defined GPIO(s).

This function enables individual HIB wakeup source(s). The parameter `ulGPIOBitMap` should be one of the following:

- PRCM_HIB_GPIO2
- PRCM_HIB_GPIO4
- PRCM_HIB_GPIO13
- PRCM_HIB_GPIO17
- PRCM_HIB_GPIO11
- PRCM_HIB_GPIO24
- PRCM_HIB_GPIO26

The parameter `ulType` sets the trigger type and can be one of the following:

- PRCM_HIB_LOW_LEVEL
- PRCM_HIB_HIGH_LEVEL
- PRCM_HIB_FALL_EDGE
- PRCM_HIB_RISE_EDGE

**Returns**

None.

```c
void
PRCMHibernateWakeupSourceDisable ( unsigned long ull )
```

Disable individual HIB wakeup source(s).

**Parameters**
ulHIBWakupSrc is logical OR of valid HIB wakeup source.

This function disable individual HIB wakeup source(s). The parameter ulHIBWakupSrc is same as bit fields defined in PRCMEnableHibernateWakeupSource()

Returns
None.

void
PRCMHibernateWakeupSourceEnable (unsigned long ulHIBWakupSrc)

Enables individual HIB wakeup source(s).

Parameters
ulHIBWakupSrc is logical OR of valid HIB wakeup source.

This function enables individual HIB wakeup source(s). The parameter ulHIBWakupSrc is the bit mask of HIB wakeup sources and should be logical OR of one or more of the following:
- PRCM_HIB_SLOW_CLK_CTR
- PRCM_HIB_GPIO2
- PRCM_HIB_GPIO4
- PRCM_HIB_GPIO13
- PRCM_HIB_GPIO17
- PRCM_HIB_GPIO11
- PRCM_HIB_GPIO24
- PRCM_HIB_GPIO26

Returns
None.

unsigned long
PRCMHIBRegRead (unsigned long ulRegAddr)

Reads 32-bit value from register at specified address.
Parameters
  ulRegAddr is the address of register to be read.

This function reads 32-bit value from the register as specified by ulRegAddr.

Returns
  Return the value of the register.

void PRCMHIBRegWrite ( unsigned long ulRegAddr,
                         unsigned long ulValue )

Writes 32-bit value to register at specified address

Parameters
  ulRegAddr is the address of register to be read.
  ulValue is the 32-bit value to be written.

This function writes 32-bit value passed as ulValue to the register as specified by ulRegAddr

Returns
  None

void PRCMI2SClockFreqSet ( unsigned long ulI2CClkFreq )

Configure I2S fractional divider

Parameters
ull2CClkFreq is the required input clock for McAPS module

This function configures I2S fractional divider. By default this divider is set to output 24 Mhz clock to I2S module.

The minimum frequency that can be obtained by configuring this divider is

\[(240000\text{KHz}/1023.99) = 234.377 \text{ KHz}\]

**Returns**

None.

---

**void PRCMIntDisable ( unsigned long ullIntFlags )**

Disables individual PRCM interrupt sources.

**Parameters**

*ullIntFlags* is the bit mask of the interrupt sources to be disabled.

This function disables the indicated ARCM interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The *ullIntFlags* parameter has the same definition as the *ullIntFlags* parameter to PRCMEnableInterrupt().

**Returns**

None.
void PRCMIntEnable ( unsigned long ullIntFlags )

Enables individual PRCM interrupt sources.

Parameters

    ullIntFlags is the bit mask of the interrupt sources to be enabled.

This function enables the indicated ARCM interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The ullIntFlags parameter is the logical OR of any of the following: -PRCM_INT_SLOW_CLK_CTR

void PRCMIntRegister ( void(*)(void) pfnHandler )

Registers an interrupt handler for the PRCM.

Parameters

    pfnHandler is a pointer to the function to be called when the interrupt is activated.

This function does the actual registering of the interrupt handler. This function enables the global interrupt in the interrupt controller;

Returns

    None.
unsigned long PRCMIntStatus ( void )

 Gets the current interrupt status.

 This function returns the PRCM interrupt status of interrupts that are allowed to reflect to the processor. The interrupts are cleared on read.

 Returns
 Returns the current interrupt status.

 void PRCMIntUnregister ( void )

 Unregisters an interrupt handler for the PRCM.

 This function does the actual unregistering of the interrupt handler. It clears the handler to be called when a PRCM interrupt occurs. This function also masks off the interrupt in the interrupt controller so that the interrupt handler no longer is called.

 Returns
 None.

 void PRCMLPDSEnter ( void )

 Puts the system into Low Power Deel Sleep (LPDS) power mode.

 This function puts the system into Low Power Deel Sleep (LPDS) power mode. A call to this function never returns
and the execution starts from Reset.

See also

PRCMLPDSRestoreInfoSet().

Returns
None.

Note
External debugger will always disconnect whenever the system enters LPDS and debug interface is shutdown until next POR reset. In order to avoid this and allow for connecting back the debugger after waking up from LPDS

See also

PRCMLPDSEnterKeepDebugIf().

void PRCMLPDSEnterKeepDebugIf ( void )

Puts the system into Low Power Deel Sleep (LPDS) power mode keeping debug interface alive.

This function puts the system into Low Power Deel Sleep (LPDS) power mode keeping debug interface alive. A call to this function never returns and the execution starts from Reset

See also

PRCMLPDSRestoreInfoSet().

Returns
None.
**Note**
External debugger will always disconnect whenever the system enters LPDS, using this API will allow connecting back the debugger after waking up from LPDS. This API is recommended for development purposes only as it adds to the current consumption of the system.

```c
void PRCMLPDSIntervalSet ( unsigned long ulTicks )
```

Sets LPDS wakeup Timer

**Parameters**
- `ulTicks` is number of 32.768 KHz clocks

This function sets internal LPDS wakeup timer running at 32.768 KHz. The timer is only configured if the parameter `ulTicks` is in valid range i.e. from 21 to 2^32.

**Returns**
- Returns `true` on success, `false` otherwise.

```c
void PRCMLPDSRestoreInfoSet ( unsigned long ulStackPtr,
                               unsigned long ulProgCntr )
```

Sets the LPDS exit PC and SP restore values.

**Parameters**
- `ulStackPtr` is the SP restore value.
ulProgCntr is the PC restore value

This function sets the LPDS exit PC and SP restore values. Setting \textit{ulProgCntr} to a non-zero value, forces bootloader to jump to that address with Stack Pointer initialized to \textit{ulStackPtr} on LPDS exit, otherwise the application's vector table entries are used.

\textbf{Returns}
None.

\begin{verbatim}
unsigned long PRCMLPDSWakeupCauseGet ( void  )

Get LPDS wakeup cause
This function gets LPDS wakeup cause

\textbf{Returns}
Returns values enumerated as described in \texttt{PRCMLPDSWakeupSourceEnable()}.  
\end{verbatim}

\begin{verbatim}
void
PRCMLPDSWakeUpGPIOSelect ( unsigned long ulGPIOPin
                           unsigned long ulType
                           )

Selects the GPIO for LPDS wakeup

\textbf{Parameters}
\texttt{ulGPIOPin} is one of the valid GPIO for LPDS wakeup.
\texttt{ulType} is the wakeup trigger type.
\end{verbatim}
This function selects the wakeup GPIO for LPDS wakeup and can be used to select one out of 7 pre-defined GPIO(s).

The parameter `ulLpdsGPIOSel` should be one of the following:

- `PRCM_LPDS_GPIO2`
- `PRCM_LPDS_GPIO4`
- `PRCM_LPDS_GPIO13`
- `PRCM_LPDS_GPIO17`
- `PRCM_LPDS_GPIO11`
- `PRCM_LPDS_GPIO24`
- `PRCM_LPDS_GPIO26`

The parameter `ulType` sets the trigger type and can be one of the following:

- `PRCM_LPDS_LOW_LEVEL`
- `PRCM_LPDS_HIGH_LEVEL`
- `PRCM_LPDS_FALL_EDGE`
- `PRCM_LPDS_RISE_EDGE`

Returns
None.

```c
void PRCMLPDSWakeupSourceDisable(unsigned long ulLpdsWakeupSrc)
{
    Disable the individual LPDS wakeup source(s).

    Parameters
    `ulLpdsWakeupSrc` is logical OR of wakeup sources.

    This function enable the individual LPDS wakeup source(s) or three wake up sources (`ulLpdsWakeupSrc`) are supported by
    `PRCM_LPDS_HOST_IRQ` - `PRCM_LPDS_GPIO` - `PRCM_LPI`
```
Returns

None.

void PRCMLPDSWakeupSourceEnable ( unsigned long ulLpdsWakeupSrc )

Enable the individual LPDS wakeup source(s).

Parameters

ulLpdsWakeupSrc is logical OR of wakeup sources.

This function enable the individual LPDS wakeup source(s) ar
three wakeup sources (ulLpdsWakeupSrc ) are supported by:
PRCM_LPDS_HOST_IRQ -PRCM_LPDS_GPIO -PRCM_LPI

Returns

None.

void PRCMMCURest ( tBoolean blIncludeSubsystem )

Performs a software reset of a MCU and associated
peripherals

Parameters

blIncludeSubsystem is true to reset associated
peripherals.

This function performs a software reset of a MCU and
associated peripherals. To reset the associated peripheral,
the parameter blIncludeSubsystem should be set to true.
**Returns**

None.

```c
unsigned long
PRCMOCRRRegisterRead ( unsigned char ucIndex )
```

Read from On-Chip Retention (OCR) register.

This function reads from On-Chip retention register. The device supports two 4-byte OCR register which are retained across all power mode.

The parameter `ucIndex` is an index of the OCR and can be 0 or 1.

**Returns**

None.

```c
void
PRCMOCRRRegisterWrite ( unsigned char ucIndex, unsigned long ulRegValue )
```

Write to On-Chip Retention (OCR) register.

This function writes to On-Chip retention register. The device supports two 4-byte OCR register which are retained across all power mode.

The parameter `ucIndex` is an index of the OCR and can be 0 or 1.
Returns
None.

void
PRCMPeripheralClkDisable ( unsigned long ulPeripheral,
                           unsigned long ulClkFlags )

Disables clock(s) to peripheral.

Parameters
ulPeripheral is one of the valid peripherals
ulClkFlags are bitmask of clock(s) to be enabled.

This function disable the clock for the specified peripheral. Peripherals are by default clock gated (disabled) and generated a bus fault if accessed.

The parameter ulClkFlags can be logical OR bit fields as defined in PRCMEnablePeripheral().

Returns
None.

void
PRCMPeripheralClkEnable ( unsigned long ulPeripheral,
                          unsigned long ulClkFlags )

Enable clock(s) to peripheral.
Parameters

ulPeripheral is one of the valid peripherals

ulClkFlags are bitmask of clock(s) to be enabled.

This function enables the clock for the specified peripheral. Peripherals are by default clock gated (disabled) and generates a bus fault if accessed.

The parameter ulClkFlags can be logical OR of the following:

- PRCM_RUN_MODE_CLK - Ungates clock to the peripheral
- PRCM_SLP_MODE_CLK - Keeps the clocks ungated in sleep.

Returns
None.

unsigned long
PRCMPeripheralClockGet ( unsigned long ulPeripheral )

Gets the input clock for the specified peripheral.

Parameters

ulPeripheral is one of the valid peripherals.

This function gets the input clock for the specified peripheral.

The parameter ulPeripheral has the same definition as that in PRCMPeripheralClkEnable();

Returns

Returns input clock frequency for specified peripheral.
void PRCMPeripheralReset ( unsigned long ulPeripheral )

Performs a software reset of a peripheral.

**Parameters**

- **ulPeripheral** is one of the valid peripheral.

This assert or deassert reset to the specified peripheral based on the `bAssert` parameter.

**Returns**

None.

`tBoolean PRCMPeripheralStatusGet ( unsigned long ulPeripheral )`

Determines if a peripheral is ready.

**Parameters**

- **ulPeripheral** is one of the valid modules

This function determines if a particular peripheral is ready to be accessed. The peripheral may be in a non-ready state if it is not enabled, is being held in reset, or is in the process of becoming ready after being enabled or taken out of reset.

**Returns**

Returns **true** if the peripheral is ready, **false** otherwise.
void PRCMRTCGet (unsigned long * ulSecs, 
               unsigned short * usMsec)

Get the instantaneous calendar time from the device.

Parameters

ulSecs  refers to the seconds part of the calendar time
usMsec refers to the fractional (ms) part of the second

This function fetches the instantaneous value of the ticking
calendar time from the device. The calendar time is
outlined in terms of seconds and milliseconds.

The device provides the calendar value that has been
maintained across active and low power states.

The function PRCMRTCSet() must have been invoked
once to set a reference.

Returns

None.

tBoolean PRCMRTCInUseGet (void)

Ascertain whether function of RTC is being used

This function indicates whether function of RTC is being
used on the device or not.

This routine should be utilized by the application software,
when returning from low-power, to confirm that RTC has
been put to use and may not need to set the value of the RTC.

The RTC feature, if set or marked, can be only reset either through reboot or power cycle.

**Returns**

None.

```c
void PRCMRTCInUseSet ( void )
```

Mark the function of RTC as being used

This function marks in HW that feature to maintain calendar time in device is being used.

Specifically, this feature reserves user's HIB Register-1 accessed through PRCMOCRRegisterWrite(1) for internal work / purpose, therefore, the stated register is not available to user. Also, users must not excercise the Slow Clock Counter API(s), if RTC has been set for use.

The RTC feature, if set or marked, can be only reset either through reboot or power cycle.

**Returns**

None.

```c
void PRCMRTCMatchGet ( unsigned long * ulSecs,
                        unsigned short * usMsec
                      )
```
Get a previously set calendar time alarm.

**Parameters**
- **ulSecs** refers to the seconds part of the calendar time
- **usMsec** refers to the fractional (ms) part of the second

This function fetches from the device a wall-clock alarm that would have been previously set in the device. The calendar time is outlined in terms of seconds and milliseconds.

If no alarm was set in the past, then this function would fetch a random information.

The function **PRCMRTCMatchSet()** must have been invoked once to set an alarm.

**Returns**
None.

```c
void PRCMRTCMatchSet ( unsigned long ulSecs,
                        unsigned short  usMsec
                      )
```

Set a calendar time alarm.

**Parameters**
- **ulSecs** refers to the seconds part of the calendar time
- **usMsec** refers to the fractional (ms) part of the second

This function sets a wall-clock alarm in the device to be reported for a futuristic calendar time. The calendar time is outlined in terms of seconds and milliseconds.
The device provides uses the calendar value that has been maintained across active and low power states to report attainment of alarm time.

The function **PRCMRTCSet()** must have been invoked once to set a reference.

Returns

None.

```c
void PRCMRTCSet ( unsigned long ulSecs, 
                 unsigned short usMsec 
)
```

Set the calendar time in the device.

**Parameters**

- **ulSecs** refers to the seconds part of the calendar time
- **usMsec** refers to the fractional (ms) part of the second

This function sets the specified calendar time in the device. The calendar time is outlined in terms of seconds and milliseconds. However, the device makes no assumption about the origin or reference of the calendar time.

The device uses the indicated calendar value to update and maintain the wall-clock time across active and low power states.

The function **PRCMRTCInUseSet()** must be invoked prior to use of this feature.
Returns
None.

`void PRCMSleepEnter ( void )`

Puts the system into Sleep.

This function puts the system into sleep power mode. System exits the power state on any one of the available interrupt. On exit from sleep mode the function returns to the calling function with all the processor core registers retained.

Returns
None.

`unsigned long long PRCMSlowClkCtrFastGet ( void )`

Gets the current value of the internal slow clock counter

This function is similar to

See also
`PRCMSlowClkCtrGet()` but reads the counter value from a relatively faster interface using an auto-latch mechanism.

Note
Due to the nature of implementation of auto latching, when using this API, the recommendation is to read the value thrice and identify the right value (as 2 out the 3 read values will always be correct and with a max. of 1
Returns
64-bit current counter value.

unsigned long long PRCMSlowClkCtrGet ( void )

Gets the current value of the internal slow clock counter

This function latches and reads the internal RTC running at 32.768 Khz

Returns
64-bit current counter value.

unsigned long long PRCMSlowClkCtrMatchGet ( void )

Gets slow clock counter match value.

This function gets the match value for slow clock counter. This is use to interrupt the processor when RTC counts to the specified value.

Returns
None.

void
PRCMSlowClkCtrMatchSet ( unsigned long long ullValue )

Sets slow clock counter match value to interrupt the
processor.

**Parameters**

*ullValue* is the match value.

This function sets the match value for slow clock counter. This is use to interrupt the processor when RTC counts to the specified value.

**Returns**

None.

```c
void PRCMSRAMRetentionDisable (unsigned long ulSramColSel, unsigned long ulFlags)
```

Disable SRAM column retention during LPDS Power mode(s).

**Parameters**

*ulSramColSel* is bit mask of valid SRAM columns.
*ulFlags* is the bit mask of power modes.

This functions disable the SRAM retention. The device support configurable SRAM column retention in Low Power Deep Sleep (LPDS). Each column is of 64 KB size.

The parameter *ulSramColSel* should be logical OR of the following:

- PRCM_SRAM_COL_1
- PRCM_SRAM_COL_2
- PRCM_SRAM_COL_3
- PRCM_SRAM_COL_4

The parameter *ulFlags* selects the power modes and sholud b
logical OR of one or more of the following - PRCM_SRAM_LPDS_RET

Returns
None.

```c
void
PRCMSRAMRetentionEnable ( unsigned long ulSramColSel
                           unsigned long ulModeFlags
                      )
```

Enable SRAM column retention during LPDS Power mode(s)

**Parameters**
- `ulSramColSel` is bit mask of valid SRAM columns.
- `ulModeFlags` is the bit mask of power modes.

This function enables the SRAM retention. The device supports configurable SRAM column retention in Low Power Deep Sleep (LPDS). Each column is of 64 KB size.

The parameter `ulSramColSel` should be logical OR of the following:- PRCM_SRAM_COL_1 - PRCM_SRAM_COL_2 - PRCM_SRAM_COL_3 - PRCM_SRAM_COL_4

The parameter `ulModeFlags` selects the power modes and should be logical OR of one or more of the following - PRCM_SRAM_LPDS_RET

**Returns**
None.
unsigned long PRCMSysResetCauseGet ( void )

Gets the reason for a reset.

This function returns the reason(s) for a reset. The reset reason are:
- **PRCM_POWER_ON** - Device is powering up.
- **PRCM_LPDS_EXIT** - Device is exiting from LPDS.
- **PRCM_CORE_RESET** - Device is exiting soft core only reset.
- **PRCM_MCU_RESET** - Device is exiting soft subsystem reset.
- **PRCM_WDT_RESET** - Device was reset by watchdog.
- **PRCM_SOC_RESET** - Device is exiting SOC reset.
- **PRCM_HIB_EXIT** - Device is exiting hibernate.

Returns
Returns one of the cause defined above.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Secure_Digital_Host_api</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void SDHostInit (unsigned long ulBase)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostCmdReset (unsigned long ulBase)</code></td>
<td></td>
</tr>
<tr>
<td><code>long SDHostCmdSend (unsigned long ulBase, unsigned long ulCmd, unsigned ulArg)</code></td>
<td></td>
</tr>
<tr>
<td><code>tBoolean SDHostDataNonBlockingWrite (unsigned long ulBase, unsigned long ulData)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostDataWrite (unsigned long ulBase, unsigned long ulData)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostDataRead (unsigned long ulBase, unsigned long *pulData)</code></td>
<td></td>
</tr>
<tr>
<td><code>tBoolean SDHostDataNonBlockingRead (unsigned long ulBase, unsigned long *pulData)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostIntRegister (unsigned long ulBase, void(*pfnHandler)(void))</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostIntUnregister (unsigned long ulBase)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostIntEnable (unsigned long ulBase, unsigned long ulIntFlags)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostIntDisable (unsigned long ulBase, unsigned long ulIntFlags)</code></td>
<td></td>
</tr>
<tr>
<td><code>SDHostIntDisable (unsigned long ulBase, unsigned long ulIntFlags)</code></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><code>void unsigned long ulIntFlags)</code></td>
<td></td>
</tr>
<tr>
<td><code>unsigned long SDHostIntStatus (unsigned long ulBase)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostIntClear (unsigned long ulBase, unsigned long ulIntFlags)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostCardErrorMaskSet (unsigned long ulBase, unsigned long ulErrMask)</code></td>
<td></td>
</tr>
<tr>
<td><code>unsigned long SDHostCardErrorMaskGet (unsigned long ulBase)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostSetExpClk (unsigned long ulBase, unsigned long ulSDHostClk, unsigned long ulCardClk)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostRespGet (unsigned long ulBase, unsigned long ulResponse[4])</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostBlockSizeSet (unsigned long ulBase, unsigned short ulBlkSize)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SDHostBlockCountSet (unsigned long ulBase, unsigned short ulBlkCount)</code></td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

```c
void SDHostBlockCountSet ( unsigned long ulBase,
                          unsigned short ulBlkCount )
```

Set the block size and count for data transfer

**Parameters**

- `ulBase` is the base address of SDHost module
- `ulBlkCount` is the number of blocks

This function sets block count for the data transfer. This needs to be set for each block transfer.

**See also**

- `SDHostBlockSizeSet()`

**Returns**

None.

```c
void SDHostBlockSizeSet ( unsigned long ulBase,
                          unsigned short ulBlkSize )
```
Set the block size for data transfer

**Parameters**

- **ulBase** is the base address of SDHost module
- **ulBlkSize** is the transfer block size in bytes

This function sets the block size the data transfer.

The parameter *ulBlkSize* is size of each data block in bytes. This should be in range 0 - 2^10.

**Returns**

None.

```
unsigned long
SDHostCardErrorMaskGet (unsigned long ulBase)
```

Gets the card status error mask.

**Parameters**

- **ulBase** is the base address of SDHost module

This function gets the card status error mask for response type R1, R1b, R5, R5b and R6 response.

**Returns**

Returns the current card status error.

```
void
SDHostCardErrorMaskSet (unsigned long ulBase,
                        unsigned long ulErrMask)
```

Sets the card status error mask.

**Parameters**

- `ulBase` is the base address of SDHost module
- `ulErrMask` is the bit mask of card status errors to be enabled

This function sets the card status error mask for response type R1, R1b, R5, R5b and R6 response. The parameter `ulErrMask` is the bit mask of card status errors to be enabled, if the corresponding bits in the 'card status' field of a response are set then the host controller indicates a card error interrupt status. Only bits referenced as type E (error) in status field in the response can set a card status error.

**Returns**

None

```c
void SDHostCmdReset ( unsigned long ulBase )
```

Resets SDHost command line

**Parameters**

- `ulBase` is the base address of SDHost module.

This function asserts a soft reset for the command line

**Returns**

None.
long SDHostCmdSend (unsigned long ulBase, 
unsigned long ulCmd, 
unsigned ulArg)

Sends command over SDHost interface

**Parameters**

- **ulBase** is the base address of SDHost module.
- **ulCmd** is the command to send.
- **ulArg** is the argument for the command.

This function sends command to the attached card over the SDHost interface.

The **ulCmd** parameter can be one of **SDHOST_CMD_0** to **SDHOST_CMD_63**. It can be logically ORed with one or more of the following:

- **SDHOST_MULTI_BLK** for multi-block transfer
- **SDHOST_WR_CMD** if command is followed by write data
- **SDHOST_RD_CMD** if command is followed by read data
- **SDHOST_DMA_EN** if SDHost need to generate DMA request.
- **SDHOST_RESP_LEN_136** if 136 bit response is expected
- **SDHOST_RESP_LEN_48** if 48 bit response is expected
- **SDHOST_RESP_LEN_48B** if 48 bit response with busy bit is expected
The parameter *ulArg* is the argument for the command

**Returns**

Returns 0 on success, -1 otherwise.

```c
// Function to read from the SDHost read buffer

#define SDHOST_DATA_NONBLOCKING_READ

unsigned long SDHostDataNonBlockingRead(unsigned long ulBase, unsigned long *pulData)

This function reads a data word from the SDHost read buffer. The function returns **true** if there was data available in the buffer, else returns **false**.

**Returns**

Return **true** on success, **false** otherwise.

```c
// Function to write to the SDHost write buffer

#define SDHOST_DATA_NONBLOCKING_WRITE

unsigned long SDHostDataNonBlockingWrite(unsigned long ulBase, unsigned long ulData)

This function writes a data word into the SDHost write buffer.

**Parameters**
**ulBase** is the base address of SDHost module. **ulData** is data word to be transfered.

This function writes a single data word into the SDHost write buffer. The function returns **true** if there was a space available in the buffer else returns **false**.

**Returns**
Return **true** on success, **false** otherwise.

```c
void SDHostDataRead ( unsigned long   ulBase,
                       unsigned long * pulData
) {
    ...  // Code for SDHostDataRead
}
```

Waits for a data word from the SDHost read buffer

**Parameters**
- **ulBase** is the base address of SDHost module.
- **pulData** is pointer to read data variable.

This function reads a single data word from the SDHost read buffer. If there is no data available in the buffer the function will wait until a data word is received before returning.

**Returns**
None.

```c
void SDHostDataWrite ( unsigned long   ulBase,
                       unsigned long   ulData
) {
    ...  // Code for SDHostDataWrite
}
```
Waits to write a data word into the SDHost write buffer.

Parameters

ulBase is the base address of SDHost module.
ulData is data word to be transfered.

This function writes ulData into the SDHost write buffer. If there is no space in the write buffer this function waits until there is a space available before returning.

Returns

None.

void SDHostInit ( unsigned long ulBase )

Configures SDHost module.

Parameters

ulBase is the base address of SDHost module.

This function configures the SDHost module, enabling internal sub-modules.

Returns

None.

void SDHostIntClear ( unsigned long ulBase, unsigned long ullIntFlags )
Clears the individual interrupt sources.

**Parameters**

- `ulBase` is the base address of SDHost module.
- `ullIntFlags` is a bit mask of the interrupt sources to be cleared.

The specified SDHost interrupt sources are cleared, so that they no longer assert. This function must be called in the interrupt handler to keep the interrupt from being recognized again immediately upon exit.

The `ullIntFlags` parameter has the same definition as the `ullIntFlags` parameter to `SDHostIntEnable()`.

**Returns**

None.

```c
void SDHostIntDisable ( unsigned long ulBase, 
                        unsigned long ullIntFlags )
```

Enable individual interrupt source for the specified SDHost

**Parameters**

- `ulBase` is the base address of SDHost module.
- `ullIntFlags` is a bit mask of the interrupt sources to be enabled.

This function disables the indicated SDHost interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have
no effect on the processor.

The `ulIntFlags` parameter has the same definition as the `ulIntFlags` parameter to `SDHostIntEnable()`.

**Returns**

None.

```c
void SDHostIntEnable ( unsigned long ulBase,
          unsigned long ulIntFlags )
```

Enable individual interrupt source for the specified SDHost

**Parameters**

- `ulBase` is the base address of SDHost module.
- `ulIntFlags` is a bit mask of the interrupt sources to be enabled.

This function enables the indicated SDHost interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The `ulIntFlags` parameter is the logical OR of any of the following:

- `SDHOST_INT_CC` Command Complete interrupt
- `SDHOST_INT_TC` Transfer Complete interrupt
- `SDHOST_INT_BWR` Buffer Write Ready interrupt
- `SDHOST_INT_BRR` Buffer Read Ready interrupt
- `SDHOST_INT_ERRI` Error interrupt
- **SDHOST_INT_CTO** Command Timeout error interrupt
- **SDHOST_INT_CEB** Command End Bit error interrupt
- **SDHOST_INT_DTO** Data Timeout error interrupt
- **SDHOST_INT_DCRC** Data CRC error interrupt
- **SDHOST_INT_DEB** Data End Bit error
- **SDHOST_INT_CERR** Cart Status Error interrupt
- **SDHOST_INT_BADA** Bad Data error interrupt
- **SDHOST_INT_DMARD** Read DMA done interrupt
- **SDHOST_INT_DMAWR** Write DMA done interrupt

Note that **SDHOST_INT_ERRI** can only be used with

See also

*SDHostIntStatus()* and is internally logical OR of all error status bits. Setting this bit alone as *ulIntFlags* doesn't generates any interrupt.

Returns

None.

```c
void SDHostIntRegister (unsigned long ulBase, 
    void(*)(void) pfnHandler
)
```

Registers the interrupt handler for SDHost interrupt

**Parameters**

- **ulBase** is the base address of SDHost module
- **pfnHandler** is a pointer to the function to be called when the SDHost interrupt occurs.

This function does the actual registering of the interrupt
This function enables the global interrupt in the interrupt controller; specific SDHost interrupts must be enabled via `SDHostIntEnable()`. It is the interrupt handler's responsibility to clear the interrupt source.

See also

`IntRegister()` for important information about registering interrupt handlers.

Returns

None.

```c
unsigned long SDHostIntStatus ( unsigned long ulBase )
```

Gets the current interrupt status.

Parameters

`ulBase` is the base address of SDHost module.

This function returns the interrupt status for the specified SDHost.

Returns

Returns the current interrupt status, enumerated as a bit field of values described in `SDHostIntEnable()`.

```c
void SDHostIntUnregister ( unsigned long ulBase )
```

Unregisters the interrupt handler for SDHost interrupt
Parameters

`ulBase` is the base address of SDHost module

This function does the actual unregistering of the interrupt handler. It clears the handler to be called when a SDHost interrupt occurs. This function also masks off the interrupt in the interrupt controller so that the interrupt handler no longer is called.

See also

`IntRegister()` for important information about registering interrupt handlers.

Returns

None.

```c
void SDHostRespGet ( unsigned long ulBase,
                    unsigned long ulResponse[4]
               )
```

Get the response for the last command.

Parameters

`ulBase` is the base address of SDHost module

`ulResponse` is 128-bit response.

This function gets the response from the SD card for the last command send.

Returns

None.
void SDHostSetExpClk ( unsigned long ulBase,
                        unsigned long ulSDHostClk,
                        unsigned long ulCardClk
                    )

Sets the SD Card clock.

**Parameters**

- **ulBase** is the base address of SDHost module
- **ulSDHostClk** is the rate of clock supplied to SDHost module
- **ulCardClk** is the required SD interface clock

This function configures the SDHost interface to supply the specified clock to the connected card.

**Returns**

None.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
</table>

**SHA_Secure_Hash_Algorithm_api**
### Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>SHAMD5DMAEnable</td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5DMADisable</td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td>uint32_t</td>
<td>SHAMD5IntStatus</td>
<td>(uint32_t ui32Base, bool bMasked)</td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5IntEnable</td>
<td>(uint32_t ui32Base, uint32_t ui32IntFlags)</td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5IntDisable</td>
<td>(uint32_t ui32Base, uint32_t ui32IntFlags)</td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5IntClear</td>
<td>(uint32_t ui32Base, uint32_t ui32IntFlags)</td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5IntRegister</td>
<td>(uint32_t ui32Base, void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5IntUnregister</td>
<td>(uint32_t ui32Base)</td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5DataLengthSet</td>
<td>(uint32_t ui32Base, uint32_t ui32Length)</td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5ConfigSet</td>
<td>(uint32_t ui32Base, uint32_t ui32Mode)</td>
</tr>
<tr>
<td>bool</td>
<td>SHAMD5DataWriteNonBlocking</td>
<td>(uint32_t ui32Base, uint8_t *pui8Src)</td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5DataWrite (uint32_t ui32Base, uint8_t *pui8Src)</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5ResultRead (uint32_t ui32Base, uint8_t *pui8Dest)</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>SHAMD5DataProcess (uint32_t ui32Base, uint8_t *pui8DataSrc, uint32_t ui32DataLength, uint8_t *pui8HashResult)</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>SHAMD5HMACProcess (uint32_t ui32Base, uint8_t *pui8DataSrc, uint32_t ui32DataLength, uint8_t *pui8HashResult)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5HMACPPKeyGenerate (uint32_t ui32Base, uint8_t *pui8Key, uint8_t *pui8PPKey)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5HMACKeySet (uint32_t ui32Base, uint8_t *pui8Src)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>SHAMD5HMACPPKeySet (uint32_t ui32Base, uint8_t *pui8Src)</td>
<td></td>
</tr>
</tbody>
</table>
### Function Documentation

```c
void SHAMD5ConfigSet ( uint32_t ui32Base,
                         uint32_t ui32Mode
                     )
```

Writes the mode in the SHA/MD5 module.

**Parameters**

- **ui32Base** is the base address of the SHA/MD5 module.
- **ui32Mode** is the mode of the SHA/MD5 module.

This function writes the mode register configuring the SHA/MD5 module.

The `ui32Mode` parameter is a bit-wise OR of values:

- **SHAMD5_ALGO_MD5** - Regular hash with MD5
- **SHAMD5_ALGO_SHA1** - Regular hash with SHA-1
- **SHAMD5_ALGO_SHA224** - Regular hash with SHA-224
- **SHAMD5_ALGO_SHA256** - Regular hash with SHA-256
- **SHAMD5_ALGO_HMAC_MD5** - HMAC with MD5
- **SHAMD5_ALGO_HMAC_SHA1** - HMAC with SHA-1
- **SHAMD5_ALGO_HMAC_SHA224** - HMAC with SHA-224
- **SHAMD5_ALGO_HMAC_SHA256** - HMAC with SHA-256

**Returns**
None

```c
void SHAMD5DataLengthSet ( uint32_t ui32Base,
                            uint32_t ui32Length
                      )
```

Write the hash length to the SHA/MD5 module.

**Parameters**

- `ui32Base` is the base address of the SHA/MD5 module.
- `ui32Length` is the hash length in bytes.

This function writes the length of the hash data of the current operation to the SHA/MD5 module. The value must be a multiple of 64 if the close hash is not set in the mode register.

**Note**

When this register is written, hash processing is triggered.

**Returns**

None.
bool SHAMD5DataProcess ( uint32_t ui32Base, 
    uint8_t * pui8DataSrc, 
    uint32_t ui32DataLength, 
    uint8_t * pui8HashResult )

Compute a hash using the SHA/MD5 module.

Parameters
    **ui32Base** is the base address of the SHA/MD5 module.
    **pui8DataSrc** is a pointer to an array of data that contains the data that will be hashed.
    **ui32DataLength** specifies the length of the data to be hashed in bytes.
    **pui8HashResult** is a pointer to an array that holds the result of the hashing operation.

This function computes the hash of an array of data using the SHA/MD5 module.

The length of the hash result is dependent on the algorithm that is in use. The following table shows the correct array size for each algorithm:
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of Words in Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>4 Words (128 bits)</td>
</tr>
<tr>
<td>SHA-1</td>
<td>5 Words (160 bits)</td>
</tr>
<tr>
<td>SHA-224</td>
<td>7 Words (224 bits)</td>
</tr>
<tr>
<td>SHA-256</td>
<td>8 Words (256 bits)</td>
</tr>
</tbody>
</table>

Returns
None

void SHAMD5DataWrite ( uint32_t ui32Base, uint8_t * pui8Src )

Perform a blocking write of 64 bytes of data to the SHA/MD5 module.

Parameters
ui32Base is the base address of the SHA/MD5 module.
pui8Src is the pointer to the 64-byte array of data that will be written.

This function does not return until the module is ready to accept data and the data has been written.

Returns
None.

bool SHAMD5DataWriteNonBlocking ( uint32_t ui32Base, uint8_t * pui8Src )
Perform a non-blocking write of 16 words of data to the SHA/MD5 module.

**Parameters**

- `ui32Base` is the base address of the SHA/MD5 module.
- `pui8Src` is the pointer to the 16-word array of data that will be written.

This function writes 16 words of data into the data register.

**Returns**

This function returns true if the write completed successfully. It returns false if the module was not ready.

```c
void SHAMD5DMADisable ( uint32_t ui32Base )
```

Disables the uDMA requests in the SHA/MD5 module.

**Parameters**

- `ui32Base` is the base address of the SHA/MD5 module.

This function configures the DMA options of the SHA/MD5 module.

**Returns**

None
void SHAMD5DMAEnable ( uint32_t ui32Base )

Enables the uDMA requests in the SHA/MD5 module.

Parameters
ui32Base is the base address of the SHA/MD5 module.

This function configures the DMA options of the SHA/MD5 module.

Returns
None

void SHAMD5HMACKeySet ( uint32_t ui32Base, 
                       uint8_t * pui8Src )

Writes an HMAC key to the digest registers in the SHA/MD5 module.

Parameters
ui32Base is the base address of the SHA/MD5 module.
pui8Src is the pointer to the 16-word array of the HMAC key.

This function is used to write HMAC key to the digest registers for key preprocessing. The size of pui8Src must be 512 bytes. If the key is less than 512 bytes, then it must be padded with zeros.
Note
It is recommended to use the SHAMD5GetIntStatus function to check whether the context is ready before writing the key.

Returns
None

```c
void SHAMD5HMACPPKeyGenerate ( uint32_t ui32Base,
                             uint8_t * pui8Key,
                             uint8_t * pui8PPKey )
```

Process an HMAC key using the SHA/MD5 module.

Parameters
- **ui32Base** is the base address of the SHA/MD5 module.
- **pui8Key** is a pointer to an array that contains the key to be processed.
- **pui8PPKey** is the pointer to the array that contains the pre-processed key.

This function processes an HMAC key using the SHA/MD5. The resultant pre-processed key can then be used with later HMAC operations to speed processing time.

The **pui8Key** array must be 512 bits long. If the key is less than 512 bits, it must be padded with zeros. The **pui8PPKey** array must each be 512 bits long.
void SHAMD5HMACPPKeySet (uint32_t ui32Base, uint8_t * pui8Src)

Writes a pre-processed HMAC key to the digest registers in the SHA/MD5 module.

Parameters
- **ui32Base** is the base address of the SHA/MD5 module.
- **pui8Src** is the pointer to the 16-word array of the HMAC key.

This function is used to write HMAC key to the digest registers for key preprocessing. The size of pui8Src must be 512 bytes. If the key is less than 512 bytes, then it must be padded with zeros.

**Note**
It is recommended to use the SHAMD5GetIntStatus function to check whether the context is ready before writing the key.

Returns
None

bool SHAMD5HMACProcess (uint32_t ui32Base, uint8_t * pui8DataSrc,
Compute a HMAC with key pre-processing using the SHA/MD5 module.

**Parameters**

- **ui32Base** is the base address of the SHA/MD5 module.
- **pui8DataSrc** is a pointer to an array of data that contains the data that is to be hashed.
- **ui32DataLength** specifies the length of the data to be hashed in bytes.
- **pui8HashResult** is a pointer to an array that holds the result of the hashing operation.

This function computes a HMAC with the given data using the SHA/MD5 module with a preprocessed key.

The length of the hash result is dependent on the algorithm that is selected with the **ui32Algo** argument. The following table shows the correct array size for each algorithm:
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of Words in Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>4 Words (128 bits)</td>
</tr>
<tr>
<td>SHA-1</td>
<td>5 Words (160 bits)</td>
</tr>
<tr>
<td>SHA-224</td>
<td>7 Words (224 bits)</td>
</tr>
<tr>
<td>SHA-256</td>
<td>8 Words (256 bits)</td>
</tr>
</tbody>
</table>

Returns
None

void SHAMD5IntClear ( uint32_t ui32Base,
                       uint32_t ui32IntFlags )

Clears interrupt sources in the SHA/MD5 module.

Parameters

ui32Base is the base address of the SHA/MD5 module.

ui32IntFlags contains desired interrupts to disable.

ui32IntFlags must be a logical OR of one or more of the following values:

- **SHAMD5_INT_CONTEXT_READY** - Context input registers are ready.
- **SHAMD5_INT_PARTHASH_READY** - Context output registers are ready after a context switch.
- **SHAMD5_INT_INPUT_READY** - Data FIFO is ready to receive data.
- **SHAMD5_INT_OUTPUT_READY** - Context output registers are ready.

Returns
None.
void SHAMD5IntDisable ( uint32_t ui32Base,
                   uint32_t ui32IntFlags )

Disable interrupt sources in the SHA/MD5 module.

Parameters

ui32Base is the base address of the SHA/MD5 module.

ui32IntFlags contains desired interrupts to disable.

ui32IntFlags must be a logical OR of one or more of the following values:

- **SHAMD5_INT_CONTEXT_READY** - Context input registers are ready.
- **SHAMD5_INT_PARTHASH_READY** - Context output registers are ready after a context switch.
- **SHAMD5_INT_INPUT_READY** - Data FIFO is ready to receive data.
- **SHAMD5_INT_OUTPUT_READY** - Context output registers are ready.

Returns

None.

void SHAMD5IntEnable ( uint32_t ui32Base,
                   uint32_t ui32IntFlags )
Enable interrupt sources in the SHA/MD5 module.

Parameters

- **ui32Base** is the base address of the SHA/MD5 module.
- **ui32IntFlags** contains desired interrupts to enable.

This function enables interrupt sources in the SHA/MD5 module. **ui32IntFlags** must be a logical OR of one or more of the following values:

- **SHAMD5_INT_CONTEXT_READY** - Context input registers are ready.
- **SHAMD5_INT_PARTHASH_READY** - Context output registers are ready after a context switch.
- **SHAMD5_INT_INPUT_READY** - Data FIFO is ready to receive data.
- **SHAMD5_INT_OUTPUT_READY** - Context output registers are ready.

Returns

None.

```c
void SHAMD5IntRegister ( uint32_t ui32Base, uint32_t ui32IntFlags, void(*)(void) pfnHandler )
```

Registers an interrupt handler for the SHA/MD5 module.

Parameters

- **ui32Base** is the base address of the SHA/MD5 module.
pfnHandler is a pointer to the function to be called when the enabled SHA/MD5 interrupts occur.

This function registers the interrupt handler in the interrupt vector table, and enables SHA/MD5 interrupts on the interrupt controller; specific SHA/MD5 interrupt sources must be enabled using SHAMD5IntEnable(). The interrupt handler being registered must clear the source of the interrupt using SHAMD5IntClear().

If the application is using a static interrupt vector table stored in flash, then it is not necessary to register the interrupt handler this way. Instead, IntEnable() should be used to enable SHA/MD5 interrupts on the interrupt controller.

See also
IntRegister() for important information about registering interrupt handlers.

Returns
None.

```
uint32_t SHAMD5IntStatus ( uint32_t ui32Base,
                  bool bMasked )
```

Get the interrupt status of the SHA/MD5 module.

Parameters
ui32Base is the base address of the SHA/MD5 module.

bMasked is **false** if the raw interrupt status is required and **true** if the masked interrupt status is required.

This function returns the current value of the IRQSTATUS register. The value will be a logical OR of the following:

- **SHAMD5_INT_CONTEXT_READY** - Context input registers are ready.
- **SHAMD5_INT_PARTHASH_READY** - Context output registers are ready after a context switch.
- **SHAMD5_INT_INPUT_READY** - Data FIFO is ready to receive data.
- **SHAMD5_INT_OUTPUT_READY** - Context output registers are ready.

**Returns**

Interrupt status

```c
void SHAMD5IntUnregister ( uint32_t ui32Base )
```

Unregisters an interrupt handler for the SHA/MD5 module.

**Parameters**

ui32Base is the base address of the SHA/MD5 module.

This function unregisters the previously registered interrupt handler and disables the interrupt in the interrupt controller.
See also

IntRegister() for important information about registering interrupt handlers.

Returns

None.

```c
void SHAMD5ResultRead ( uint32_t ui32Base,
                        uint8_t * pui8Dest )
```

Reads the result of a hashing operation.

**Parameters**

- `ui32Base` is the base address of the SHA/MD5 module.
- `pui8Dest` is the pointer to the byte array of data that will be written.

This function does not return until the module is ready to accept data and
the data has been written.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of Words in Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>16 Bytes (128 bits)</td>
</tr>
<tr>
<td>SHA-1</td>
<td>20 Bytes (160 bits)</td>
</tr>
<tr>
<td>SHA-224</td>
<td>28 Bytes (224 bits)</td>
</tr>
<tr>
<td>SHA-256</td>
<td>32 Bytes (256 bits)</td>
</tr>
</tbody>
</table>

Returns
None.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
</table>

**SPI_Serial_Peripheral_Interface_api**
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void SPIEnable (unsigned long ulBase)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SPIDisable (unsigned long ulBase)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SPIDmaEnable (unsigned long ulBase, unsigned long ulFlags)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SPIDmaDisable (unsigned long ulBase, unsigned long ulFlags)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SPIReset (unsigned long ulBase)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SPIConfigSetExpClk (unsigned long ulBase, unsigned long ulSPIClk, unsigned long ulBitRate, unsigned long ulMode, unsigned long ulSubMode, unsigned long ulConfig)</code></td>
<td></td>
</tr>
<tr>
<td><code>long SPIDataGetNonBlocking (unsigned long ulBase, unsigned long *pulData)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SPIDataGet (unsigned long ulBase, unsigned long *pulData)</code></td>
<td></td>
</tr>
<tr>
<td><code>long SPIDataPutNonBlocking (unsigned long ulBase, unsigned long ulData)</code></td>
<td></td>
</tr>
<tr>
<td><code>void SPIDataPut (unsigned long ulBase, unsigned long ulData)</code></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>void SPIFIFOEnable</td>
<td>(unsigned long ulBase, unsigned long ulFlags)</td>
</tr>
<tr>
<td>void SPIFIFODisable</td>
<td>(unsigned long ulBase, unsigned long ulFlags)</td>
</tr>
<tr>
<td>void SPIFIFOLevelSet</td>
<td>(unsigned long ulBase, unsigned long ulTxLevel, unsigned long ulRxLevel)</td>
</tr>
<tr>
<td>void SPIFIFOLevelGet</td>
<td>(unsigned long ulBase, unsigned long *pulTxLevel, unsigned long *pulRxLevel)</td>
</tr>
<tr>
<td>void SPIWordCountSet</td>
<td>(unsigned long ulBase, unsigned long ulWordCount)</td>
</tr>
<tr>
<td>void SPIIntRegister</td>
<td>(unsigned long ulBase, void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>void SPIIntUnregister</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void SPIIntEnable</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
</tr>
<tr>
<td>void SPIIntDisable</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
</tr>
<tr>
<td>unsigned long SPIIntStatus</td>
<td>(unsigned long ulBase, tBoolean bMasked)</td>
</tr>
<tr>
<td>void SPIIntClear</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
</tr>
</tbody>
</table>
long ullIntFlags)

void **SPICSEnable** (unsigned long ulBase)

void **SPICSDisable** (unsigned long ulBase)

long **SPITransfer** (unsigned long ulBase, unsigned char *ucDout, unsigned char *ucDin, unsigned long ulCount, unsigned long ulFlags)
void SPIConfigSetExpClk ( unsigned long ulBase,
unigned long ulSPIClk,
unsigned long ulBitRate,
unsigned long ulMode,
unsigned long ulSubMode,
unsigned long ulConfig )

Sets the configuration of a SPI module

Parameters
ulBase is the base address of the SPI module
ulSPIClk is the rate of clock supplied to the SPI module.
ulBitRate is the desired bit rate.(master mode)
ulMode is the mode of operation.
ulSubMode is one of the valid sub-modes.
ulConfig is logical OR of configuration parameters.

This function configures SPI port for operation in specified sub-mode and required bit rated as specified by ulMode and ulBitRate parameters respectively.
The SPI module can operate in either master or slave mode. The parameter *ulMode* can be one of the following - **SPI_MODE_MASTER** - **SPI_MODE_SLAVE**

The SPI module supports 4 sub modes based on SPI clock polarity and phase.

<table>
<thead>
<tr>
<th>Polarity</th>
<th>Phase</th>
<th>Sub-Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Required sub mode can be select by setting *ulSubMode* parameter to one of the following:

- **SPI_SUB_MODE_0**
- **SPI_SUB_MODE_1**
- **SPI_SUB_MODE_2**
- **SPI_SUB_MODE_3**

The parameter *ulConfig* is logical OR of five values: the word length, active level for chip select, software or hardware controled chip select, 3 or 4 pin mode and turbo mode.

SPI support 8, 16 and 32 bit word lengths defined by:-

- **SPI_WL_8**
- **SPI_WL_16**
- **SPI_WL_32**

Active state of Chip[ Selece can be defined by:-

- **SPI_CS_ACTIVELOW**
• **SPI_CS_ACTIVEHIGH**

SPI chip select can be configured to be controlled either by hardware or software:-

• **SPI_SW_CS**
• **SPI_HW_CS**

The module can work in 3 or 4 pin mode defined by:-

• **SPI_3PIN_MODE**
• **SPI_4PIN_MODE**

Turbo mode can be set on or turned off using:-

• **SPI_TURBO_MODE_ON**
• **SPI_TURBO_MODE_OFF**

### Returns
None.

```c
void SPICSDisable ( unsigned long ulBase )
```

Disables the chip select in software controlled mode

**Parameters**
ulBase is the base address of the SPI module.

This function disables the Chip select in software controlled mode. The active state of CS will depend on the configuration done via sa **SPIConfigSetExpClk()**.

**Returns**
void SPICSEnable( unsigned long ulBase )

Enables the chip select in software controlled mode

Parameters

ulBase is the base address of the SPI module.

This function enables the Chip select in software controlled mode. The active state of CS will depend on the configuration done via

See also

SPIConfigExpClkSet().

Returns

None.

void SPIDataGet( unsigned long ulBase, unsigned long * pulData )

Waits for the word to be received on the specified port.

Parameters

ulBase is the base address of the SPI module.

pulData is pointer to receive data variable.

This function gets a SPI word from the receive FIFO for the specified port. If there is no word available, this function
waits until a word is received before returning.

**Returns**

Returns the word read from the specified port, cast as an *unsigned* long.

```c
long SPIDataGetNonBlocking ( unsigned long ulBase, unsigned long * pulData )
```

Receives a word from the specified port.

**Parameters**

- `ulBase` is the base address of the SPI module.
- `pulData` is pointer to receive data variable.

This function gets a SPI word from the receive FIFO for the specified port.

**Returns**

Returns the number of elements read from the receive FIFO.

```c
void SPIDataPut ( unsigned long ulBase, unsigned long ulData )
```

Waits until the word is transmitted on the specified port.

**Parameters**
ulBase is the base address of the SPI module
ulData is data to be transmitted.

This function transmits a SPI word on the transmit FIFO for the specified port. This function waits until the space is available on transmit FIFO

Returns
None

long SPIDataPutNonBlocking ( unsigned long ulBase, unsigned long ulData )

Transmits a word on the specified port.

Parameters
ulBase is the base address of the SPI module
ulData is data to be transmitted.

This function transmits a SPI word on the transmit FIFO for the specified port.

Returns
Returns the number of elements written to the transmit FIFO.

void SPIDisable ( unsigned long ulBase )

Disables the transmitting and receiving.
Parameters

ulBase is the base address of the SPI module

This function disables the SPI channel for transmitting and receiving.

Returns

None

void SPIDmaDisable ( unsigned long ulBase,
                      unsigned long ulFlags )

Disables the SPI DMA operation for transmitting and/or receiving.

Parameters

ulBase is the base address of the SPI module
ulFlags selectes the DMA signal for transmit and/or receive.

This function disables transmit and/or receive DMA request based on the ulFlags parameter.

The parameter ulFlags is the logical OR of one or more of the following :

- SPI_RX_DMA
- SPI_TX_DMA

Returns

None.
void SPI_dmaEnable ( unsigned long ulBase,  
                 unsigned long ulFlags )  

Enables the SPI DMA operation for transmitting and/or receiving.

Parameters

ulBase is the base address of the SPI module
ulFlags selects the DMA signal for transmit and/or receive.

This function enables transmit and/or receive DMA request based on the ulFlags parameter.

The parameter ulFlags is the logical OR of one or more of the following:

- SPI_RX_DMA
- SPI_TX_DMA

Returns

None.

void SPIEnable ( unsigned long ulBase )

Enables transmitting and receiving.

Parameters

ulBase is the base address of the SPI module
This function enables the SPI channel for transmitting and receiving.

**Returns**
- None

```c
void SPIFIFODisable ( unsigned long ulBase,
                      unsigned long ulFlags )
```

Disables the transmit and/or receive FIFOs.

**Parameters**
- `ulBase` is the base address of the SPI module
- `ulFlags` selects the FIFO(s) to be enabled

This function disables transmit and/or receive FIFOs. as specified by `ulFlags`. The parameter `ulFlags` should be logical OR of one or more of the following:

- `SPI_TX_FIFO`
- `SPI_RX_FIFO`

**Returns**
- None.

```c
void SPIFIFOEnable ( unsigned long ulBase,
                     unsigned long ulFlags )
```

Enables the transmit and/or receive FIFOs.
Parameters

ulBase is the base address of the SPI module
ulFlags selects the FIFO(s) to be enabled

This function enables the transmit and/or receive FIFOs as specified by ulFlags. The parameter ulFlags should be logical OR of one or more of the following:

- SPI_TX_FIFO
- SPI_RX_FIFO

Returns

None.

void SPIFIFOLevelGet (unsigned long ulBase, unsigned long * pulTxLevel, unsigned long * pulRxLevel)

Gets the FIFO level at which DMA requests or interrupts are generated.

Parameters

ulBase is the base address of the SPI module
pulTxLevel is a pointer to storage for the transmit FIFO level
pulRxLevel is a pointer to storage for the receive FIFO level

This function gets the FIFO level at which DMA requests or interrupts are generated.
Returns
None.

```c
void SPIFIFOLevelSet ( unsigned long ulBase,
                        unsigned long ulTxLevel,
                        unsigned long ulRxLevel )
```

Sets the FIFO level at which DMA requests or interrupts are generated.

Parameters
- `ulBase` is the base address of the SPI module
- `ulTxLevel` is the Almost Empty Level for transmit FIFO.
- `ulRxLevel` is the Almost Full Level for the receive FIFO.

This function Sets the FIFO level at which DMA requests or interrupts are generated.

Returns
None.

```c
void SPIIntClear ( unsigned long ulBase,
                   unsigned long ulIntFlags )
```

Clears SPI interrupt sources.
Parameters

ulBase  is the base address of the SPI module
ullIntFlags is a bit mask of the interrupt sources to be cleared.

The specified SPI interrupt sources are cleared, so that they no longer assert. This function must be called in the interrupt handler to keep the interrupt from being recognized again immediately upon exit.

The ullIntFlags parameter has the same definition as the ullIntFlags parameter to SPIIntEnable().

Returns

None.

void SPIIntDisable ( unsigned long  ulBase,
                    unsigned long  ullIntFlags
)

Disables individual SPI interrupt sources.

Parameters

ulBase  is the base address of the SPI module
ullIntFlags is the bit mask of the interrupt sources to be disabled.

This function 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.
The `ulIntFlags` parameter has the same definition as the `ulIntFlags` parameter to `SPIIntEnable()`.

Returns
None.

```c
void SPIIntEnable ( unsigned long ulBase,
                    unsigned long ulIntFlags )
```

Enables individual SPI interrupt sources.

Parameters

- `ulBase` is the base address of the SPI module
- `ulIntFlags` is the bit mask of the interrupt sources to be enabled.

This function 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.

The `ulIntFlags` parameter is the logical OR of any of the following:

- `SPI_INT_DMATX`
- `SPI_INT_DMARX`
- `SPI_INT_EOW`
- `SPI_INT_RX_OVRFLOW`
- `SPI_INT_RX_FULL`
- `SPI_INT_TX_UDRFLOW`
- `SPI_INT_TX_EMPTY`
void SPIIntRegister ( unsigned long ulBase,  
    void(*)(void) pfnHandler  
  )

Registers an interrupt handler for a SPI interrupt.

Parameters
  ulBase is the base address of the SPI module  
  pfnHandler is a pointer to the function to be called  
    when the SPI interrupt occurs.

This function does the actual registering of the interrupt handler. This function enables the global interrupt in the  
interrupt controller; specific SPI interrupts must be enabled  
via SPIIntEnable(). It is the interrupt handler's responsibility  
to clear the interrupt source.

See also
  IntRegister() for important information about  
      registering interrupt handlers.

Returns
  None.

unsigned long SPIIntStatus ( unsigned long ulBase,  
    tBoolean bMasked  
  )
Gets the current interrupt status.

**Parameters**

ulBase is the base address of the SPI module  

bMasked is **false** if the raw interrupt status is required and **true** if the masked interrupt status is required.

This function returns the interrupt status for the specified SPI. The status of interrupts that are allowed to reflect to the processor can be returned.

**Returns**

Returns the current interrupt status, enumerated as a bit field of values described in `SPIIntEnable()`.

```c
void SPIIntUnregister ( unsigned long ulBase )
```

Unregisters an interrupt handler for a SPI interrupt.

**Parameters**

ulBase is the base address of the SPI module

This function does the actual unregistering of the interrupt handler. It clears the handler to be called when a SPI interrupt occurs. This function also masks off the interrupt in the interrupt controller so that the interrupt handler no longer is called.

**See also**

`IntRegister()` for important information about registering interrupt handlers.
Returns
None.

```c
void SPIReset ( unsigned long ulBase )
```

Performs a software reset of the specified SPI module

**Parameters**

- **ulBase** is the base address of the SPI module

This function performs a software reset of the specified SPI module

Returns
None.

```c
long SPITransfer ( unsigned long ulBase, unsigned char * ucDout, unsigned char * ucDin, unsigned long ulCount, unsigned long ulFlags )
```

Send/Receive data buffer over SPI channel

**Parameters**

- **ulBase** is the base address of SPI module
- **ucDout** is the pointer to Tx data buffer or 0.
- **ucDin** is pointer to Rx data buffer or 0
- **ulCount** is the size of data in bytes.
ulFlags controls chip select toggling.

This function transfers ulCount bytes of data over SPI channel. Since the API sends a SPI word at a time ulCount should be a multiple of word length set using SPIConfigSetExpClk().

If the ucDout parameter is set to 0, the function will send 0xFF over the SPI MOSI line.

If the ucDin parameter is set to 0, the function will ignore data on SPI MISO line.

The parameter ulFlags is logical OR of one or more of the following

- SPI_CS_ENABLE if CS needs to be enabled at start of transfer.
- SPI_CS_DISABLE if CS need to be disabled at the end of transfer.

This function will not return until data has been transmitted.

**Returns**

Returns 0 on success, -1 otherwise.

```c
void SPIWordCountSet ( unsigned long ulBase,
                        unsigned long ulWordCount )
```

Sets the word count.

**Parameters**
**ulBase** is the base address of the SPI module.

**ulWordCount** is number of SPI words to be transmitted.

This function sets the word count, which is the number of SPI word to be transferred on channel when using the FIFO buffer.

**Returns**

None.
<table>
<thead>
<tr>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systick_api</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>void</th>
<th>SysTickEnable (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>SysTickDisable (void)</td>
</tr>
<tr>
<td>void</td>
<td>SysTickIntRegister (void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>void</td>
<td>SysTickIntUnregister (void)</td>
</tr>
<tr>
<td>void</td>
<td>SysTickIntEnable (void)</td>
</tr>
<tr>
<td>void</td>
<td>SysTickIntDisable (void)</td>
</tr>
<tr>
<td>void</td>
<td>SysTickPeriodSet (unsigned long ulPeriod)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>SysTickPeriodGet (void)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>SysTickValueGet (void)</td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

`void SysTickDisable ( void )`

Disables the SysTick counter.

This function stops the SysTick counter. If an interrupt handler has been registered, it is not called until SysTick is restarted.

**Returns**

None.

`void SysTickEnable ( void )`

Enables the SysTick counter.

This function starts the SysTick counter. If an interrupt handler has been registered, it is called when the SysTick counter rolls over.

**Note**

Calling this function causes the SysTick counter to (re)commence counting from its current value. The counter is not automatically reloaded with the period as specified in a previous call to `SysTickPeriodSet()`. If an
immediate reload is required, the `NVIC_ST_CURRENT` register must be written to force the reload. Any write to this register clears the SysTick counter to 0 and causes a reload with the supplied period on the next clock.

**Returns**
None.

```c
void SysTickIntDisable ( void )
```

Disables the SysTick interrupt.

This function disables the SysTick interrupt, preventing it from being reflected to the processor.

**Returns**
None.

```c
void SysTickIntEnable ( void )
```

Enables the SysTick interrupt.

This function enables the SysTick interrupt, allowing it to be reflected to the processor.

**Note**
The SysTick interrupt handler is not required to clear the SysTick interrupt source because it is cleared automatically by the NVIC when the interrupt handler is called.

**Returns**
void SysTickIntRegister ( void(*)(void) pfnHandler )

Registers an interrupt handler for the SysTick interrupt.

Parameters

- **pfnHandler** is a pointer to the function to be called when the SysTick interrupt occurs.

This function registers the handler to be called when a SysTick interrupt occurs.

See also

- **IntRegister()** for important information about registering interrupt handlers.

Returns

None.

void SysTickIntUnregister ( void )

Unregisters the interrupt handler for the SysTick interrupt.

This function unregisters the handler to be called when a SysTick interrupt occurs.

See also

- **IntRegister()** for important information about registering interrupt handlers.

Returns

None.
**unsigned long SysTickPeriodGet ( void )**

Gets the period of the SysTick counter.

This function returns the rate at which the SysTick counter wraps, which equates to the number of processor clocks between interrupts.

**Returns**

Returns the period of the SysTick counter.

**void SysTickPeriodSet ( unsigned long ulPeriod )**

Sets the period of the SysTick counter.

**Parameters**

- `ulPeriod` is the number of clock ticks in each period of the SysTick counter and must be between 1 and 16,777,216, inclusive.

This function sets the rate at which the SysTick counter wraps, which equates to the number of processor clocks between interrupts.

**Note**

Calling this function does not cause the SysTick counter to reload immediately. If an immediate reload is required, the `NVIC_ST_CURRENT` register must be written. Any write to this register clears the SysTick
counter to 0 and causes a reload with the \textit{ulPeriod} supplied here on the next clock after SysTick is enabled.

\textbf{Returns}
None.

\begin{verbatim}
unsigned long SysTickValueGet ( void )
\end{verbatim}

Gets the current value of the SysTick counter.

This function returns the current value of the SysTick counter, which is a value between the period - 1 and zero, inclusive.

\textbf{Returns}
Returns the current value of the SysTick counter.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
</table>

**GPT_General_Purpose_Timer_api**
### Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>TimerEnable</td>
<td>(unsigned long ulBase, unsigned long ulTimer)</td>
</tr>
<tr>
<td>void</td>
<td>TimerDisable</td>
<td>(unsigned long ulBase, unsigned long ulTimer)</td>
</tr>
<tr>
<td>void</td>
<td>TimerConfigure</td>
<td>(unsigned long ulBase, unsigned long ulConfig)</td>
</tr>
<tr>
<td>void</td>
<td>TimerControlLevel</td>
<td>(unsigned long ulBase, unsigned long ulTimer, tBoolean bInvert)</td>
</tr>
<tr>
<td>void</td>
<td>TimerControlEvent</td>
<td>(unsigned long ulBase, unsigned long ulTimer, unsigned long ulEvent)</td>
</tr>
<tr>
<td>void</td>
<td>TimerControlStall</td>
<td>(unsigned long ulBase, unsigned long ulTimer, tBoolean bStall)</td>
</tr>
<tr>
<td>void</td>
<td>TimerPrescaleSet</td>
<td>(unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>TimerPrescaleGet</td>
<td>(unsigned long ulBase, unsigned long ulTimer)</td>
</tr>
<tr>
<td>void</td>
<td>TimerPrescaleMatchSet</td>
<td>(unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)</td>
</tr>
<tr>
<td>Data Type</td>
<td>Function Name</td>
<td>Parameters</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>unsigned long</td>
<td>TimerPrescaleMatchGet</td>
<td>(unsigned long ulBase, unsigned long ulTimer)</td>
</tr>
<tr>
<td>void</td>
<td>TimerLoadSet</td>
<td>(unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>TimerLoadGet</td>
<td>(unsigned long ulBase, unsigned long ulTimer)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>TimerValueGet</td>
<td>(unsigned long ulBase, unsigned long ulTimer)</td>
</tr>
<tr>
<td>void</td>
<td>TimerValueSet</td>
<td>(unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)</td>
</tr>
<tr>
<td>void</td>
<td>TimerMatchSet</td>
<td>(unsigned long ulBase, unsigned long ulTimer, unsigned long ulValue)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>TimerMatchGet</td>
<td>(unsigned long ulBase, unsigned long ulTimer)</td>
</tr>
<tr>
<td>void</td>
<td>TimerIntRegister</td>
<td>(unsigned long ulBase, unsigned long ulTimer, void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>void</td>
<td>TimerIntUnregister</td>
<td>(unsigned long ulBase, unsigned long ulTimer)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>void TimerIntEnable</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
<td></td>
</tr>
<tr>
<td>void TimerIntDisable</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
<td></td>
</tr>
<tr>
<td>unsigned long TimerIntStatus</td>
<td>(unsigned long ulBase, tBoolean bMasked)</td>
<td></td>
</tr>
<tr>
<td>void TimerIntClear</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
<td></td>
</tr>
<tr>
<td>void TimerDMAEventSet</td>
<td>(unsigned long ulBase, unsigned long ulDMAEvent)</td>
<td></td>
</tr>
<tr>
<td>unsigned long TimerDMAEventGet</td>
<td>(unsigned long ulBase)</td>
<td></td>
</tr>
</tbody>
</table>
Function Documentation

```c
void TimerConfigure ( unsigned long ulBase,
                      unsigned long ulConfig )
```

Configures the timer(s).

**Parameters**

- `ulBase` is the base address of the timer module.
- `ulConfig` is the configuration for the timer.

This function configures the operating mode of the timer(s). The timer module is disabled before being configured, and is left in the disabled state. The 16/32-bit timer is comprised of two 16-bit timers that can operate independently or be concatenated to form a 32-bit timer.

The configuration is specified in `ulConfig` as one of the following values:

- `TIMER_CFG_ONE_SHOT` - Full-width one-shot timer
- `TIMER_CFG_ONE_SHOT_UP` - Full-width one-shot timer that counts up instead of down (not available on all parts)
- `TIMER_CFG_PERIODIC` - Full-width periodic timer
• **TIMER_CFG_PERIODIC_UP** - Full-width periodic timer that counts up instead of down (not available on all parts)
• **TIMER_CFG_SPLIT_PAIR** - Two half-width timers

When configured for a pair of half-width timers, each timer is separately configured. The first timer is configured by setting `ulConfig` to the result of a logical OR operation between one of the following values and `ulConfig`:

• **TIMER_CFG_A_ONE_SHOT** - Half-width one-shot timer
• **TIMER_CFG_A_ONE_SHOT_UP** - Half-width one-shot timer that counts up instead of down (not available on all parts)
• **TIMER_CFG_A_PERIODIC** - Half-width periodic timer
• **TIMER_CFG_A_PERIODIC_UP** - Half-width periodic timer that counts up instead of down (not available on all parts)
• **TIMER_CFG_A_CAP_COUNT** - Half-width edge count capture
• **TIMER_CFG_A_CAP_TIME** - Half-width edge time capture
• **TIMER_CFG_A_PWM** - Half-width PWM output

Similarly, the second timer is configured by setting `ulConfig` to the result of a logical OR operation between one of the corresponding **TIMER_CFG_B_*** values and `ulConfig`.

**Returns**
None.

```c
void TimerControlEvent ( unsigned long ulBase, 
```
Controls the event type.

**Parameters**

- **ulBase** is the base address of the timer module.
- **ulTimer** specifies the timer(s) to be adjusted; must be one of `TIMER_A`, `TIMER_B`, or `TIMER_BOTH`.
- **ulEvent** specifies the type of event; must be one of `TIMER_EVENT_POS_EDGE`, `TIMER_EVENT_NEG_EDGE`, or `TIMER_EVENT_BOTH_EDGES`.

This function sets the signal edge(s) that triggers the timer when in capture mode.

**Returns**

None.

```c
void TimerControlLevel ( unsigned long ulBase,
                        unsigned long ulTimer,
                        tBoolean bInvert )
```

Controls the output level.

**Parameters**

- **ulBase** is the base address of the timer module.
**ulTimer** specifies the timer(s) to adjust; must be one of **TIMER_A**, **TIMER_B**, or **TIMER_BOTH**.

**bInvert** specifies the output level.

This function sets the PWM output level for the specified timer. If the **bInvert** parameter is **true**, then the timer's output is made active low; otherwise, it is made active high.

**Returns**

None.

```c
void TimerControlStall ( unsigned long ulBase,
                         unsigned long ulTimer,
                         tBoolean bStall )
```

Controls the stall handling.

**Parameters**

- **ulBase** is the base address of the timer module.
- **ulTimer** specifies the timer(s) to be adjusted; must be one of **TIMER_A**, **TIMER_B**, or **TIMER_BOTH**.
- **bStall** specifies the response to a stall signal.

This function controls the stall response for the specified timer. If the **bStall** parameter is **true**, then the timer stops counting if the processor enters debug mode; otherwise the timer keeps running while in debug mode.

**Returns**
void TimerDisable ( unsigned long ulBase,
                unsigned long ulTimer )

Disables the timer(s).

Parameters

ulBase is the base address of the timer module.
ulTimer specifies the timer(s) to disable; must be one of TIMER_A, TIMER_B, or TIMER_BOTH.

This function disables operation of the timer module.

Returns

None.

unsigned long TimerDMAEventGet ( unsigned long ulBase )

Returns the events that can trigger a DMA request.

Parameters

ulBase is the base address of the timer module.

This function returns the timer events that can trigger the start of a DMA sequence. The DMA trigger events are the logical OR of the following values:

- TIMER_DMA_MODEMATCH_B - Enables the mode
match DMA trigger for timer B.
- **TIMER_DMA_CAPEVENT_B** - Enables the capture event DMA trigger for timer B.
- **TIMER_DMA_CAPMATCH_B** - Enables the capture match DMA trigger for timer B.
- **TIMER_DMA_TIMEOUT_B** - Enables the timeout DMA trigger for timer B.
- **TIMER_DMA_MODEMATCH_A** - Enables the mode match DMA trigger for timer A.
- **TIMER_DMA_CAPEVENT_A** - Enables the capture event DMA trigger for timer A.
- **TIMER_DMA_CAPMATCH_A** - Enables the capture match DMA trigger for timer A.
- **TIMER_DMA_TIMEOUT_A** - Enables the timeout DMA trigger for timer A.

**Returns**

The timer events that trigger the uDMA.

```c
void TimerDMAEventSet ( unsigned long ulBase,
                        unsigned long ulDMAEvent
                    )
```

Enables the events that can trigger a DMA request.

**Parameters**

- **ulBase** is the base address of the timer module.
- **ulDMAEvent** is a bit mask of the events that can trigger DMA.

This function enables the timer events that can trigger the start of a DMA sequence. The DMA trigger events are
specified in the *ui32DMAEvent* parameter by passing in the logical OR of the following values:

- **TIMER_DMA_MODEMATCH_B** - The mode match DMA trigger for timer B is enabled.
- **TIMER_DMA_CAPEVENT_B** - The capture event DMA trigger for timer B is enabled.
- **TIMER_DMA_CAPMATCH_B** - The capture match DMA trigger for timer B is enabled.
- **TIMER_DMA_TIMEOUT_B** - The timeout DMA trigger for timer B is enabled.
- **TIMER_DMA_MODEMATCH_A** - The mode match DMA trigger for timer A is enabled.
- **TIMER_DMA_CAPEVENT_A** - The capture event DMA trigger for timer A is enabled.
- **TIMER_DMA_CAPMATCH_A** - The capture match DMA trigger for timer A is enabled.
- **TIMER_DMA_TIMEOUT_A** - The timeout DMA trigger for timer A is enabled.

**Returns**

None.

```c
void TimerEnable ( unsigned long ulBase,
                  unsigned long ulTimer )
```

Enables the timer(s).

**Parameters**

- **ulBase** is the base address of the timer module.
- **ulTimer** specifies the timer(s) to enable; must be one
of TIMER_A, TIMER_B, or TIMER_BOTH.

This function enables operation of the timer module. The timer must be configured before it is enabled.

Returns
None.

```c
void TimerIntClear ( unsigned long ulBase,
                    unsigned long ulIntFlags )
```

Clears timer interrupt sources.

Parameters
- **ulBase** is the base address of the timer module.
- **ulIntFlags** is a bit mask of the interrupt sources to be cleared.

The specified timer interrupt sources are cleared, so that they no longer assert. This function must be called in the interrupt handler to keep the interrupt from being triggered again immediately upon exit.

The **ulIntFlags** parameter has the same definition as the **ulIntFlags** parameter to `TimerIntEnable()`.

**Note**
Because there is a write buffer in the Cortex-M3 processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early
in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

**Returns**
None.

```c
void TimerIntDisable ( unsigned long ulBase,
                       unsigned long ulIntFlags )
```

Disables individual timer interrupt sources.

**Parameters**
- `ulBase` is the base address of the timer module.
- `ulIntFlags` is the bit mask of the interrupt sources to be disabled.

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

The `ulIntFlags` parameter has the same definition as the `ulIntFlags` parameter to `TimerIntEnable()`.

**Returns**
None.
void TimerIntEnable ( unsigned long ulBase,  
               unsigned long ullIntFlags  
           )

Enables individual timer interrupt sources.

Parameters
  ulBase is the base address of the timer module.
  ullIntFlags is the bit mask of the interrupt sources to be enabled.

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

The ullIntFlags parameter must be the logical OR of any combination of the following:

- **TIMER_CAPB_EVENT** - Capture B event interrupt
- **TIMER_CAPB_MATCH** - Capture B match interrupt
- **TIMER_TIMB_TIMEOUT** - Timer B timeout interrupt
- **TIMER_CAPA_EVENT** - Capture A event interrupt
- **TIMER_CAPA_MATCH** - Capture A match interrupt
- **TIMER_TIMA_TIMEOUT** - Timer A timeout interrupt

Returns
  None.

void TimerIntRegister ( unsigned long ulBase,  
                       unsigned long ulTimer,  
                       void(*)(void) pfnHandler  
                   )
Registers an interrupt handler for the timer interrupt.

Parameters

- `ulBase` is the base address of the timer module.
- `ulTimer` specifies the timer(s); must be one of `TIMER_A`, `TIMER_B`, or `TIMER_BOTH`.
- `PFNHandler` is a pointer to the function to be called when the timer interrupt occurs.

This function sets the handler to be called when a timer interrupt occurs. In addition, this function enables the global interrupt in the interrupt controller; specific timer interrupts must be enabled via `TimerIntEnable()`. It is the interrupt handler's responsibility to clear the interrupt source via `TimerIntClear()`.

See also

- `IntRegister()` for important information about registering interrupt handlers.

Returns

None.

```c
unsigned long TimerIntStatus( unsigned long ulBase,
                              tBoolean bMasked )
```

Gets the current interrupt status.
Parameters

`ulBase` is the base address of the timer module.
`bMasked` is false if the raw interrupt status is required and true if the masked interrupt status is required.

This function returns the interrupt status for the timer module. Either the raw interrupt status or the status of interrupts that are allowed to reflect to the processor can be returned.

Returns

The current interrupt status, enumerated as a bit field of values described in `TimerIntEnable()`.

```
void TimerIntUnregister ( unsigned long ulBase, 
                        unsigned long ulTimer 
                    )
```

Unregisters an interrupt handler for the timer interrupt.

Parameters

`ulBase` is the base address of the timer module.
`ulTimer` specifies the timer(s); must be one of `TIMER_A`, `TIMER_B`, or `TIMER_BOTH`.

This function clears the handler to be called when a timer interrupt occurs. This function also masks off the interrupt in the interrupt controller so that the interrupt handler no longer is called.

See also
IntRegister() for important information about registering interrupt handlers.

Returns
None.

unsigned long TimerLoadGet ( unsigned long ulBase, unsigned long ulTimer )

Gets the timer load value.

Parameters
ulBase is the base address of the timer module.
ulTimer specifies the timer; must be one of TIMER_A or TIMER_B. Only TIMER_A should be used when the timer is configured for full-width operation.

This function gets the currently programmed interval load value for the specified timer.

Note
This function can be used for both full- and half-width modes of 16/32-bit timers.

Returns
Returns the load value for the timer.

void TimerLoadSet ( unsigned long ulBase, unsigned long ulTimer,
unsigned long ulValue
)

Sets the timer load value.

Parameters

ulBase is the base address of the timer module.
ulTimer specifies the timer(s) to adjust; must be one of TIMER_A, TIMER_B, or TIMER_BOTH. Only TIMER_A should be used when the timer is configured for full-width operation.
ulValue is the load value.

This function sets the timer load value; if the timer is running then the value is immediately loaded into the timer.

Note
This function can be used for both full- and half-width modes of 16/32-bit timers.

Returns
None.

unsigned long TimerMatchGet ( unsigned long ulBase,
               unsigned long ulTimer
            )

Gets the timer match value.

Parameters

ulBase is the base address of the timer module.
**ulTimer** specifies the timer; must be one of **TIMER_A** or **TIMER_B**. Only **TIMER_A** should be used when the timer is configured for 32-bit operation.

This function gets the match value for the specified timer.

**Returns**

Returns the match value for the timer.

```c
void TimerMatchSet ( unsigned long ulBase,
                     unsigned long ulTimer,
                     unsigned long ulValue
                  )
```

Sets the timer match value.

**Parameters**

- **ulBase** is the base address of the timer module.
- **ulTimer** specifies the timer(s) to adjust; must be one of **TIMER_A**, **TIMER_B**, or **TIMER_BOTH**. Only **TIMER_A** should be used when the timer is configured for 32-bit operation.
- **ulValue** is the match value.

This function sets the match value for a timer. This is used in capture count mode to determine when to interrupt the processor and in PWM mode to determine the duty cycle of the output signal.

**Returns**

None.
unsigned long TimerPrescaleGet (unsigned long ulBase, unsigned long ulTimer)

Get the timer prescale value.

**Parameters**

- **ulBase** is the base address of the timer module.
- **ulTimer** specifies the timer; must be one of `TIMER_A` or `TIMER_B`.

This function gets the value of the input clock prescaler. The prescaler is only operational when in half-width mode and is used to extend the range of the half-width timer modes.

**Returns**

The value of the timer prescaler.

unsigned long TimerPrescaleMatchGet (unsigned long ulBase, unsigned long ulTimer)

Get the timer prescale match value.

**Parameters**

- **ulBase** is the base address of the timer module.
**ulTimer** specifies the timer; must be one of **TIMER_A** or **TIMER_B**.

This function gets the value of the input clock prescaler match value. When in a half-width mode that uses the counter match and prescaler, the prescale match effectively extends the range of the match.

**Note**

The availability of the prescaler match varies with the part and timer mode in use. Please consult the datasheet for the part you are using to determine whether this support is available.

**Returns**

The value of the timer prescale match.

```c
void TimerPrescaleMatchSet ( unsigned long ulBase, 
                           unsigned long ulTimer, 
                           unsigned long ulValue )
```

Set the timer prescale match value.

**Parameters**

- **ulBase** is the base address of the timer module.
- **ulTimer** specifies the timer(s) to adjust; must be one of **TIMER_A**, **TIMER_B**, or **TIMER_BOTH**.
- **ulValue** is the timer prescale match value which must be between 0 and 255 (inclusive) for 16/32-bit timers.
This function sets the value of the input clock prescaler match value. When in a half-width mode that uses the counter match and the prescaler, the prescale match effectively extends the range of the match.

**Note**
The availability of the prescaler match varies with the part and timer mode in use. Please consult the datasheet for the part you are using to determine whether this support is available.

**Returns**
None.

```c
void TimerPrescaleSet ( unsigned long ulBase,
                        unsigned long ulTimer,
                        unsigned long ulValue )
```

Set the timer prescale value.

**Parameters**
- **ulBase** is the base address of the timer module.
- **ulTimer** specifies the timer(s) to adjust; must be one of `TIMER_A`, `TIMER_B`, or `TIMER_BOTH`.
- **ulValue** is the timer prescale value which must be between 0 and 255 (inclusive) for 16/32-bit timers.

This function sets the value of the input clock prescaler. The prescaler is only operational when in half-width mode.
and is used to extend the range of the half-width timer modes.

**Returns**
None.

```c
unsigned long TimerValueGet ( unsigned long ulBase, 
                              unsigned long ulTimer )
```

Gets the current timer value.

**Parameters**
- `ulBase` is the base address of the timer module.
- `ulTimer` specifies the timer; must be one of `TIMER_A` or `TIMER_B`. Only `TIMER_A` should be used when the timer is configured for 32-bit operation.

This function reads the current value of the specified timer.

**Returns**
Returns the current value of the timer.

```c
void TimerValueSet ( unsigned long ulBase, 
                     unsigned long ulTimer, 
                     unsigned long ulValue )
```

Sets the current timer value.
Parameters

ulBase is the base address of the timer module.

ulTimer specifies the timer; must be one of TIMER_A or TIMER_B. Only TIMER_A should be used when the timer is configured for 32-bit operation.

ulValue is the new value of the timer to be set.

This function sets the current value of the specified timer.

Returns

None.
# UART_api

<table>
<thead>
<tr>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_api</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void UARTParityModeSet</code></td>
<td>(unsigned long ulBase, unsigned long ulParity)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>unsigned long UARTParityModeGet</code></td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void UARTFIFOLevelSet</code></td>
<td>(unsigned long ulBase, unsigned long ulTxLevel, unsigned long ulRxLevel)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void UARTFIFOLevelGet</code></td>
<td>(unsigned long ulBase, unsigned long *pulTxLevel, unsigned long *pulRxLevel)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void UARTConfigSetExpClk</code></td>
<td>(unsigned long ulBase, unsigned long ulUARTClk, unsigned long ulBaud, unsigned long ulConfig)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void UARTConfigGetExpClk</code></td>
<td>(unsigned long ulBase, unsigned long ulUARTClk, unsigned long *pulBaud, unsigned long *pulConfig)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void UARTEnable</code></td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void UARTDisable</code></td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>void UARTFIFOEnable</code></td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void UARTFIFODisable</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void UARTModemControlSet</td>
<td>(unsigned long ulBase, unsigned long ulControl)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void UARTModemControlClear</td>
<td>(unsigned long ulBase, unsigned long ulControl)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned long UARTModemControlGet</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned long UARTModemStatusGet</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void UARTFlowControlSet</td>
<td>(unsigned long ulBase, unsigned long ulMode)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned long UARTFlowControlGet</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void UARTTxIntModeSet</td>
<td>(unsigned long ulBase, unsigned long ulMode)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned long UARTTxIntModeGet</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tBoolean UARTCharsAvail</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tBoolean UARTSpaceAvail</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long UARTCharGetNonBlocking</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Function Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------</td>
<td>------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td>UARTCharGet</td>
<td>(unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td>tBoolean</td>
<td>UARTCharPutNonBlocking</td>
<td>(unsigned long ulBase, unsigned char ucData)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>UARTCharPut</td>
<td>(unsigned long ulBase, unsigned char ucData)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>UARTBreakCtl</td>
<td>(unsigned long ulBase, tBoolean bBreakState)</td>
<td></td>
</tr>
<tr>
<td>tBoolean</td>
<td>UARTBusy</td>
<td>(unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>UARTIntRegister</td>
<td>(unsigned long ulBase, void(*pfnHandler)(void))</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>UARTIntUnregister</td>
<td>(unsigned long ulBase)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>UARTIntEnable</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>UARTIntDisable</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
<td></td>
</tr>
<tr>
<td>unsigned long</td>
<td>UARTIntStatus</td>
<td>(unsigned long ulBase, tBoolean bMasked)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>UARTIntClear</td>
<td>(unsigned long ulBase, unsigned long ullIntFlags)</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td>UARTDMAEnable</td>
<td>(unsigned long ulBase, unsigned long ulDMAFlags)</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void UARTDMADisable</td>
<td>(unsigned long ulBase, unsigned long ulDMAFlags)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned long UARTRxErrorGet</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void UARTRxErrorClear</td>
<td>(unsigned long ulBase)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
void UARTBreakCtl (unsigned long ulBase, tBoolean bBreakState)

Causes a BREAK to be sent.

Parameters
  ulBase is the base address of the UART port.
  bBreakState controls the output level.

Calling this function with bBreakState set to true asserts a break condition on the UART. Calling this function with bBreakState set to false removes the break condition. For proper transmission of a break command, the break must be asserted for at least two complete frames.

Returns
  None.

tBoolean UARTBusy (unsigned long ulBase)

Determines whether the UART transmitter is busy or not.
Parameters
ulBase is the base address of the UART port.

Allows the caller to determine whether all transmitted bytes have cleared the transmitter hardware. If false is returned, the transmit FIFO is empty and all bits of the last transmitted character, including all stop bits, have left the hardware shift register.

Returns
Returns true if the UART is transmitting or false if all transmissions are complete.

long UARTCharGet (unsigned long ulBase)

Waits for a character from the specified port.

Parameters
ulBase is the base address of the UART port.

This function gets a character from the receive FIFO for the specified port. If there are no characters available, this function waits until a character is received before returning.

Returns
Returns the character read from the specified port, cast as a long.

long UARTCharGetNonBlocking (unsigned long ulBase)
Receives a character from the specified port.

**Parameters**

**ulBase** is the base address of the UART port.

This function gets a character from the receive FIFO for the specified port.

**Returns**

Returns the character read from the specified port, cast as a *long*. A -1 is returned if there are no characters present in the receive FIFO. The **UARTCharsAvail()** function should be called before attempting to call this function.

```c
void UARTCharPut(unsigned long ulBase,
                 unsigned char ucData)
```

Waits to send a character from the specified port.

**Parameters**

**ulBase** is the base address of the UART port.

**ucData** is the character to be transmitted.

This function sends the character **ucData** to the transmit FIFO for the specified port. If there is no space available in the transmit FIFO, this function waits until there is space available before returning.

**Returns**
None.

tBoolean UARTCharPutNonBlocking ( unsigned long ulBase, unsigned char ucData )

Sends a character to the specified port.

**Parameters**
- `ulBase` is the base address of the UART port.
- `ucData` is the character to be transmitted.

This function writes the character `ucData` to the transmit FIFO for the specified port. This function does not block, so if there is no space available, then a `false` is returned, and the application must retry the function later.

**Returns**
- Returns `true` if the character was successfully placed in the transmit FIFO or `false` if there was no space available in the transmit FIFO.

`tBoolean UARTCharsAvail ( unsigned long ulBase )`

Determines if there are any characters in the receive FIFO.

**Parameters**
- `ulBase` is the base address of the UART port.

This function returns a flag indicating whether or not there
is data available in the receive FIFO.

**Returns**

Returns **true** if there is data in the receive FIFO or **false** if there is no data in the receive FIFO.

```c
void UARTConfigGetExpClk ( unsigned long ulBase,
                          unsigned long ulUARTClk,
                          unsigned long * pulBaud,
                          unsigned long * pulConfig )
```

Gets the current configuration of a UART.

**Parameters**

- **ulBase** is the base address of the UART port.
- **ulUARTClk** is the rate of the clock supplied to the UART module.
- **pulBaud** is a pointer to storage for the baud rate.
- **pulConfig** is a pointer to storage for the data format.

The baud rate and data format for the UART is determined, given an explicitly provided peripheral clock (hence the ExpClk suffix). The returned baud rate is the actual baud rate; it may not be the exact baud rate requested or an ``official" baud rate. The data format returned in `pulConfig` is enumerated the same as the `ulConfig` parameter of `UARTConfigSetExpClk()`.

The peripheral clock is the same as the processor clock.
The frequency of the system clock is the value returned by SysCtlClockGet(), or it can be explicitly hard coded if it is constant and known (to save the code/execution overhead of a call to SysCtlClockGet()).

Returns
None.

```c
void UARTConfigSetExpClk (unsigned long ulBase, unsigned long ulUARTClk, unsigned long ulBaud, unsigned long ulConfig )
```

Sets the configuration of a UART.

**Parameters**

- **ulBase** is the base address of the UART port.
- **ulUARTClk** is the rate of the clock supplied to the UART module.
- **ulBaud** is the desired baud rate.
- **ulConfig** is the data format for the port (number of data bits, number of stop bits, and parity).

This function configures the UART for operation in the specified data format. The baud rate is provided in the **ulBaud** parameter and the data format in the **ulConfig** parameter.

The **ulConfig** parameter is the logical OR of three values: the number of data bits, the number of stop bits, and the
parity. \texttt{UART\_CONFIG\_WLEN\_8}, \texttt{UART\_CONFIG\_WLEN\_7}, \texttt{UART\_CONFIG\_WLEN\_6}, and \texttt{UART\_CONFIG\_WLEN\_5} select from eight to five data bits per byte (respectively). \texttt{UART\_CONFIG\_STOP\_ONE} and \texttt{UART\_CONFIG\_STOP\_TWO} select one or two stop bits (respectively). \texttt{UART\_CONFIG\_PAR\_NONE}, \texttt{UART\_CONFIG\_PAR\_EVEN}, \texttt{UART\_CONFIG\_PAR\_ODD}, \texttt{UART\_CONFIG\_PAR\_ONE}, and \texttt{UART\_CONFIG\_PAR\_ZERO} select the parity mode (no parity bit, even parity bit, odd parity bit, parity bit always one, and parity bit always zero, respectively).

The peripheral clock is the same as the processor clock. The frequency of the system clock is the value returned by \texttt{SysCtl\_Clock\_Get()}, or it can be explicitly hard coded if it is constant and known (to save the code/execution overhead of a call to \texttt{SysCtl\_Clock\_Get()}).

\textbf{Returns}

None.

\begin{verbatim}
void UART\_Disable ( unsigned long \texttt{ulBase} )

Disables transmitting and receiving.

\textbf{Parameters}

\begin{verbatim}
\texttt{ulBase} is the base address of the UART port.
\end{verbatim}

This function clears the \texttt{UARTEN}, \texttt{TXE}, and \texttt{RXE} bits, waits for the end of transmission of the current character, and flushes the transmit FIFO.
\end{verbatim}
Returns
None.

```c
void UARTDMADisable( unsigned long ulBase,
                     unsigned long ulDMAFlags )
```

Disable UART DMA operation.

Parameters
- `ulBase` is the base address of the UART port.
- `ulDMAFlags` is a bit mask of the DMA features to disable.

This function is used to disable UART DMA features that were enabled by `UARTDMAEnable()`. The specified UART DMA features are disabled. The `ulDMAFlags` parameter is the logical OR of any of the following values:

- UART_DMA_RX - disable DMA for receive
- UART_DMA_TX - disable DMA for transmit
- UART_DMA_ERR_RXSTOP - do not disable DMA receive on UART error

Returns
None.

```c
void UARTDMAEnable( unsigned long ulBase,
                    unsigned long ulDMAFlags )
```
Enable UART DMA operation.

**Parameters**

- `ulBase` is the base address of the UART port.
- `ulDMAFlags` is a bit mask of the DMA features to enable.

The specified UART DMA features are enabled. The UART can be configured to use DMA for transmit or receive, and to disable receive if an error occurs. The `ulDMAFlags` parameter is the logical OR of any of the following values:

- UART_DMA_RX - enable DMA for receive
- UART_DMA_TX - enable DMA for transmit
- UART_DMA_ERR_RXSTOP - disable DMA receive on UART error

**Note**
The uDMA controller must also be set up before DMA can be used with the UART.

**Returns**
None.

```c
void UARTEnable ( unsigned long ulBase )
```

Enables transmitting and receiving.

**Parameters**

- `ulBase` is the base address of the UART port.

This function sets the UARTEN, TXE, and RXE bits, and
enables the transmit and receive FIFOs.

**Returns**
None.

### void UARTFIFODisable (unsigned long ulBase)

Disables the transmit and receive FIFOs.

**Parameters**
- **ulBase** is the base address of the UART port.

This function disables the transmit and receive FIFOs in the UART.

**Returns**
None.

### void UARTFIFOEnable (unsigned long ulBase)

Enables the transmit and receive FIFOs.

**Parameters**
- **ulBase** is the base address of the UART port.

This function enables the transmit and receive FIFOs in the UART.

**Returns**
None.
void UARTFIFOLevelGet ( unsigned long ulBase,  
               unsigned long * pulTxLevel,  
               unsigned long * pulRxLevel )

Gets the FIFO level at which interrupts are generated.

Parameters

- **ulBase** is the base address of the UART port.
- **pulTxLevel** is a pointer to storage for the transmit FIFO level, returned as one of
  - UART_FIFO_TX1_8,
  - UART_FIFO_TX2_8,
  - UART_FIFO_TX4_8,
  - UART_FIFO_TX6_8, or
  - UART_FIFO_TX7_8.
- **pulRxLevel** is a pointer to storage for the receive FIFO level, returned as one of
  - UART_FIFO_RX1_8,
  - UART_FIFO_RX2_8,
  - UART_FIFO_RX4_8,
  - UART_FIFO_RX6_8, or
  - UART_FIFO_RX7_8.

This function gets the FIFO level at which transmit and receive interrupts are generated.

Returns

- None.

void UARTFIFOLevelSet ( unsigned long ulBase,  
                           unsigned long pulBase,  
                           unsigned long pulTxLevel,  
                           unsigned long pulRxLevel )
unsigned long ulTxLevel, unsigned long ulRxLevel)

Sets the FIFO level at which interrupts are generated.

Parameters
ulBase is the base address of the UART port.
ulTxLevel is the transmit FIFO interrupt level, specified as one of UART_FIFO_TX1_8, UART_FIFO_TX2_8, UART_FIFO_TX4_8, UART_FIFO_TX6_8, or UART_FIFO_TX7_8.
ulRxLevel is the receive FIFO interrupt level, specified as one of UART_FIFO_RX1_8, UART_FIFO_RX2_8, UART_FIFO_RX4_8, UART_FIFO_RX6_8, or UART_FIFO_RX7_8.

This function sets the FIFO level at which transmit and receive interrupts are generated.

Returns
None.

unsigned long UARTFlowControlGet (unsigned long ulBase)

Returns the UART hardware flow control mode currently in use.
Parameters

**ulBase** is the base address of the UART port.

This function returns the current hardware flow control mode.

**Note**
The availability of hardware flow control varies with the part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

**Returns**
Returns the current flow control mode in use. This is a logical OR combination of values

**UART_FLOWCONTROL_TX** if transmit (CTS) flow control is enabled and **UART_FLOWCONTROL_RX** if receive (RTS) flow control is in use. If hardware flow control is disabled, **UART_FLOWCONTROL_NONE** is returned.

```c
void UARTFlowControlSet (unsigned long ulBase,
                             unsigned long ulMode )
```

Sets the UART hardware flow control mode to be used.

**Parameters**

**ulBase** is the base address of the UART port.

**ulMode** indicates the flow control modes to be used.

This parameter is a logical OR combination of values **UART_FLOWCONTROL_TX** and
UART_FLOWCONTROL_RX to enable hardware transmit (CTS) and receive (RTS) flow control or
UART_FLOWCONTROL_NONE to disable hardware flow control.

This function sets the required hardware flow control modes. If ulMode contains flag
UART_FLOWCONTROL_TX, data is only transmitted if the incoming CTS signal is asserted. If ulMode contains flag
UART_FLOWCONTROL_RX, the RTS output is controlled by the hardware and is asserted only when there is space available in the receive FIFO. If no hardware flow control is required, UART_FLOWCONTROL_NONE should be passed.

Note
The availability of hardware flow control varies with the part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns
None.

void UARTIntClear ( unsigned long ulBase,
                   unsigned long ulIntFlags
)

Clears UART interrupt sources.

Parameters
ulBase is the base address of the UART port.
ullIntFlags is a bit mask of the interrupt sources to be cleared.

The specified UART interrupt sources are cleared, so that they no longer assert. This function must be called in the interrupt handler to keep the interrupt from being recognized again immediately upon exit.

The ullIntFlags parameter has the same definition as the ullIntFlags parameter to UARTIntEnable().

**Note**
Because there is a write buffer in the Cortex-M3 processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

**Returns**
None.

```c
void UARTIntDisable ( unsigned long ulBase,
                     unsigned long ullIntFlags )
```

Disables individual UART interrupt sources.
Parameters

ulBase is the base address of the UART port.
ullIntFlags is the bit mask of the interrupt sources to be disabled.

This function 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.

The ullIntFlags parameter has the same definition as the ullIntFlags parameter to UARTIntEnable().

Returns

None.

void UARTIntEnable ( unsigned long ulBase, 
                      unsigned long ullIntFlags 
                    )

Enables individual UART interrupt sources.

Parameters

ulBase is the base address of the UART port.
ullIntFlags is the bit mask of the interrupt sources to be enabled.

This function enables 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.
The *ulIntFlags* parameter is the logical OR of any of the following:

- **UART_INT_OE** - Overrun Error interrupt
- **UART_INT_BE** - Break Error interrupt
- **UART_INT_PE** - Parity Error interrupt
- **UART_INT_FE** - Framing Error interrupt
- **UART_INT_RT** - Receive Timeout interrupt
- **UART_INT_TX** - Transmit interrupt
- **UART_INT_RX** - Receive interrupt
- **UART_INT_CTS** - CTS interrupt

**Returns**

None.

```c
void UARTIntRegister ( unsigned long ulBase,
             void(*)(void) pfnHandler
 )
```

Registers an interrupt handler for a UART interrupt.

**Parameters**

- **ulBase** is the base address of the UART port.
- **pfnHandler** is a pointer to the function to be called when the UART interrupt occurs.

This function does the actual registering of the interrupt handler. This function enables the global interrupt in the interrupt controller; specific UART interrupts must be enabled via `UARTIntEnable()`. It is the interrupt handler's responsibility to clear the interrupt source.
See also `IntRegister()` for important information about registering interrupt handlers.

Returns
None.

```c
unsigned long UARTIntStatus ( unsigned long ulBase, tBoolean bMasked )
```

Gets the current interrupt status.

Parameters
- `ulBase` is the base address of the UART port.
- `bMasked` is `false` if the raw interrupt status is required and `true` if the masked interrupt status is required.

This function returns the interrupt status for the specified UART. Either the raw interrupt status or the status of interrupts that are allowed to reflect to the processor can be returned.

Returns
Returns the current interrupt status, enumerated as a bit field of values described in `UARTIntEnable()`.

```c
void UARTIntUnregister ( unsigned long ulBase )
```

Unregisters an interrupt handler for a UART interrupt.
Parameters

**ulBase** is the base address of the UART port.

This function does the actual unregistering of the interrupt handler. It clears the handler to be called when a UART interrupt occurs. This function also masks off the interrupt in the interrupt controller so that the interrupt handler no longer is called.

See also

[IntRegister()](#) for important information about registering interrupt handlers.

Returns

None.

```c
void UARTModemControlClear (unsigned long ulBase, unsigned long ulControl)
```

Clears the states of the RTS modem control signals.

Parameters

**ulBase** is the base address of the UART port.

**ulControl** is a bit-mapped flag indicating which modem control bits should be set.

This function clears the states of the RTS modem handshake outputs from the UART.

The **ulControl** parameter is the logical OR of any of the
following:

- **UART_OUTPUT_RTS** - The Modem Control RTS signal

**Note**

The availability of hardware modem handshake signals varies with the part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

**Returns**

None.

```c
unsigned long UARTModemControlGet (unsigned long ulBase)
```

Gets the states of the RTS modem control signals.

**Parameters**

- **ulBase** is the base address of the UART port.

This function returns the current states of each of the UART modem control signal, RTS.

**Note**

The availability of hardware modem handshake signals varies with the part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

**Returns**

Returns the states of the handshake output signal.
void UARTModemControlSet ( unsigned long ulBase,  
                          unsigned long ulControl )

Sets the states of the RTS modem control signals.

Parameters
   ulBase  is the base address of the UART port.
   ulControl is a bit-mapped flag indicating which modem control bits should be set.

This function sets the states of the RTS modem handshake outputs from the UART.

The ulControl parameter is the logical OR of any of the following:

- **UART_OUTPUT_RTS** - The Modem Control RTS signal

Note
   The availability of hardware modem handshake signals varies with the part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

Returns
   None.

unsigned long UARTModemStatusGet ( unsigned long ulBase )
Gets the states of the CTS modem status signal.

**Parameters**

ulBase is the base address of the UART port.

This function returns the current states of the UART modem status signal, CTS.

**Note**
The availability of hardware modem handshake signals varies with the part and UART in use. Please consult the datasheet for the part you are using to determine whether this support is available.

**Returns**
Returns the states of the handshake output signal

```c
unsigned long UARTParityModeGet (unsigned long ulBase)
```

Gets the type of parity currently being used.

**Parameters**

ulBase is the base address of the UART port.

This function gets the type of parity used for transmitting data and expected when receiving data.

**Returns**
Returns the current parity settings, specified as one of UART_CONFIG_PAR_NONE,
UART_CONFIG_PAR_EVEN, UART_CONFIG_PAR_ODD, UART_CONFIG_PAR_ONE, or UART_CONFIG_PAR_ZERO.

```c
void UARTParityModeSet ( unsigned long ulBase, unsigned long ulParity )
```

Sets the type of parity.

**Parameters**
- `ulBase` is the base address of the UART port.
- `ulParity` specifies the type of parity to use.

This function sets the type of parity to use for transmitting and expect when receiving. The `ulParity` parameter must be one of UART_CONFIG_PAR_NONE, UART_CONFIG_PAR_EVEN, UART_CONFIG_PAR_ODD, UART_CONFIG_PAR_ONE, or UART_CONFIG_PAR_ZERO. The last two allow direct control of the parity bit; it is always either one or zero based on the mode.

**Returns**
- None.

```c
void UARTRxErrorClear ( unsigned long ulBase )
```

Clears all reported receiver errors.
Parameters

ulBase is the base address of the UART port.

This function is used to clear all receiver error conditions reported via UARTRxErrorGet(). If using the overrun, framing error, parity error or break interrupts, this function must be called after clearing the interrupt to ensure that later errors of the same type trigger another interrupt.

Returns

None.

unsigned long
UARTRxErrorGet ( unsigned long ulBase )

Gets current receiver errors.

Parameters

ulBase is the base address of the UART port.

This function returns the current state of each of the 4 receiver error sources. The returned errors are equivalent to the four error bits returned via the previous call to UARTCharGet() or UARTCharGetNonBlocking() with the exception that the overrun error is set immediately the overrun occurs rather than when a character is next read.

Returns

Returns a logical OR combination of the receiver error flags, UART_RXERROR_FRAMING, UART_RXERROR_PARITY, UART_RXERROR_BREAK and
tBoolean UARTSpaceAvail (unsigned long ulBase)

Determines if there is any space in the transmit FIFO.

Parameters
ulBase is the base address of the UART port.

This function returns a flag indicating whether or not there is space available in the transmit FIFO.

Returns
Returns true if there is space available in the transmit FIFO or false if there is no space available in the transmit FIFO.

unsigned long UARTTxIntModeGet (unsigned long ulBase)

Returns the current operating mode for the UART transmit interrupt.

Parameters
ulBase is the base address of the UART port.

This function returns the current operating mode for the UART transmit interrupt. The return value is UART_TXINT_MODE_EOT if the transmit interrupt is currently set to be asserted once the transmitter is completely idle - the transmit FIFO is empty and all bits,
including any stop bits, have cleared the transmitter. The return value is **UART_TXINT_MODE_FIFO** if the interrupt is set to be asserted based upon the level of the transmit FIFO.

**Note**
The availability of end-of-transmission mode varies with the part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

**Returns**
Returns **UART_TXINT_MODE_FIFO** or **UART_TXINT_MODE_EOT**.

```c
void UARTTxIntModeSet (unsigned long ulBase,
                       unsigned long ulMode)
```

Sets the operating mode for the UART transmit interrupt.

**Parameters**

- **ulBase** is the base address of the UART port.
- **ulMode** is the operating mode for the transmit interrupt. It may be **UART_TXINT_MODE_EOT** to trigger interrupts when the transmitter is idle or **UART_TXINT_MODE_FIFO** to trigger based on the current transmit FIFO level.

This function allows the mode of the UART transmit interrupt to be set. By default, the transmit interrupt is
asserted when the FIFO level falls past a threshold set via a call to `UARTFIFOLeveleSet()`. Alternatively, if this function is called with `ulMode` set to `UART_TXINT_MODE_EOT`, the transmit interrupt is asserted once the transmitter is completely idle - the transmit FIFO is empty and all bits, including any stop bits, have cleared the transmitter.

**Note**

The availability of end-of-transmission mode varies with the part in use. Please consult the datasheet for the part you are using to determine whether this support is available.

**Returns**

None.
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>UDMA_Micro_Direct_Memory_Access_api</td>
<td></td>
</tr>
</tbody>
</table>
Data Structures

<table>
<thead>
<tr>
<th>struct</th>
<th>tDMAControlTable</th>
</tr>
</thead>
</table>

### Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void uDMAEnable (void)</td>
<td>Enable uDMA functionality.</td>
</tr>
<tr>
<td>void uDMADisable (void)</td>
<td>Disable uDMA functionality.</td>
</tr>
<tr>
<td>unsigned long uDMAErrorStatusGet (void)</td>
<td>Get uDMA error status.</td>
</tr>
<tr>
<td>void uDMAErrorStatusClear (void)</td>
<td>Clear uDMA error status.</td>
</tr>
<tr>
<td>void uDMAChannelEnable (unsigned long ulChannelNum)</td>
<td>Enable a specific uDMA channel.</td>
</tr>
<tr>
<td>void uDMAChannelDisable (unsigned long ulChannelNum)</td>
<td>Disable a specific uDMA channel.</td>
</tr>
<tr>
<td>tBoolean uDMAChannelIsEnabled (unsigned long ulChannelNum)</td>
<td>Check if a specific uDMA channel is enabled.</td>
</tr>
<tr>
<td>void uDMAControlBaseSet (void *pControlTable)</td>
<td>Set uDMA control base address.</td>
</tr>
<tr>
<td>void * uDMAControlBaseGet (void)</td>
<td>Get uDMA control base address.</td>
</tr>
<tr>
<td>void * uDMAControlAlternateBaseGet (void)</td>
<td>Get uDMA alternate control base address.</td>
</tr>
<tr>
<td>void uDMAChannelRequest (unsigned long ulChannelNum)</td>
<td>Request uDMA transfer on a specific channel.</td>
</tr>
<tr>
<td>void uDMAChannelAttributeEnable (unsigned long ulChannelNum, unsigned long ulAttr)</td>
<td>Enable uDMA channel attributes.</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>void uDMAChannelAttributeDisable</td>
<td>(unsigned long ulChannelNum, unsigned long ulAttr)</td>
</tr>
<tr>
<td>unsigned long uDMAChannelAttributeGet</td>
<td>(unsigned long ulChannelNum)</td>
</tr>
<tr>
<td>void uDMAChannelControlSet</td>
<td>(unsigned long ulChannelStructIndex, unsigned long ulControl)</td>
</tr>
<tr>
<td>void uDMAChannelTransferSet</td>
<td>(unsigned long ulChannelStructIndex, unsigned long ulMode, void *pvSrcAddr, void *pvDstAddr, unsigned long ulTransferSize)</td>
</tr>
<tr>
<td>void uDMAChannelScatterGatherSet</td>
<td>(unsigned long ulChannelNum, unsigned ulTaskCount, void *pvTaskList, unsigned long ulIsPeriphSG)</td>
</tr>
<tr>
<td>unsigned long uDMAChannelSizeGet</td>
<td>(unsigned long ulChannelStructIndex)</td>
</tr>
<tr>
<td>unsigned long uDMAChannelModeGet</td>
<td>(unsigned long ulChannelStructIndex)</td>
</tr>
<tr>
<td>void uDMAIntRegister</td>
<td>(unsigned long ulIntChannel, void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>void uDMAIntUnregister</td>
<td>(unsigned long ulIntChannel)</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>unsigned long <code>uDMAIntStatus</code> (void)</td>
<td></td>
</tr>
<tr>
<td>void <code>uDMAIntClear</code> (unsigned long <code>ulChanMask</code>)</td>
<td></td>
</tr>
<tr>
<td>void <code>uDMAChannelAssign</code> (unsigned long <code>ulMapping</code>)</td>
<td></td>
</tr>
</tbody>
</table>
void uDMAChannelAssign ( unsigned long ulMapping )

Assigns a peripheral mapping for a uDMA channel.

Parameters

ulMapping is a macro specifying the peripheral assignment for a channel.

This function assigns a peripheral mapping to a uDMA channel. It is used to select which peripheral is used for a uDMA channel. The parameter ulMapping should be one of the macros named UDMA_CHn_tttt from the header file udma.h. For example, to assign uDMA channel 0 to the UART2 RX channel, the parameter should be the macro UDMA_CH0_UART2RX.

Please consult the data sheet for a table showing all the possible peripheral assignments for the uDMA channels for a particular device.

Note

This function is only available on devices that have the DMA Channel Map Select registers (DMACHMAP0-3). Please consult the data sheet for your part.
Returns
None.

```c
void uDMAChannelAttributeDisable ( unsigned long ulChannel
                                        unsigned long ulAttr
                      )
```

Disables attributes of a uDMA channel.

Parameters
- **ulChannelNum** is the channel to configure.
- **ulAttr** is a combination of attributes for the channel.

This function is used to disable attributes of a uDMA channel.

The *ulAttr* parameter is the logical OR of any of the following:

- **UDMA_ATTR_USEBURST** is used to restrict transfers to only burst mode.
- **UDMA_ATTR_ALTSELECT** is used to select the alternate control structure for this channel.
- **UDMA_ATTR_HIGH_PRIORITY** is used to set this channel to high priority.
- **UDMA_ATTR_REQMASK** is used to mask the hardware request signal from the peripheral for this channel.

Returns
None.

```c
void
```
uDMAChannelAttributeEnable (unsigned long ulChannelNum unsigned long ulAttr)

Enables attributes of a uDMA channel.

Parameters

ulChannelNum is the channel to configure.
ulAttr is a combination of attributes for the channel.

This function is used to enable attributes of a uDMA channel.

The ulAttr parameter is the logical OR of any of the following:

- **UDMA_ATTR_USEBURST** is used to restrict transfers to only burst mode.
- **UDMA_ATTR_ALTSELECT** is used to select the alternate control structure for this channel (it is very unlikely that this flag should be used).
- **UDMA_ATTR_HIGH_PRIORITY** is used to set this channel to high priority.
- **UDMA_ATTR_REQMASK** is used to mask the hardware request signal from the peripheral for this channel.

Returns

None.

unsigned long uDMAChannelAttributeGet (unsigned long ulChannelNum

Gets the enabled attributes of a uDMA channel.
Parameters

ulChannelNum is the channel to configure.

This function returns a combination of flags representing the attributes of the uDMA channel.

Returns

Returns the logical OR of the attributes of the uDMA channel, which can be any of the following:

- **UDMA_ATTR_USEBURST** is used to restrict transfers to use only burst mode.
- **UDMA_ATTR_ALTSELECT** is used to select the alternate control structure for this channel.
- **UDMA_ATTR_HIGH_PRIORITY** is used to set this channel to high priority.
- **UDMA_ATTR_REQMASK** is used to mask the hardware request signal from the peripheral for this channel.

```c
void uDMAChannelControlSet (unsigned long ulChannelStructIndex, unsigned long ulControl)
```

Sets the control parameters for a uDMA channel control structure.

Parameters

ulChannelStructIndex is the logical OR of the uDMA channel number with **UDMA_PRI_SELECT** and **UDMA_ALT_SELECT**.

ulControl is logical OR of several control values to set the control parameters for the channel.
This function is used to set control parameters for a uDMA transfer. These parameters are typically not changed often.

The `ulChannelStructIndex` parameter should be the logical OR of the channel number with one of `UDMA_PRI_SELECT` or `UDMA_ALT_SELECT` to choose whether the primary or alternate data structure is used.

The `ulControl` parameter is the logical OR of five values: the data size, the source address increment, the destination address increment, the arbitration size, and the use burst flag. The choices available for each of these values is described below.

Choose the data size from one of `UDMA_SIZE_8`, `UDMA_SIZE_16` or `UDMA_SIZE_32` to select a data size of 8, 16, or 32 bits.

Choose the source address increment from one of `UDMA_SRC_INC_8`, `UDMA_SRC_INC_16`, `UDMA_SRC_INC_32` or `UDMA_SRC_INC_NONE` to select an address increment of 8-bit bytes, 16-bit halfwords, 32-bit words, or to select non-incrementing.

Choose the destination address increment from one of `UDMA_DST_INC_8`, `UDMA_DST_INC_16`, `UDMA_DST_INC_32` or `UDMA_DST_INC_NONE` to select an address increment of 8-bit bytes, 16-bit halfwords, 32-bit words, or to select non-incrementing.

The arbitration size determines how many items are transferred before the uDMA controller re-arbitrates for the bus. Choose the arbitration size from one of `UDMA_ARB_1`, `UDMA_ARB_2`, `UDMA_ARB_4`, `UDMA_ARB_8`, through `UDMA_ARB_1024` to select the arbitration size from 1 to 1024 items, in powers of 2.
The value **UDMA_NEXT_USEBURST** is used to force the channel to only respond to burst requests at the tail end of a scatter-gather transfer.

**Note**

The address increment cannot be smaller than the data size.

**Returns**

None.

```c
void uDMAChannelDisable (unsigned long ulChannelNum)
```

Disables a uDMA channel for operation.

**Parameters**

*ulChannelNum* is the channel number to disable.

This function disables a specific uDMA channel. Once disabled, a channel cannot respond to uDMA transfer requests until re-enabled via **uDMAChannelEnable()**.

**Returns**

None.

```c
void uDMAChannelEnable (unsigned long ulChannelNum)
```

Enables a uDMA channel for operation.

**Parameters**
ulChannelNum is the channel number to enable.

This function enables a specific uDMA channel for use. This function must be used to enable a channel before it can be used to perform a uDMA transfer.

When a uDMA transfer is completed, the channel is automatically disabled by the uDMA controller. Therefore, this function should be called prior to starting up any new transfer.

Returns
None.

tBoolean uDMAChannelIsEnabled ( unsigned long ulChannelNum )

Checks if a uDMA channel is enabled for operation.

Parameters
ulChannelNum is the channel number to check.

This function checks to see if a specific uDMA channel is enabled. This function can be used to check the status of a transfer, as the channel is automatically disabled at the end of a transfer.

Returns
Returns true if the channel is enabled, false if disabled.

unsigned long uDMAChannelModeGet ( unsigned long ulChannelStructIndex )
Gets the transfer mode for a uDMA channel control structure.

**Parameters**

`ulChannelStructIndex` is the logical OR of the uDMA channel number with either
`UDMA_PRI_SELECT` or `UDMA_ALT_SELECT`.

This function is used to get the transfer mode for the uDMA channel and to query the status of a transfer on a channel. When the transfer is complete the mode is **UDMA_MODE_STOP**.

**Returns**

Returns the transfer mode of the specified channel and control structure, which is one of the following values:
- `UDMA_MODE_STOP`
- `UDMA_MODE_BASIC`
- `UDMA_MODE_AUTO`
- `UDMA_MODE_PINGPONG`
- `UDMA_MODE_MEM_SCATTER_GATHER`
- `UDMA_MODE_PER_SCATTER_GATHER`.

```c
void
uDMAChannelRequest ( unsigned long ulChannelNum )
```

Requests a uDMA channel to start a transfer.

**Parameters**

`ulChannelNum` is the channel number on which to request a uDMA transfer.

This function allows software to request a uDMA channel to begin a transfer. This function could be used for performing
a memory-to-memory transfer or if for some reason, a transfer needs to be initiated by software instead of the peripheral associated with that channel.

**Note**
If the channel is UDMA_CHANNEL_SW and interrupts are used, then the completion is signaled on the uDMA dedicated interrupt. If a peripheral channel is used, then the completion is signaled on the peripheral's interrupt.

**Returns**
None.

```c
void uDMAChannelScatterGatherSet ( unsigned long ulChannelNum, unsigned ulTaskCount, void *pvTaskList, unsigned long ulIsPeriphSG )
```

Configures a uDMA channel for scatter-gather mode.

**Parameters**
- `ulChannelNum` is the uDMA channel number.
- `ulTaskCount` is the number of scatter-gather tasks to execute.
- `pvTaskList` is a pointer to the beginning of the scatter-gather task list.
- `ulIsPeriphSG` is a flag to indicate it is a peripheral scatter-gather transfer (else it is memory scatter-gather transfer).
This function is used to configure a channel for scatter-gather mode. The caller must have already set up a task list and must pass a pointer to the start of the task list as the `pvTaskList` parameter. The `ulTaskCount` parameter is the count of tasks in task list, not the size of the task list. The flag `bIsPeriphSG` should be used to indicate if scatter-gather should be configured for peripheral or memory operation.

See also
- `uDMATaskStructEntry`

Returns
None.

```c
unsigned long uDMAChannelSizeGet (unsigned long ulChannelStructIndex)
```

Gets the current transfer size for a uDMA channel control structure.

**Parameters**

- `ulChannelStructIndex` is the logical OR of the uDMA channel number with either `UDMA_PRI_SELECT` or `UDMA_ALT_SELECT`.

This function is used to get the uDMA transfer size for a channel. The transfer size is the number of items to transfer, where the size of an item might be 8, 16, or 32 bits. If a partial transfer has already occurred, then the number of remaining items is returned. If the transfer is complete, then 0 is returned.
Returns
Returns the number of items remaining to transfer.

```c
void uDMAChannelTransferSet ( unsigned long ulChannelStructIndex, unsigned long ulMode, void * pvSrcAddr, void * pvDstAddr, unsigned long ulTransferSize )
```

Sets the transfer parameters for a uDMA channel control structure.

Parameters

- **ulChannelStructIndex** is the logical OR of the uDMA channel number with either `UDMA_PRI_SELECT` or `UDMA_ALT_SELECT`.
- **ulMode** is the type of uDMA transfer.
- **pvSrcAddr** is the source address for the transfer.
- **pvDstAddr** is the destination address for the transfer.
- **ulTransferSize** is the number of data items to transfer.

This function is used to configure the parameters for a uDMA transfer. These parameters are typically changed often. The function `uDMAChannelControlSet()` MUST be called at least once for this channel prior to calling this function.

The **ulChannelStructIndex** parameter should be the logical OR of the channel number with one of `UDMA_PRI_SELECT` or `UDMA_ALT_SELECT`. The **ulMode** parameter can be `UDMA_DMA_TRANSFER_MODE`, `UDMA_PERF_TRANSFER_MODE`, or `UDMA_PERF_DMA_TRANSFER_MODE`.
UDMA_ALT_SELECT to choose whether the primary or alternate data structure is used.

The *ulMode* parameter should be one of the following values:

- **UDMA_MODE_STOP** stops the uDMA transfer. The controller sets the mode to this value at the end of a transfer.
- **UDMA_MODE_BASIC** to perform a basic transfer based on request.
- **UDMA_MODE_AUTO** to perform a transfer that always completes once started even if the request is removed.
- **UDMA_MODE_PINGPONG** to set up a transfer that switches between the primary and alternate control structures for the channel. This mode allows use of ping-pong buffering for transfers.
- **UDMA_MODE_MEM_SCATTER_GATHER** to set up a memory scatter-gather transfer.
- **UDMA_MODE_PER_SCATTER_GATHER** to set up a peripheral scatter-gather transfer.

The *pvSrcAddr* and *pvDstAddr* parameters are pointers to the location of the data to be transferred. These addresses should be aligned according to the item size. The compiler takes care of alignment if the pointers are pointing to storage of the appropriate data type.

The *ulTransferSize* parameter is the number of data items, not the number of bytes. The value of this parameter should not exceed 1024.

The two scatter-gather modes, memory and peripheral, are actually different depending on whether the primary or alternate control structure is selected. This function looks for the **UDMA_PRI_SELECT** and **UDMA_ALT_SELECT** flag along with
channel number and sets the scatter-gather mode as appropriate for the primary or alternate control structure.

The channel must also be enabled using `uDMAChannelEnable()` after calling this function. The transfer does not begin until the channel has been configured and enabled. Note that the channel is automatically disabled after the transfer is completed, meaning `uDMAChannelEnable()` must be called again after setting up the next transfer.

**Note**

Great care must be taken to not modify a channel control structure that is in use or else the results are unpredictable, including the possibility of undesired data transfers to or from memory or peripherals. For BASIC and AUTO modes, it is safe to make changes when the channel is disabled, or the `uDMAChannelModeGet()` function returns `UDMA_MODE_STOP`. For PINGPONG or one of the SCATTER_GATHER modes, it is safe to modify the primary or alternate control structure only when the other is being used. The `uDMAChannelModeGet()` function returns `UDMA_MODE_STOP` when a channel control structure is inactive and safe to modify.

**Returns**

None.

```c
void* uDMAControlAlternateBaseGet ( void )
```

Gets the base address for the channel control table alternate structures.

This function gets the base address of the second half of
the channel control table that holds the alternate control structures for each channel.

**Returns**
Returns a pointer to the base address of the second half of the channel control table.

### void* uDMAControlBaseGet ( void )

Gets the base address for the channel control table.

This function gets the base address of the channel control table. This table resides in system memory and holds control information for each uDMA channel.

**Returns**
Returns a pointer to the base address of the channel control table.

### void uDMAControlBaseSet ( void * pControlTable )

Sets the base address for the channel control table.

**Parameters**
- **pControlTable** is a pointer to the 1024-byte-aligned base address of the uDMA channel control table.

This function configures the base address of the channel control table. This table resides in system memory and holds control information for each uDMA channel. The table
must be aligned on a 1024-byte boundary. The base address must be configured before any of the channel functions can be used.

The size of the channel control table depends on the number of uDMA channels and the transfer modes that are used. Refer to the introductory text and the microcontroller datasheet for more information about the channel control table.

Returns
None.

void uDMADisable ( void )

Disables the uDMA controller for use.

This function disables the uDMA controller. Once disabled, the uDMA controller cannot operate until re-enabled with uDMAEnable().

Returns
None.

void uDMAEnable ( void )

Enables the uDMA controller for use.

This function enables the uDMA controller. The uDMA controller must be enabled before it can be configured and used.
Returns
None.

```c
void uDMAErrorStatusClear ( void )
```

Clears the uDMA error interrupt.

This function clears a pending uDMA error interrupt. This function should be called from within the uDMA error interrupt handler to clear the interrupt.

Returns
None.

```c
unsigned long uDMAErrorStatusGet ( void )
```

Gets the uDMA error status.

This function returns the uDMA error status. It should be called from within the uDMA error interrupt handler to determine if a uDMA error occurred.

Returns
Returns non-zero if a uDMA error is pending.

```c
void uDMAIntClear ( unsigned long ulChanMask )
```

Clears uDMA interrupt status.

Parameters
**ulChanMask** is a 32-bit mask with one bit for each uDMA channel.

This function clears bits in the uDMA interrupt status register according to which bits are set in *ulChanMask*. There is one bit for each channel. If a a bit is set in *ulChanMask*, then that corresponding channel's interrupt status is cleared (if it was set).

**Note**
This function is only available on devices that have the DMA Channel Interrupt Status Register (DMACHIS). Please consult the data sheet for your part.

**Returns**
None.

```c
void uDMAIntRegister ( unsigned long ulIntChannel, 
                        void(*)(void) pfnHandler )
```

Registers an interrupt handler for the uDMA controller.

**Parameters**
- **ulIntChannel** identifies which uDMA interrupt is to be registered.
- **pfnHandler** is a pointer to the function to be called when the interrupt is activated.

This function registers and enables the handler to be called when the uDMA controller generates an interrupt. The *ulIntChannel* parameter should be one of the following:
- **UDMA_INT_SW** to register an interrupt handler to process interrupts from the uDMA software channel (UDMA_CHANNEL_SW)
- **UDMA_INT_ERR** to register an interrupt handler to process uDMA error interrupts

See also

[**IntRegister()**](#) for important information about registering interrupt handlers.

**Note**

The interrupt handler for the uDMA is for transfer completion when the channel UDMA_CHANNEL_SW is used and for error interrupts. The interrupts for each peripheral channel are handled through the individual peripheral interrupt handlers.

**Returns**

None.

```c
unsigned long uDMAIntStatus ( void )
```

Gets the uDMA controller channel interrupt status.

This function is used to get the interrupt status of the uDMA controller. The returned value is a 32-bit bit mask that indicates which channels are requesting an interrupt. This function can be used from within an interrupt handler to determine or confirm which uDMA channel has requested an interrupt.

**Note**

This function is only available on devices that have the
DMA Channel Interrupt Status Register (DMACHIS).
Please consult the data sheet for your part.

**Returns**
Returns a 32-bit mask which indicates requesting uDMA channels. There is a bit for each channel and a 1 indicates that the channel is requesting an interrupt. Multiple bits can be set.

```c
void uDMAIntUnregister ( unsigned long ulIntChannel )
```

Unregisters an interrupt handler for the uDMA controller.

**Parameters**
- `ulIntChannel` identifies which uDMA interrupt to unregister.

This function disables and unregisters the handler to be called for the specified uDMA interrupt. The `ulIntChannel` parameter should be one of **UDMA_INT_SW** or **UDMA_INT_ERR** as documented for the function `uDMAIntRegister()`.

**See also**
- `IntRegister()` for important information about registering interrupt handlers.

**Returns**
None.
## Data Structures

### tDMAControlTable

**Struct Reference**

UDMA_Micro_Direct_Memory_Access_api
## Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>volatile void *</td>
<td>pvSrcEndAddr</td>
</tr>
<tr>
<td>volatile void *</td>
<td>pvDstEndAddr</td>
</tr>
<tr>
<td>volatile unsigned long</td>
<td>ulControl</td>
</tr>
<tr>
<td>volatile unsigned long</td>
<td>ulSpare</td>
</tr>
</tbody>
</table>

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

- D:/D-Drive/ti/CC3200SDK_1.2.0/driverlib/udma.h

Generated on Thu Feb 18 2016 13:22:02 for CC3200 Peripheral Driver Library User's Guide by **doxygen** 1.8.11
<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Utils_api</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void __attribute__((naked))</code></td>
<td></td>
</tr>
</tbody>
</table>
| `void __asm (" .sect ".text:UtilsDelay\"n\" .clink\n\
.thumbfunc UtilsDelay\"n\" .thumb\n\" global
UtilsDelay\n\"UtilsDelay: \n\" subs r0, #1\n\" bne.n
UtilsDelay\n\" bx lr\n")` |                                                  |
void UtilsDelay ( unsigned long ulCount )

Provides a small delay.

Parameters

ulCount is the number of delay loop iterations to perform.

This function provides a means of generating a constant length delay. It is written in assembly to keep the delay consistent across tool chains, avoiding the need to tune the delay based on the tool chain in use.

The loop takes 3 cycles/loop.

Returns

None.
## WDT_Watchdog_Timer_api

The `WDT_Watchdog_Timer_api` is part of the CC3200 Peripheral Driver Library User's Guide 1.2.0. This section likely discusses the functionality and usage of the Watchdog Timer API within the CC3200 microcontroller platform. The Watchdog Timer is a critical feature for ensuring system stability and preventing unexpected resets when the microcontroller is out of control or in an unresponsive state.

### Table of Contents

- **Modules**
- **Data Structures**
- **Files**
- **Functions**

The **WDT_Watchdog_Timer_api** provides hooks for configuring and using the Watchdog Timer, including setting its enabled state, timeout value, and enabling or disabling the reset on timeout feature.
# Functions

<table>
<thead>
<tr>
<th>Type</th>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tBoolean</td>
<td>WatchdogRunning</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void</td>
<td>WatchdogEnable</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void</td>
<td>WatchdogLock</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void</td>
<td>WatchdogUnlock</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>tBoolean</td>
<td>WatchdogLockState</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void</td>
<td>WatchdogReloadSet</td>
<td>(unsigned long ulBase, unsigned long ulLoadVal)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>WatchdogReloadGet</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>WatchdogValueGet</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>void</td>
<td>WatchdogIntRegister</td>
<td>(unsigned long ulBase, void(*pfnHandler)(void))</td>
</tr>
<tr>
<td>void</td>
<td>WatchdogIntUnregister</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>WatchdogIntStatus</td>
<td>(unsigned long ulBase, tBoolean bMasked)</td>
</tr>
<tr>
<td>void</td>
<td>WatchdogIntClear</td>
<td>(unsigned long ulBase)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>void <code>WatchdogStallEnable</code> (unsigned long ulBase)</td>
<td>Enables the watchdog timer with a base address.</td>
<td></td>
</tr>
<tr>
<td>void <code>WatchdogStallDisable</code> (unsigned long ulBase)</td>
<td>Disables the watchdog timer using the base address.</td>
<td></td>
</tr>
</tbody>
</table>
Detailed Description

Function Documentation

void WatchdogEnable ( unsigned long ulBase )

Enables the watchdog timer.

Parameters
  ulBase is the base address of the watchdog timer module.

This will enable the watchdog timer counter and interrupt.

Note
  This function will have no effect if the watchdog timer has been locked.

See also
  WatchdogLock(), WatchdogUnlock()

Returns
  None.

void WatchdogIntClear ( unsigned long ulBase )

Clears the watchdog timer interrupt.
Parameters

\textbf{ulBase} is the base address of the watchdog timer module.

The watchdog timer interrupt source is cleared, so that it no longer asserts.

\textbf{Note}

Because there is a write buffer in the Cortex-M3 processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

\textbf{Returns}

None.

\begin{verbatim}
void WatchdogIntRegister ( unsigned long ulBase, 
                          void(*)(void) pfnHandler 
) 
\end{verbatim}

Registers an interrupt handler for watchdog timer interrupt.

Parameters

\textbf{ulBase} is the base address of the watchdog timer module.

\textbf{pfnHandler} is a pointer to the function to be called
when the watchdog timer interrupt occurs.

This function does the actual registering of the interrupt handler. This will enable the global interrupt in the interrupt controller; the watchdog timer interrupt must be enabled via WatchdogEnable(). It is the interrupt handler's responsibility to clear the interrupt source via WatchdogIntClear().

See also

IntRegister() for important information about registering interrupt handlers.

Note

This function will only register the standard watchdog interrupt handler. To register the NMI watchdog handler, use IntRegister() to register the handler for the FAULT_NMI interrupt.

Returns

None.

unsigned long WatchdogIntStatus ( unsigned long ulBase, tBoolean bMasked )

Gets the current watchdog timer interrupt status.

Parameters

ulBase is the base address of the watchdog timer module.
bMasked is **false** if the raw interrupt status is required and **true** if the masked interrupt status is required.

This returns the interrupt status for the watchdog timer module. Either the raw interrupt status or the status of interrupt that is allowed to reflect to the processor can be returned.

**Returns**

Returns the current interrupt status, where a 1 indicates that the watchdog interrupt is active, and a 0 indicates that it is not active.

```c
void WatchdogIntUnregister ( unsigned long ulBase )
```

Unregisters an interrupt handler for the watchdog timer interrupt.

**Parameters**

- **ulBase** is the base address of the watchdog timer module.

This function does the actual unregistering of the interrupt handler. This function will clear the handler to be called when a watchdog timer interrupt occurs. This will also mask off the interrupt in the interrupt controller so that the interrupt handler no longer is called.

**See also**

- **IntRegister()** for important information about registering interrupt handlers.
Note
This function will only unregister the standard watchdog interrupt handler. To unregister the NMI watchdog handler, use `IntUnregister()` to unregister the handler for the `FAULT_NMI` interrupt.

Returns
None.

`void WatchdogLock ( unsigned long ulBase )`

Enables the watchdog timer lock mechanism.

Parameters
`ulBase` is the base address of the watchdog timer module.

Locks out write access to the watchdog timer configuration registers.

Returns
None.

`tBoolean WatchdogLockState ( unsigned long ulBase )`

Gets the state of the watchdog timer lock mechanism.

Parameters
`ulBase` is the base address of the watchdog timer module.
Returns the lock state of the watchdog timer registers.

**Returns**

Returns **true** if the watchdog timer registers are locked, and **false** if they are not locked.

```c
unsigned long WatchdogReloadGet ( unsigned long ulBase )
```

Gets the watchdog timer reload value.

**Parameters**

- **ulBase** is the base address of the watchdog timer module.

This function gets the value that is loaded into the watchdog timer when the count reaches zero for the first time.

**See also**

- *WatchdogReloadSet()*

**Returns**

None.

```c
void WatchdogReloadSet ( unsigned long ulBase, unsigned long ulLoadVal )
```

Sets the watchdog timer reload value.
Parameters

ulBase is the base address of the watchdog timer module.

ulLoadVal is the load value for the watchdog timer.

This function sets the value to load into the watchdog timer when the count reaches zero for the first time; if the watchdog timer is running when this function is called, then the value will be immediately loaded into the watchdog timer counter. If the ulLoadVal parameter is 0, then an interrupt is immediately generated.

Note

This function will have no effect if the watchdog timer has been locked.

See also

WatchdogLock(), WatchdogUnlock(), WatchdogReloadGet()

Returns

None.

tBoolean WatchdogRunning ( unsigned long  ulBase )

Determines if the watchdog timer is enabled.

Parameters

ulBase is the base address of the watchdog timer module.

This will check to see if the watchdog timer is enabled.
Returns
Returns true if the watchdog timer is enabled, and false if it is not.

void WatchdogStallDisable ( unsigned long ulBase )

Disables stalling of the watchdog timer during debug events.

Parameters
ulBase is the base address of the watchdog timer module.

This function disables the debug mode stall of the watchdog timer. By doing so, the watchdog timer continues to count regardless of the processor debug state.

Returns
None.

void WatchdogStallEnable ( unsigned long ulBase )

Enables stalling of the watchdog timer during debug events.

Parameters
ulBase is the base address of the watchdog timer module.

This function allows the watchdog timer to stop counting when the processor is stopped by the debugger. By doing
so, the watchdog is prevented from expiring (typically almost immediately from a human time perspective) and resetting the system (if reset is enabled). The watchdog will instead expired after the appropriate number of processor cycles have been executed while debugging (or at the appropriate time after the processor has been restarted).

**Returns**

None.

```c
void WatchdogUnlock ( unsigned long ulBase )
```

Disables the watchdog timer lock mechanism.

**Parameters**

`ulBase` is the base address of the watchdog timer module.

Enables write access to the watchdog timer configuration registers.

**Returns**

None.

```c
unsigned long WatchdogValueGet ( unsigned long ulBase )
```

Gets the current watchdog timer value.

**Parameters**

`ulBase` is the base address of the watchdog timer
module.

This function reads the current value of the watchdog timer.

**Returns**

Returns the current value of the watchdog timer.
Data Structures

Here are the data structures with brief descriptions:

- `_PRCM_PeripheralRegs_`
- `tDMAControlTable`
- `uVectorEntry`

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Structures</td>
<td>Data Structure Index</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

_PinCM_PeripheralRegs_ Struct Reference
Data Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned long</td>
<td>ulClkReg</td>
</tr>
<tr>
<td>unsigned long</td>
<td>ulRstReg</td>
</tr>
</tbody>
</table>

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

- D:/D-Drive/ti/CC3200SDK_1.2.0/driverlib/prcm.h

## uVectorEntry Union Reference

<table>
<thead>
<tr>
<th>Data Structures</th>
<th>Data Structure Index</th>
<th>Data Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>uVectorEntry Union</td>
<td>Reference</td>
<td></td>
</tr>
</tbody>
</table>
Data Fields

void(* pfnHandler )(void)

unsigned long ulPtr

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

- D:/D-Drive/ti/CC3200SDK_1.2.0/driverlib/interrupt.h

## Data Structure Index

<table>
<thead>
<tr>
<th>T</th>
<th>U</th>
<th>_</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>PRCM_PeripheralRegs</em></td>
<td>tDMAControlTable</td>
<td>uVectorE</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>driverlib</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>adc.h</td>
<td></td>
</tr>
<tr>
<td>aes.h</td>
<td></td>
</tr>
<tr>
<td>camera.h</td>
<td></td>
</tr>
<tr>
<td>cpu.h</td>
<td></td>
</tr>
<tr>
<td>crc.h</td>
<td></td>
</tr>
<tr>
<td>debug.h</td>
<td></td>
</tr>
<tr>
<td>des.h</td>
<td></td>
</tr>
<tr>
<td>flash.h</td>
<td></td>
</tr>
<tr>
<td>gpio.h</td>
<td></td>
</tr>
<tr>
<td>hwspinlock.h</td>
<td></td>
</tr>
<tr>
<td>i2c.h</td>
<td></td>
</tr>
<tr>
<td>i2s.h</td>
<td></td>
</tr>
<tr>
<td>interrupt.h</td>
<td></td>
</tr>
<tr>
<td>pin.h</td>
<td></td>
</tr>
<tr>
<td>prcm.h</td>
<td></td>
</tr>
<tr>
<td>rom.h</td>
<td></td>
</tr>
<tr>
<td>rom_map.h</td>
<td></td>
</tr>
<tr>
<td>Header File</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>rom_patch.h</td>
<td></td>
</tr>
<tr>
<td>sdhost.h</td>
<td></td>
</tr>
<tr>
<td>shamd5.h</td>
<td></td>
</tr>
<tr>
<td>spi.h</td>
<td></td>
</tr>
<tr>
<td>systick.h</td>
<td></td>
</tr>
<tr>
<td>timer.h</td>
<td></td>
</tr>
<tr>
<td>uart.h</td>
<td></td>
</tr>
<tr>
<td>udma.h</td>
<td></td>
</tr>
<tr>
<td>utils.h</td>
<td></td>
</tr>
<tr>
<td>version.h</td>
<td></td>
</tr>
<tr>
<td>wdt.h</td>
<td></td>
</tr>
</tbody>
</table>

**MainPage**

**mainpage.h**

<table>
<thead>
<tr>
<th>Main Page</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>driverlib</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**driverlib Directory Reference**
adc.h

1 //******************************************************************************
2 **************************
3 // adc.h
4 //
5 // Defines and Macros for the ADC.
6 //
7 // Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
8 //
9 // Redistributions and use in source and binary forms, with or without
10 // modification, are permitted provided that the following conditions
11 // are met:
12 //
13 // Redistributions of source code must retain the above copyright
14 // notice, this list of conditions and the following disclaimer.
15 //
16 // Redistributions in binary form must
reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the
distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```c
//ifndef __ADC_H__
#define __ADC_H__

//******************************************
***********************************
//
// If building with a C++ compiler, make all of the definitions in this header
// have a C binding.
//
//******************************************
***********************************
#ifdef __cplusplus
extern "C"
{
#endif

//******************************************
***********************************
// Values that can be passed to APIs as ulChannel parameter
//******************************************
***********************************
#define ADC_CH_0   0x00000000
#define ADC_CH_1   0x00000008
#define ADC_CH_2   0x00000010
#define ADC_CH_3   0x00000018

//******************************************
***********************************
// Values that can be passed to
// ADCIntEnable(), ADCIntDisable()
```

and ADCIntClear() as ulIntFlags, and returned from ADCIntStatus()

//******************************************
***********************************
#define ADC_DMA_DONE 0x00000010
#define ADC_FIFO_OVERFLOW 0x00000008
#define ADC_FIFO_UNDERFLOW 0x00000004
#define ADC_FIFO_EMPTY 0x00000002
#define ADC_FIFO_FULL 0x00000001

//******************************************
***********************************
// API Function prototypes
//******************************************
extern void ADCEnable(unsigned long ulBase);
extern void ADCDisable(unsigned long ulBase);
extern void ADCChannelEnable(unsigned long ulBase, unsigned long ulChannel);
extern void ADCChannelDisable(unsigned long ulBase, unsigned long ulChannel);
extern void ADCIntRegister(unsigned long ulBase, unsigned long ulChannel,
    void (*pfnHandler)(void));
extern void ADCIntUnregister(unsigned long ulBase, unsigned long ulChannel);
extern void ADCIntEnable(unsigned long ulBase, unsigned long ulChannel,
    unsigned long ulIntFlags);
extern void ADCIntDisable(unsigned long ulBase, unsigned long ulChannel,
unsigned long ulIntFlags);

extern unsigned long ADCIntStatus(unsigned long ulBase,unsigned long ulChannel);

extern void ADCIntClear(unsigned long ulBase, unsigned long ulChannel,
unsigned long ulIntFlags);

extern void ADCDMAEnable(unsigned long ulBase, unsigned long ulChannel);

extern void ADCDMADisable(unsigned long ulBase, unsigned long ulChannel);

extern void ADCTimerConfig(unsigned long ulBase, unsigned long ulValue);

extern void ADCTimerEnable(unsigned long ulBase);

extern void ADCTimerDisable(unsigned long ulBase);

extern void ADCTimerReset(unsigned long ulBase);

extern unsigned long ADCTimerValueGet(unsigned long ulBase);

extern unsigned char ADCFIFOLvlGet(unsigned long ulBase,
unsigned long ulChannel);

extern unsigned long ADCFIFORead(unsigned long ulBase,
unsigned long ulChannel);

//******************************************
//	Mark	the	end	of	the	C	bindings	section
for	C++	compilers.
//******************************************
#ifdef __cplusplus
}
#endif

#endif // __ADC_H__
1 //******************************************************************************
2 ********************************************
3 //
4 // aes.h
5 //
6 // Defines and Macros for the AES module.
7 //
8 // Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
9 //
10 // Redistribution and use in source and binary forms, with or without
11 // modification, are permitted provided that the following conditions
12 // are met:
13 //
14 // Redistributions of source code must retain the above copyright
15 // notice, this list of conditions and the following disclaimer.
16 //
17 // Redistributions in binary form must
reproduce the above copyright

// notice, this list of conditions and
the following disclaimer in the
documentation and/or other materials
provided with the
distribution.

// Neither the name of Texas Instruments
Incorporated nor the names of
its contributors may be used to
endorse or promote products derived
from this software without specific
prior written permission.

// THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

//
/* ******************************************************************************
 * __DRIVERLIB_AES_H__
 */

#ifndef __DRIVERLIB_AES_H__
#define __DRIVERLIB_AES_H__

/* ******************************************************************************
 *
*/

// If building with a C++ compiler, make all of the definitions in this header have a C binding.

#ifdef __cplusplus
extern "C"
{
#endif

/* ******************************************************************************
 *
*/

// The following defines are used to specify the operation direction in the ui32Config argument in the AESConfig function. Only one is permitted.

#define AES_CFG_DIR_ENCRYPT 0x00000004
#define AES_CFG_DIR_DECRYPT 0x00000000

/* ******************************************************************************
 *
*/

// The following defines are used to specify
the key size in the ui32Config

// argument in the AESConfig function. Only one is permitted.

//

//******************************************************************************
******************************************************************************

#define AES_CFG_KEY_SIZE_128BIT 0x00000008
#define AES_CFG_KEY_SIZE_192BIT 0x00000010
#define AES_CFG_KEY_SIZE_256BIT 0x00000018

//******************************************************************************
******************************************************************************

// The following defines are used to specify the mode of operation in the
// ui32Config argument in the AESConfig function. Only one is permitted.

//******************************************************************************
******************************************************************************

#define AES_CFG_MODE_M 0x2007fe60
#define AES_CFG_MODE_ECB 0x00000000
#define AES_CFG_MODE_CBC 0x00000020
#define AES_CFG_MODE_CTR 0x00000040
#define AES_CFG_MODE_ICM 0x00000020
#define AES_CFG_MODE_CFB 0x00000040
#define AES_CFG_MODE_XTS_TWEAKJL 0x00000080
#define AES_CFG_MODE_XTS_K2IJL 0x00000100
#define AES_CFG_MODE_XTS_K2ILJ0 0x00000180
#define AES_CFG_MODE_F8 0x00000200
#define AES_CFG_MODE_F9 0x20000400
#define AES_CFG_MODE_CBCMAC 0x20000800
#define AES_CFG_MODE_GCM_HLY0ZERO 0x20010040
// The following defines are used to specify the counter width in the
// ui32Config argument in the AESConfig function. It is only required to
// be defined when using CTR, CCM, or GCM modes. Only one length is permitted.

// The following defines are used to define the width of the length field for
// CCM operation through the ui32Config argument in the AESConfig function.
// This value is also known as L. Only one is permitted.
#define AES_CFG_CCM_L_8 0x00380000

// The following defines are used to define the length of the authentication field for CCM operations through the ui32Config argument in the AESConfig function. This value is also known as M. Only one is permitted.

#define AES_CFG_CCM_M_4 0x00400000
#define AES_CFG_CCM_M_6 0x00800000
#define AES_CFG_CCM_M_8 0x00c00000
#define AES_CFG_CCM_M_10 0x01000000
#define AES_CFG_CCM_M_12 0x01400000
#define AES_CFG_CCM_M_14 0x01800000
#define AES_CFG_CCM_M_16 0x01c00000

// Interrupt flags for use with the AESIntEnable, AESIntDisable, and AESIntStatus functions.

#define AES_INT_CONTEXT_IN 0x00000001
#define AES_INT_CONTEXT_OUT 0x00000008
#define AES_INT_DATA_IN 0x00000002
#define AES_INT_DATA_OUT 0x00000004
#define AES_INT_DMA_CONTEXT_IN 0x00010000
#define AES_INT_DMA_CONTEXT_OUT 0x00020000
#define AES_INT_DMA_DATA_IN 0x00040000
#define AES_INT_DMA_DATA_OUT 0x00080000

// Defines used when enabling and disabling DMA requests in the
// AESEnableDMA and AESDisableDMA functions.

#define AES_DMA_DATA_IN 0x00000040
#define AES_DMA_DATA_OUT 0x00000020
#define AES_DMA_CONTEXT_IN 0x00000080
#define AES_DMA_CONTEXT_OUT 0x000000100

extern void AESConfigSet(uint32_t ui32Base, uint32_t ui32Config);
extern void AESKey1Set(uint32_t ui32Base, uint8_t *pui8Key, uint32_t ui32Keysize);
extern void AESKey2Set(uint32_t ui32Base, uint8_t *pui8Key, uint32_t ui32Keysize);
extern void AESKey3Set(uint32_t ui32Base, uint8_t *pui8Key);
extern void AESIVSet(uint32_t ui32Base,
extern void AESIVGet(uint32_t ui32Base, uint8_t *pui8IVdata);
extern void AESTagRead(uint32_t ui32Base, uint8_t *pui8TagData);
extern void AESDataLengthSet(uint32_t ui32Base, uint64_t ui64Length);
extern void AESAuthDataLengthSet(uint32_t ui32Base, uint32_t ui32Length);
extern bool AESDataReadNonBlocking(uint32_t ui32Base, uint8_t *pui8Dest, uint8_t ui8Length);
extern void AESDataRead(uint32_t ui32Base, uint8_t *pui8Dest, uint8_t ui8Length);
extern bool AESDataWriteNonBlocking(uint32_t ui32Base, uint8_t *pui8Src, uint8_t ui8Length);
extern void AESDataWrite(uint32_t ui32Base, uint8_t *pui8Src, uint8_t ui8Length);
extern bool AESDataProcess(uint32_t ui32Base, uint8_t *pui8Src, uint8_t *pui8Dest, uint32_t ui32Length);
extern bool AESDataProcessAE(uint32_t ui32Base, uint8_t *pui8Src, uint8_t *pui8Dest, uint32_t ui32Length, uint8_t *pui8Tag);
extern bool AESDataMAC(uint32_t ui32Base, uint8_t *pui8Src, uint32_t ui32Length, uint8_t *pui8Tag);
*pui8AuthSrc, uint32_t ui32AuthLength,
  uint8_t *pui8Tag);

extern uint32_t AESIntStatus(uint32_t ui32Base, bool bMasked);
extern void AESIntEnable(uint32_t ui32Base, uint32_t ui32IntFlags);
extern void AESIntDisable(uint32_t ui32Base, uint32_t ui32IntFlags);
extern void AESIntClear(uint32_t ui32Base, uint32_t ui32IntFlags);
extern void AESIntRegister(uint32_t ui32Base, void(*pfnHandler)(void));
extern void AESIntUnregister(uint32_t ui32Base);

extern void AESDMAEnable(uint32_t ui32Base, uint32_t ui32Flags);
extern void AESDMADisable(uint32_t ui32Base, uint32_t ui32Flags);

//******************************************************************************
//
// Mark the end of the C bindings section for C++ compilers.
//******************************************************************************

#endif __cplusplus
}
#endif // __DRIVERLIB_AES_H__

#define __DRIVERLIB_AES_H__
//******************************************
//camera.h
/* Prototypes and macros for the camera controller module. */
//Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//Redistributions of source code must retain the above copyright
//notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//******************************************
//******************************************

#ifndef __CAMERA_H__
#define __CAMERA_H__

//******************************************
//******************************************

//
// If building with a C++ compiler, make all of the definitions in this header have a C binding.
//
//******************************************
//******************************************

#ifdef __cplusplus
extern "C"
{
#endif

//******************************************
//******************************************

// Macro defining Camera buffer address
//******************************************

#define CAM_BUFFER_ADDR 0x44018100

//******************************************
//******************************************

// Value that can be passed to CameraXClkSet().
//******************************************

#define CAM_XCLK_STABLE_LO 0x00
#define CAM_XCLK_STABLE_HI 0x01
#define CAM_XCLK_DIV_BYPASS 0x02

//***************************************************************
// Value that can be passed to CameraIntEnable(), CameraIntDisable,
// CameraIntClear() or returned from CameraIntStatus().
//***************************************************************
#define CAM_INT_DMA 0x80000000
#define CAM_INT_FE 0x00010000
#define CAM_INT_FIFO_NOEMPTY 0x00000010
#define CAM_INT_FIFO_FULL 0x00000008
#define CAM_INT_FIFO_THR 0x00000004
#define CAM_INT_FIFO_OF 0x00000002
#define CAN_INT_FIFO_UR 0x00000001

//***************************************************************
// Value that can be passed to CameraXClkConfig().
//***************************************************************
#define CAM_HS_POL_HI 0x00000000
#define CAM_HS_POL_LO 0x00000200
#define CAM_VS_POL_HI 0x00000000
#define CAM_VS_POL_LO 0x00000100
#define CAM_PCLK_RISE_EDGE 0x00000000
#define CAM_PCLK_FALL_EDGE 0x00000400
#define CAM_ORDERCAM_SWAP 0x00000800
#define CAM_NOBT_SYNCHRO 0x00002000
#define CAM_IF_SYNCHRO 0x00080000
# API Function prototypes

```c
extern void CameraReset(unsigned long ulBase);
extern void CameraParamsConfig(unsigned long ulBase, unsigned long ulHSPol,
                                 unsigned long ulVSPol, unsigned long ulFlags);
extern void CameraXClkConfig(unsigned long ulBase, unsigned long ulCamClkIn,
                               unsigned long ulXClk);
extern void CameraXClkSet(unsigned long ulBase, unsigned char bXClkFlags);
extern void CameraDMAEnable(unsigned long ulBase);
extern void CameraDMADisable(unsigned long ulBase);
extern void CameraThresholdSet(unsigned long ulBase, unsigned long ulThreshold);
extern void CameraIntRegister(unsigned long ulBase, void (*pfnHandler)(void));
extern void CameraIntUnregister(unsigned long ulBase);
extern void CameraIntEnable(unsigned long ulBase, unsigned long ulIntFlags);
extern void CameraIntDisable(unsigned long ulBase, unsigned long ulIntFlags);
extern unsigned long CameraIntStatus(unsigned long ulBase);
```
extern void CameraIntClear(unsigned long ulBase, unsigned long ulIntFlags);
extern void CameraCaptureStop(unsigned long ulBase, tBoolean bImmediate);
extern void CameraCaptureStart(unsigned long ulBase);
extern void CameraBufferRead(unsigned long ulBase, unsigned long *pBuffer,
                                unsigned char ucSize);

//******************************************
//  Mark the end of the C bindings section for C++ compilers.
//******************************************
#ifdef __cplusplus
#endif
#endif //__CAMERA_H__

//******************************************
//cpu.h
//Prototypes for the CPU instruction wrapper functions.
//Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
//Redistribution and use in source and binary forms, with or without
//modification, are permitted provided that the following conditions
//are met:
//Redistributions of source code must retain the above copyright
//notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
//******************************************************************************
 ******************************************************************************

#ifndef __CPU_H__
#define __CPU_H__

//******************************************************************************
******************************************************************************

// If building with a C++ compiler, make all of the definitions in this header have a C binding.

//******************************************************************************
******************************************************************************

#ifdef __cplusplus
extern "C"
{
#endif

//******************************************************************************
******************************************************************************

// Prototypes.

//******************************************************************************
******************************************************************************

extern unsigned long CPUcpsid(void);
extern unsigned long CPUcpsie(void);
extern unsigned long CPUprimask(void);
extern void CPUwfi(void);
extern unsigned long CPUbasepriGet(void);
extern void CPUbasepriSet(unsigned long ulNewBasepri);

//******************************************************************************
******************************************************************************
// Mark the end of the C bindings section for C++ compilers.

#ifndef __cplusplus
}
#endif
#endif // __CPU_H__

#define __cplusplus

#endif // __CPU_H__
**cc.h**

---

```c
/* ********************************************************************
******************************************************************** */

//
// crc.h
//
// Defines and Macros for CRC module.
//
// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//
// Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//
// Redistributions in binary form must
```
reproduce the above copyright

notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
/******************************/
#define __DRIVERLIB_CRC_H__
/******************************/

// If building with a C++ compiler, make all of the definitions in this header have a C binding.

#ifdef __cplusplus
extern "C" {
#endif

// The following defines are used in the ui32Config argument of the ECConfig function.

#define CRC_CFG_INIT_SEED        0x00000000  // Initialize with seed
#define CRC_CFG_INIT_0           0x00004000  // Initialize to all '0s'
#define CRC_CFG_INIT_1           0x00006000  // Initialize to all '1s'
#define CRC_CFG_SIZE_8BIT        0x00001000  // Input Data Size
#define CRC_CFG_SIZE_32BIT 0x00000000
  // Input Data Size
#define CRC_CFG_RESINV 0x00000200
  // Result Inverse Enable
#define CRC_CFG_OBR 0x00000100
  // Output Reverse Enable
#define CRC_CFG_IBR 0x00000080
  // Bit reverse enable
#define CRC_CFG_ENDIAN_SBHW 0x00000000
  // Swap byte in half-word
#define CRC_CFG_ENDIAN_SHW 0x00000010
  // Swap half-word
#define CRC_CFG_TYPE_P8005 0x00000000
  // Polynomial 0x8005
#define CRC_CFG_TYPE_P1021 0x00000001
  // Polynomial 0x1021
#define CRC_CFG_TYPE_P4C11DB7 0x00000002
  // Polynomial 0x4C11DB7
#define CRC_CFG_TYPE_P1EDC6F41 0x00000003
  // Polynomial 0x1EDC6F41
#define CRC_CFG_TYPE_TCPCHKSUM 0x00000008
  // TCP checksum

//******************************************
// Function prototypes.
//******************************************
extern void CRCConfigSet(uint32_t ui32Base, uint32_t ui32CRCConfig);
extern uint32_t CRCDataProcess(uint32_t ui32Base, void *puiDataIn,
    uint32_t ui32DataLength, uint32_t ui32Config);
extern void CRCDataWrite(uint32_t ui32Base,
#ifndef __cplusplus
    }
#endif // __cplusplus

#endif // __DRIVERLIB_CRC_H__
//****************************************************************************
************ Macros for assisting debug of the driver library. ****************
// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
// Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//
Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//******************************************
/**                                    */
#elseif __DEBUG_H__
#define __DEBUG_H__

//******************************************
/**                                    */

// Prototype for the function that is called when an invalid argument is passed
// to an API. This is only used when doing a DEBUG build.

//******************************************
/**                                    */

extern void __error__(char *pcFilename,
unsigned long ulLine);

//******************************************
/**                                    */

// The ASSERT macro, which does the actual assertion checking. Typically, this
// will be for procedure arguments.

//******************************************
/**                                    */

#ifndef DEBUG
#define ASSERT(expr)
if(!(expr))
{
__error__(__FILE__, __LINE__);
}
#endif DEBUG
#define ASSERT(expr)
61 

62
63 #else
64 #define ASSERT(expr)
65 #endif
66
67 #endif // __DEBUG_H__
1 //***************************************************************************************
2 //
3 // des.h
4 //
5 // Defines and Macros for the DES module.
6 //
7 // Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
8 //
9 //
10 // Redistribution and use in source and binary forms, with or without
11 // modification, are permitted provided that the following conditions
12 // are met:
13 //
14 // Redistributions of source code must retain the above copyright
15 // notice, this list of conditions and the following disclaimer.
16 //
17 // Redistributions in binary form must
reproduce the above copyright

// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the
// distribution.

// Neither the name of Texas Instruments Incorporated nor the names of
// its contributors may be used to endorse or promote products derived
// from this software without specific prior written permission.

// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
// OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
// SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
// DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

//
```c
#ifndef __DRIVERLIB_DES_H__
#define __DRIVERLIB_DES_H__

//******************************************
***********************************

//
// If building with a C++ compiler, make all
// of the definitions in this header
// have a C binding.
//
//******************************************
***********************************

#ifdef __cplusplus
extern "C"
{
#endif

//******************************************
***********************************

//
// The following defines are used to specify
// the direction with the
// ui32Config argument in the DESConfig() function. Only one is permitted.
//
//******************************************
***********************************

#define DES_CFG_DIR_DECRYPT 0x00000000
#define DES_CFG_DIR_ENCRYPT 0x00000004

//******************************************
***********************************

//
// The following defines are used to specify
```
the operational with the
// ui32Config argument in the DESConfig() function. Only one is permitted.

//******************************************************************************
//******************************************************************************
#define DES_CFG_MODE_ECB 0x00000000
#define DES_CFG_MODE_CBC 0x00000010
#define DES_CFG_MODE_CFB 0x00000020

//******************************************************************************
//******************************************************************************
// The following defines are used to select between single DES and triple DES
// with the ui32Config argument in the DESConfig() function. Only one is permitted.

//******************************************************************************
//******************************************************************************
#define DES_CFG_SINGLE 0x00000000
#define DES_CFG_TRIPLE 0x00000008

//******************************************************************************
//******************************************************************************
// The following defines are used with the DESIntEnable(), DESIntDisable() and
// DESIntStatus() functions.

//******************************************************************************
#define DES_INT_CONTEXT_IN 0x00000001
#define DES_INT_DATA_IN 0x00000002
#define DES_INT_DATA_OUT 0x00000004
#define DES_INT_DMA_CONTEXT_IN 0x00010000
#define DES_INT_DMA_DATA_IN       0x00020000
#define DES_INT_DMA_DATA_OUT      0x00040000

// The following defines are used with the DESEnableDMA() and DESDisableDMA() functions.

#define DES_DMA_CONTEXT_IN        0x00000080
#define DES_DMA_DATA_OUT           0x00000040
#define DES_DMA_DATA_IN            0x00000020
.extern void DESConfigSet(uint32_t ui32Base, uint32_t ui32Config);
.extern void DESSDataRead(uint32_t ui32Base, uint8_t *pui8Dest,
   uint8_t ui8Length);
.extern bool DESSDataReadNonBlocking(uint32_t ui32Base, uint8_t *pui8Dest,
   uint8_t ui8Length);
.extern bool DESSDataProcess(uint32_t ui32Base, uint8_t *pui8Src,
   uint8_t *pui8Dest, uint32_t ui32Length);
.extern void DESSDataWrite(uint32_t ui32Base,
extern bool DESDataWriteNonBlocking(uint32_t ui32Base, uint8_t *pui8Src, uint8_t ui8Length);

extern void DESDMADisable(uint32_t ui32Base, uint32_t ui32Flags);

extern void DESDMAEnable(uint32_t ui32Base, uint32_t ui32Flags);

extern void DESIntClear(uint32_t ui32Base, uint32_t ui32IntFlags);

extern void DESIntDisable(uint32_t ui32Base, uint32_t ui32IntFlags);

extern void DESIntEnable(uint32_t ui32Base, uint32_t ui32IntFlags);

extern void DESIntRegister(uint32_t ui32Base, void (*pfnHandler)(void));

extern uint32_t DESIntStatus(uint32_t ui32Base, bool bMasked);

extern void DESIntUnregister(uint32_t ui32Base);

extern bool DESIVSet(uint32_t ui32Base, uint8_t *pui8IVdata);

extern void DESKeySet(uint32_t ui32Base, uint8_t *pui8Key);

extern void DESDataLengthSet(uint32_t ui32Base, uint32_t ui32Length);

//******************************************
// Mark the end of the C bindings section
// for C++ compilers.
//******************************************
```c
#ifdef __cplusplus
} // __cplusplus
#endif
#endif // __DRIVERLIB_DES_H__
```
```c
//******************************************
***********************************
//
// flash.h
// Prototypes for the flash driver.
//
// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//
// Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//
// Redistributions in binary form must
```
reproduce the above copyright

// notice, this list of conditions and
the following disclaimer in the
documentation and/or other materials
provided with the
distribution.

// Neither the name of Texas Instruments
Incorporated nor the names of
its contributors may be used to
endorse or promote products derived
from this software without specific
prior written permission.

// THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
-INCLUDING NEGLIGENCE OR OTHERWISE)
 ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

//
// **************************************************************************
// ** FLASH_H **
// **************************************************************************

#ifndef __FLASH_H__
#define __FLASH_H__

// **************************************************************************
// **
// ** If building with a C++ compiler, make all of the definitions in this header
// ** have a C binding.
// //
// **************************************************************************

#ifdef __cplusplus
extern "C"
{
#endif

// **************************************************************************
// **
// ** Values that can be passed to FlashProtectSet(), and returned by FlashProtectGet().
// //
// **************************************************************************

typedef enum
{
    FlashReadWrite,      // Flash can be read and written
    FlashReadOnly,       // Flash can only be read
    FlashExecuteOnly    // Flash can only be executed
};

#endif

}
tFlashProtection;

//******************************************  
//
//	Values passed to FlashIntEnable(),  
// FlashIntDisable() and FlashIntClear() and  
// returned from FlashIntStatus().
//
//******************************************  

#define FLASH_INT_PROGRAM	0x00000002	// Programming Interrupt Mask
#define FLASH_INT_ACCESS	0x00000001	// Access Interrupt Mask
#define FLASH_INT_EEPROM	0x00000004	// EEPROM Interrupt Mask
#define FLASH_INT_VOLTAGE_ERR	0x00000200	// Voltage Error Interrupt Mask
#define FLASH_INT_DATA_ERR	0x00000400	// Invalid Data Interrupt Mask
#define FLASH_INT_ERASE_ERR	0x00000800	// Erase Error Interrupt Mask
#define FLASH_INT_PROGRAM_ERR	0x00002000	// Program Verify Error Interrupt Mask

//******************************************  
// Prototypes for the APIs.
//
//******************************************  

extern void FlashDisable(void);
extern long FlashErase(unsigned long
ulAddress);

extern void FlashEraseNonBlocking(unsigned long ulAddress);

extern long FlashMassErase(void);

extern void FlashMassEraseNonBlocking(void);

extern long FlashProgram(unsigned long *pulData, unsigned long ulAddress,
                             unsigned long ulAddress,
                             unsigned long ulCount);

extern long FlashProgramNonBlocking(unsigned long *pulData,
                                      unsigned long ulAddress,
                                      unsigned long ulAddress,
                                      unsigned long ulCount);

extern void FlashIntRegister(void (*pfnHandler)(void));

extern void FlashIntUnregister(void);

extern void FlashIntEnable(unsigned long ulIntFlags);

extern void FlashIntDisable(unsigned long ulIntFlags);

extern unsigned long FlashIntStatus(tBoolean bMasked);

extern void FlashIntClear(unsigned long ulIntFlags);

extern tFlashProtection FlashProtectGet(unsigned long ulAddress);

//********************************************************************************

// Mark the end of the C bindings section for C++ compilers.

//********************************************************************************
#ifndef __cplusplus
}
#endif
#endif // __FLASH_H__

//******************************************
***********************************
//
// gpio.h
// Defines and Macros for GPIO API.
//
// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//
// Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//
// Redistributions in binary form must
reproduce the above copyright

// notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the
distribution.

// Neither the name of Texas Instruments Incorporated nor the names of
its contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
/******************************************
***********************************
 ifndef __GPIO_H__
#define __GPIO_H__

/******************************************
***********************************
 //
 // If building with a C++ compiler, make all
 // of the definitions in this header
 // have a C binding.
 //
 //******************************************
***********************************

#ifndef __cplusplus
extern "C"
{
#endif

/******************************************
***********************************
 // The following values define the bit field
 for the ucPins argument to several
 // of the APIs.
 //
 //******************************************
***********************************
#define GPIO_PIN_0  0x00000001
 // GPIO pin 0
#define GPIO_PIN_1  0x00000002
 // GPIO pin 1
#define GPIO_PIN_2  0x00000004
 // GPIO pin 2
#define GPIO_PIN_3  0x00000008
 // GPIO pin 3
```c
// GPIO pin 4
#define GPIO_PIN_4 0x00000010

// GPIO pin 5
#define GPIO_PIN_5 0x00000020

// GPIO pin 6
#define GPIO_PIN_6 0x00000040

// GPIO pin 7
#define GPIO_PIN_7 0x00000080

// Values that can be passed to
// GPIODirModeSet as the ulPinIO parameter, and
// returned from GPIODirModeGet.

// Pin is a GPIO input
#define GPIO_DIR_MODE_IN 0x00000000

// Pin is a GPIO output
#define GPIO_DIR_MODE_OUT 0x00000001

// Values that can be passed to
// GPIOIntTypeSet as the ulIntType parameter, and
// returned from GPIOIntTypeGet.

// Interrupt on falling edge
#define GPIO_FALLING_EDGE 0x00000000

// Interrupt on rising edge
#define GPIO_RISING_EDGE 0x00000004

// Interrupt on both edges
#define GPIO_BOTH_EDGES 0x00000001
```
// Interrupt on both edges
#define GPIO_LOW_LEVEL 0x00000002
// Interrupt on low level
#define GPIO_HIGH_LEVEL 0x00000006
// Interrupt on high level

// Values that can be passed to GPIOIntEnable() and GPIOIntDisable() functions in the ulIntFlags parameter.

#define GPIO_INT_DMA 0x00000100
#define GPIO_INT_PIN_0 0x00000001
#define GPIO_INT_PIN_1 0x00000002
#define GPIO_INT_PIN_2 0x00000004
#define GPIO_INT_PIN_3 0x00000008
#define GPIO_INT_PIN_4 0x00000010
#define GPIO_INT_PIN_5 0x00000020
#define GPIO_INT_PIN_6 0x00000040
#define GPIO_INT_PIN_7 0x00000080

// Prototypes for the APIs.
extern void GPIODirModeSet(unsigned long ulPort, unsigned char ucPins, unsigned long ulPinIO);
extern unsigned long GPIODirModeGet(unsigned long ulPort, unsigned char ucPins, unsigned long ulPinIO);
extern void GPIOIntTypeSet(unsigned long ulPort, unsigned char ucPins, unsigned long ulIntType);
extern void GPIODMATriggerEnable(unsigned long ulPort);
extern void GPIODMATriggerDisable(unsigned long ulPort);
extern unsigned long GPIOIntTypeGet(unsigned long ulPort, unsigned char ucPin);
extern void GPIOIntEnable(unsigned long ulPort, unsigned long ulIntFlags);
extern void GPIOIntDisable(unsigned long ulPort, unsigned long ulIntFlags);
extern long GPIOIntStatus(unsigned long ulPort, tBoolean bMasked);
extern void GPIOIntClear(unsigned long ulPort, unsigned long ulIntFlags);
extern void GPIOIntRegister(unsigned long ulPort,
                              void (*pfnIntHandler)(void));
extern void GPIOIntUnregister(unsigned long ulPort);
extern long GPIOPinRead(unsigned long ulPort, unsigned char ucPins);
extern void GPIOPinWrite(unsigned long ulPort, unsigned char ucPins, unsigned char ucVal);

// Mark the end of the C bindings section for C++ compilers.
//******************************************************
***********************************
#endif __cplusplus
}
#endif // __GPIO_H__

// ***********************************
// hwspinlock.h
// Prototypes for the Apps-NWP spinlock.
// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions are met:
// Redistributions of source code must retain the above copyright notice, this list of conditions and
// the following disclaimer.
// Redistributions in binary form must
reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//******************************************
#ifndef __HWSPINLOCK_H__
define __HWSPINLOCK_H__

//******************************************
// If building with a C++ compiler, make all
of the definitions in this header
// have a C binding.
//
//******************************************
#ifdef __cplusplus
extern "C"
{
#endif

//******************************************
// values that can be passed to API as
ui32LockID parameter
//******************************************
#define HWSPINLOCK_SSPI						0x02000000

// Values that are returned from
HwSpinLockTest()
//******************************************
#define HWSPINLOCK_OWNER_APPS	0x00000001
#define HWSPINLOCK_OWNER_NWP		0x00000002
#define HWSPINLOCK_OWNER_NONE	0x00000000
// API Function prototypes

extern void HwSpinLockAcquire(uint32_t ui32LockID);
extern int32_t HwSpinLockTryAcquire(uint32_t ui32LockID, uint32_t ui32Retry);
extern void HwSpinLockRelease(uint32_t ui32LockID);
extern uint32_t HwSpinLockTest(uint32_t ui32LockID, bool bCurrentStatus);

// Mark the end of the C bindings section for C++ compilers.

#ifndef __cplusplus
#endif // __HWSPINLOCK_H__

// i2c.h

// Prototypes for the I2C Driver.

// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:

// Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.

// Redistributions in binary form must
reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//******************************************
//***********************************
#ifndef __DRIVERLIB_I2C_H__
define __DRIVERLIB_I2C_H__

//******************************************
//***********************************
//
// If building with a C++ compiler, make all of the definitions in this header have a C binding.
//
//******************************************
//***********************************
#ifndef __cplusplus
extern "C"
{
#endif

//******************************************
//***********************************
// Defines for the API.
//
//******************************************
//***********************************
#ifndef __cplusplus

#define I2C_INT_MASTER 0x00000001
#define I2C_INT_SLAVE 0x00000002

//******************************************
//***********************************
// Interrupt defines.
//
//******************************************
//***********************************
#define I2C_INT_MASTER 0x00000001
#define I2C_INT_SLAVE 0x00000002
#define I2C_MASTER_CMD_SINGLE_SEND 0x00000007
#define I2C_MASTER_CMD_SINGLE_RECEIVE 0x00000007
#define I2C_MASTER_CMD_BURST_SEND_START 0x00000003
#define I2C_MASTER_CMD_BURST_SEND_CONT 0x00000001
#define I2C_MASTER_CMD_BURST_SEND_FINISH 0x00000005
#define I2C_MASTER_CMD_BURST_SEND_STOP 0x00000004
#define I2C_MASTER_CMD_BURST_SEND_ERROR_STOP 0x00000004
#define I2C_MASTER_CMD_BURST_RECEIVE_START 0x0000000b
#define I2C_MASTER_CMD_BURST_RECEIVE_CONT 0x00000009
#define I2C_MASTER_CMD_BURST_RECEIVE_FINISH 0x00000009
#define I2C_MASTER_CMD_BURST_RECEIVE_ERROR_STOP 0x00000005
#define I2C_MASTER_CMD_QUICK_COMMAND 0x00000004
#define I2C_MASTER_CMD_HS_MASTER_CODE_SEND 0x00000027
#define I2C_MASTER_CMD_FIFO_SINGLE_SEND 0x00000013
#define I2C_MASTER_CMD_FIFO_SINGLE_RECEIVE 0x00000046
#define I2C_MASTER_CMD_FIFO_BURST_SEND_START 0x00000042
#define I2C_MASTER_CMD_FIFO_BURST_SEND_CONT 0x00000040
#define I2C_MASTER_CMD_FIFO_BURST_SEND_FINISH 0x00000044
#define I2C_MASTER_CMD_FIFO_BURST_SEND_ERROR_STOP 0x00000004
#define I2C_MASTER_CMD_FIFO_BURST_RECEIVE_START 0x0000004a
#define I2C_MASTER_CMD_FIFO_BURST_RECEIVE_CONT 0x0000004a


114 #define I2C_MASTER_CMD_FIFO_BURST_RECEIVE_FINISH
115 0x00000048

116 #define I2C_MASTER_CMD_FIFO_BURST_RECEIVE_ERROR_STOP
117 0x00000044

118 //******************************************
119 //
120 //
121 //
122 // I2C Master glitch filter configuration.
123 //
124 //******************************************

125 #define I2C_MASTER_GLITCH_FILTER_DISABLED
126 0

127 #define I2C_MASTER_GLITCH_FILTER_1
128 0x00010000

129 #define I2C_MASTER_GLITCH_FILTER_2
130 0x00020000

131 #define I2C_MASTER_GLITCH_FILTER_3
132 0x00030000

133 #define I2C_MASTER_GLITCH_FILTER_4
134 0x00040000

135 #define I2C_MASTER_GLITCH_FILTER_8
136 0x00050000

137 #define I2C_MASTER_GLITCH_FILTER_16
```
#define I2C_MASTER_GLITCH_FILTER_32 0x00060000

#define I2C_MASTER_ERR_NONE 0
#define I2C_MASTER_ERR_ADDR_ACK 0x00000004
#define I2C_MASTER_ERR_DATA_ACK 0x00000008
#define I2C_MASTER_ERR_ARB_LOST 0x00000010
#define I2C_MASTER_ERR_CLK_TOUT 0x00000080

#define I2C_SLAVE_ACT_NONE 0
#define I2C_SLAVE_ACT_RREQ 0x00000001
// Master has sent data
#define I2C_SLAVE_ACT_TREQ 0x00000002
// Master has requested data
#define I2C_SLAVE_ACT_RREQ_FBR 0x00000005
// Master has sent first byte
#define I2C_SLAVE_ACT_OWN2SEL 0x00000008
// Master requested secondary slave
#define I2C_SLAVE_ACT_QCMD 0x00000010
// Master has sent a Quick Command
```
#define I2C_SLAVE_ACT_QCMD_DATA 0x00000020
// Master Quick Command value

//**************************************************************
// Miscellaneous I2C driver definitions.
//**************************************************************

#define I2C_MASTER_MAX_RETRIES 1000
// Number of retries

//**************************************************************
// I2C Master interrupts.
//**************************************************************

#define I2C_MASTER_INT_RX_FIFO_FULL 0x00000800
// RX FIFO Full Interrupt
#define I2C_MASTER_INT_TX_FIFO_EMPTY 0x00000400
// TX FIFO Empty Interrupt
#define I2C_MASTER_INT_RX_FIFO_REQ 0x00000200
// RX FIFO Request Interrupt
#define I2C_MASTER_INT_TX_FIFO_REQ 0x00000100
// TX FIFO Request Interrupt
#define I2C_MASTER_INT_ARB_LOST
// Arb Lost Interrupt
#define I2C_MASTER_INT_STOP 0x00000040
// Stop Condition Interrupt
#define I2C_MASTER_INT_START 0x00000020
// Start Condition Interrupt
#define I2C_MASTER_INT_NACK 0x00000010
// Addr/Data NACK Interrupt
#define I2C_MASTER_INT_TX_DMA_DONE 0x00000008
// TX DMA Complete Interrupt
#define I2C_MASTER_INT_RX_DMA_DONE 0x00000004
// RX DMA Complete Interrupt
#define I2C_MASTER_INT_TIMEOUT 0x00000002
// Clock Timeout Interrupt
#define I2C_MASTER_INT_DATA 0x00000001
// Data Interrupt

//**********************************************************
//**********************************************************
// I2C Slave interrupts.
//**********************************************************
//**********************************************************
#define I2C_SLAVE_INT_RX_FIFO_FULL 0x00000100
// RX FIFO Full Interrupt
#define I2C_SLAVE_INT_TX_FIFO_EMPTY 0x00000080
// TX FIFO Empty Interrupt
#define I2C_SLAVE_INT_RX_FIFO_REQ 0x00000040
    // RX FIFO Request Interrupt
#define I2C_SLAVE_INT_TX_FIFO_REQ 0x00000020
    // TX FIFO Request Interrupt
#define I2C_SLAVE_INT_TX_DMA_DONE 0x00000010
    // TX DMA Complete Interrupt
#define I2C_SLAVE_INT_RX_DMA_DONE 0x00000008
    // RX DMA Complete Interrupt
#define I2C_SLAVE_INT_STOP 0x00000004
    // Stop Condition Interrupt
#define I2C_SLAVE_INT_START 0x00000002
    // Start Condition Interrupt
#define I2C_SLAVE_INT_DATA 0x00000001
    // Data Interrupt

/*********************************
I2C Slave FIFO configuration macros.
**********************************/
#define I2C_SLAVE_TX_FIFO_ENABLE 0x00000002
#define I2C_SLAVE_RX_FIFO_ENABLE 0x00000004
I2C FIFO configuration macros.

```c
#define I2C_FIFO_CFG_TX_MASTER 0x00000000
#define I2C_FIFO_CFG_TX_SLAVE 0x00008000
#define I2C_FIFO_CFG_RX_MASTER 0x00000000
#define I2C_FIFO_CFG_RX_SLAVE 0x80000000
#define I2C_FIFO_CFG_TX_MASTER_DMA 0x00002000
#define I2C_FIFO_CFG_TX_SLAVE_DMA 0x0000a000
#define I2C_FIFO_CFG_RX_MASTER_DMA 0x20000000
#define I2C_FIFO_CFG_RX_SLAVE_DMA 0xa0000000
#define I2C_FIFO_CFG_TX_NO_TRIG 0x00000000
#define I2C_FIFO_CFG_TX_TRIG_1 0x00000001
#define I2C_FIFO_CFG_TX_TRIG_2 0x00000002
#define I2C_FIFO_CFG_TX_TRIG_3 0x00000003
#define I2C_FIFO_CFG_TX_TRIG_4 0x00000004
#define I2C_FIFO_CFG_TX_TRIG_5 0x00000005
#define I2C_FIFO_CFG_TX_TRIG_6 0x00000006
#define I2C_FIFO_CFG_TX_TRIG_7 0x00000007
#define I2C_FIFO_CFG_TX_TRIG_8 0x00000008
#define I2C_FIFO_CFG_RX_NO_TRIG 0x00000000
#define I2C_FIFO_CFG_RX_TRIG_1 0x00010000
#define I2C_FIFO_CFG_RX_TRIG_2 0x00020000
#define I2C_FIFO_CFG_RX_TRIG_3 0x00030000
#define I2C_FIFO_CFG_RX_TRIG_4 0x00040000
```
#define I2C_FIFO_CFG_RX_TRIG_5 0x00050000
#define I2C_FIFO_CFG_RX_TRIG_6 0x00060000
#define I2C_FIFO_CFG_RX_TRIG_7 0x00070000
#define I2C_FIFO_CFG_RX_TRIG_8 0x00080000

//******************************************
// I2C FIFO status.
//******************************************
#define I2C_FIFO_RX_BELOW_TRIG_LEVEL 0x00040000
#define I2C_FIFO_RX_FULL 0x00020000
#define I2C_FIFO_RX_EMPTY 0x00010000
#define I2C_FIFO_TX_BELOW_TRIG_LEVEL 0x00000004
#define I2C_FIFO_TX_FULL 0x00000002
#define I2C_FIFO_TX_EMPTY 0x00000001

//******************************************
// Prototypes for the APIs.
//******************************************
extern void I2CIntRegister(uint32_t ui32Base, void (pfnHandler)(void));
extern void I2CIntUnregister(uint32_t ui32Base);
extern void I2CTxFIFOConfigSet(uint32_t ui32Base, uint32_t ui32Config);
extern void I2CTxFIFOFlush(uint32_t
extern void I2CRxFIFOConfigSet(uint32_t ui32Base, uint32_t ui32Config);
extern void I2CRxFIFOFlush(uint32_t ui32Base);
extern uint32_t I2CFIFOStatus(uint32_t ui32Base);
extern void I2CFIFODataPut(uint32_t ui32Base, uint8_t ui8Data);
extern uint32_t I2CFIFODataPutNonBlocking(uint32_t ui32Base, uint8_t ui8Data);
extern uint32_t I2CFIFODataGet(uint32_t ui32Base);
extern uint32_t I2CFIFODataGetNonBlocking(uint32_t ui32Base, uint8_t *pui8Data);
extern void I2CMasterBurstLengthSet(uint32_t ui32Base, uint8_t ui8Length);
extern uint32_t I2CMasterBurstCountGet(uint32_t ui32Base);
extern void I2CMasterGlitchFilterConfigSet(uint32_t ui32Base, uint32_t ui32Config);
extern void I2CSlaveFIFOEnable(uint32_t ui32Base, uint32_t ui32Config);
extern void I2CSlaveFIFODisable(uint32_t ui32Base);
extern bool I2CMasterBusBusy(uint32_t ui32Base);
extern bool I2CMasterBusy(uint32_t ui32Base);
extern void I2CMasterControl(uint32_t ui32Base, uint32_t ui32Cmd);

extern uint32_t I2CMasterDataGet(uint32_t ui32Base);

extern void I2CMasterDataPut(uint32_t ui32Base, uint8_t ui8Data);

extern void I2CMasterDisable(uint32_t ui32Base);

extern void I2CMasterEnable(uint32_t ui32Base);

extern uint32_t I2CMasterErr(uint32_t ui32Base);

extern void I2CMasterInitExpClk(uint32_t ui32Base, uint32_t ui32I2CClk, bool bFast);

extern void I2CMasterIntClear(uint32_t ui32Base);

extern void I2CMasterIntDisable(uint32_t ui32Base);

extern void I2CMasterIntEnable(uint32_t ui32Base);

extern bool I2CMasterIntStatus(uint32_t ui32Base, bool bMasked);

extern void I2CMasterIntEnableEx(uint32_t ui32Base, uint32_t ui32IntFlags);

extern void I2CMasterIntDisableEx(uint32_t ui32Base, uint32_t ui32IntFlags);

extern uint32_t I2CMasterIntStatusEx(uint32_t ui32Base, bool bMasked);

extern void I2CMasterIntClearEx(uint32_t ui32Base);
ui32Base,
ui32IntFlags);
extern void I2CMasterTimeoutSet(uint32_t ui32Base, uint32_t ui32Value);
extern void I2CSlaveACKOverride(uint32_t ui32Base, bool bEnable);
extern void I2CSlaveACKValueSet(uint32_t ui32Base, bool bACK);
extern uint32_t I2CMasterLineStateGet(uint32_t ui32Base);
extern void I2CMasterSlaveAddrSet(uint32_t ui32Base,
uint8_t ui8SlaveAddr,
bool bReceive);
extern uint32_t I2CSlaveDataGet(uint32_t ui32Base);
extern void I2CSlaveDataPut(uint32_t ui32Base, uint8_t ui8Data);
extern void I2CSlaveDisable(uint32_t ui32Base);
extern void I2CSlaveEnable(uint32_t ui32Base);
extern void I2CSlaveInit(uint32_t ui32Base,
uint8_t ui8SlaveAddr);
extern void I2CSlaveAddressSet(uint32_t ui32Base, uint8_t ui8AddrNum,
uint8_t ui8SlaveAddr);
extern void I2CSlaveIntClear(uint32_t ui32Base);
extern void I2CSlaveIntDisable(uint32_t ui32Base);
extern void I2CSlaveIntEnable(uint32_t ui32Base);
extern void I2CSlaveIntClearEx(uint32_t ui32Base, uint32_t ui32IntFlags);
extern void I2CSlaveIntDisableEx(uint32_t ui32Base, uint32_t ui32IntFlags);
extern void I2CSlaveIntEnableEx(uint32_t ui32Base, uint32_t ui32IntFlags);
extern bool I2CSlaveIntStatus(uint32_t ui32Base, bool bMasked);
extern uint32_t I2CSlaveIntStatusEx(uint32_t ui32Base, bool bMasked);
extern uint32_t I2CSlaveStatus(uint32_t ui32Base);

// ******************************************
//     Mark the end of the C bindings section
// for C++ compilers.
// ******************************************

#ifdef __cplusplus
}
#endif
#endif // __DRIVERLIB_I2C_H__

//i2s.h

//******************************************
***********************************
//		i2s.h
//		Defines and Macros for the I2S.
//
// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//
// Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//
// Redistributions in binary form must
reproduce the above copyright

// notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the
distribution.

// Neither the name of Texas Instruments Incorporated nor the names of
its contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#ifndef __I2S_H__
#define __I2S_H__

// If building with a C++ compiler, make all of the definitions in this header have a C binding.
#endif

#ifndef __cplusplus
extern "C"
{
#define I2S_TX_DMA_PORT	0x4401E200
#define I2S_RX_DMA_PORT	0x4401E280

// Values that can be passed to I2SCfgSetExpClk() as the ulConfig parameter.

//

}}
//******************************************
******************************************
#define I2S_SLOT_SIZE_8 0x00300032
#define I2S_SLOT_SIZE_16 0x00700074
#define I2S_SLOT_SIZE_24 0x00B000B6

#define I2S_PORT_CPU 0x00080008
#define I2S_PORT_DMA 0x00000000

#define I2S_MODE_MASTER 0x00000000
#define I2S_MODE_SLAVE 0x00008000

//******************************************
******************************************
// Values that can be passed as ulDataLine parameter.
//******************************************
******************************************
#define I2S_DATA_LINE_0 0x00000001
#define I2S_DATA_LINE_1 0x00000002

//******************************************
******************************************
// Values that can be passed to I2SSerializerConfig() as the ulSerMode parameter.
//******************************************
******************************************
#define I2S_SER_MODE_TX 0x00000001
#define I2S_SER_MODE_RX 0x00000002
#define I2S_SER_MODE_DISABLE 0x00000000
Values that can be passed to
I2SSerializerConfig() as the ulInActState parameter.

Values that can be passed to
I2SIntEnable() and I2SIntDisable() as the
ulIntFlags parameter.

#define I2S_INT_XUNDRN 0x00000001
#define I2S_INT_XSYNCERR 0x00000002
#define I2S_INT_XLAST 0x00000010
#define I2S_INT_XDATA 0x00000020
#define I2S_INT_XSTAFRM 0x00000080
#define I2S_INT_XDMA 0x08000000
#define I2S_INT_ROVRN 0x00010000
#define I2S_INT_RSYNCERR 0x00020000
#define I2S_INT_RLAST 0x00100000
#define I2S_INT_RDATA 0x00200000
#define I2S_INT_RSTAFRM 0x00800000
#define I2S_INT_RDMA 0x04000000
// Values that can be passed to I2SRxActiveSlotSet() and I2STxActiveSlotSet

// Values that can be passed to I2SIntClear() as the ulIntFlags parameter and returned from I2SIntStatus().

#define I2S_STS_XERR 0x00000100
#define I2S_STS_XDMAERR 0x00000080
#define I2S_STS_XSTAFRM 0x00000040
#define I2S_STS_XDATA 0x00000020
#define I2S_STS_XLAST 0x00000010
#define I2S_STS_XSYNCERR 0x00000002
#define I2S_STS_XUNDRN 0x00000001
#define I2S_STS_RERR 0x01000000
#define I2S_STS_RDMAERR 0x00800000
#define I2S_STS_RSTAFRM 0x00400000
#define I2S_STS_RDATA 0x00200000
#define I2S_STS_RLAST 0x00100000
#define I2S_STS_RSYNCERR 0x00020000
#define I2S_STS_ROVERN 0x00010000
#define I2S_STS_RDMA 0x40000000
/******************************************

// Values that can be passed to I2SEnable() as the ulMode parameter.

SqlConnection, 

I2S_ENABLE, 

I2S_DISABLE, 

I2S_DATA_PUT, 

I2S_DATA_GET,

I2S_DATA_PUT_NON_BLOCKING,

I2S_DATA_GET_NON_BLOCKING,

I2S_MODE_TX_ONLY, 

I2S_MODE_TX_RX_SYNC,

#define I2S_MODE_TX_ONLY 0x00000001
#define I2S_MODE_TX_RX_SYNC 0x00000003

// API Function prototypes

extern void I2SEnable(unsigned long ulBase, 
                        unsigned long ulMode);
extern void I2SDisable(unsigned long ulBase);

extern void I2SDataPut(unsigned long ulBase, 
                        unsigned long ulDataLine, 
                        unsigned long ulData);

extern long I2SDataPutNonBlocking(unsigned long ulBase, 
                                    unsigned long ulDataLine, 
                                    unsigned long ulData);

extern void I2SDataGet(unsigned long ulBase, 
                        unsigned long ulDataLine, 
                        unsigned long *pulData);

extern long I2SDataGetNonBlocking(unsigned long ulBase, 
                                    unsigned long ulDataLine, 
                                    unsigned long *pulData);
unsigned long ulDataLine, unsigned long *pulData);

extern void I2SConfigSetExpClk(unsigned long ulBase, unsigned long ulI2SClk,
                                  unsigned long ulBitClk, unsigned long ulConfig);

extern void I2STxFIFOEnable(unsigned long ulBase, unsigned long ulTxLevel,
                                unsigned long ulWordsPerTransfer);

extern void I2STxFIFODisable(unsigned long ulBase);

extern void I2SRxFIFOEnable(unsigned long ulBase, unsigned long ulRxLevel,
                               unsigned long ulWordsPerTransfer);

extern void I2SRxFIFODisable(unsigned long ulBase);

extern unsigned long I2STxFIFOStatusGet(unsigned long ulBase);

extern unsigned long I2SRxFIFOStatusGet(unsigned long ulBase);

extern void I2SSerializerConfig(unsigned long ulBase, unsigned long ulDataLine,
                                  unsigned long ulSerMode, unsigned long ulInActState);

extern void I2SIntEnable(unsigned long ulBase, unsigned long ulIntFlags);

extern void I2SIntDisable(unsigned long ulBase, unsigned long ulIntFlags);

extern unsigned long I2SIntStatus(unsigned long ulBase);

extern void I2SIntClear(unsigned long ulBase);
ulBase, unsigned long ulIntFlags);

extern void I2SIntRegister(unsigned long ulBase, void (*pfnHandler)(void));

extern void I2SIntUnregister(unsigned long ulBase);

extern void I2STxActiveSlotSet(unsigned long ulBase, unsigned long ulActSlot);

extern void I2SRxActiveSlotSet(unsigned long ulBase, unsigned long ulActSlot);

//******************************************
//******************************************
//	Mark the end of the C bindings section
// for C++ compilers.
//
//******************************************
//******************************************

#ifdef __cplusplus
}
#endif
#endif //__I2S_H__
interrupt.h

/*--------------------------------*/
*/
*/ interrupt.h
*/
*/ Prototypes for the NVIC Interrupt
*/ Controller Driver.
*/
*/ Copyright (C) 2014 Texas Instruments
*/ Incorporated - http://www.ti.com/
*/
*/ Redistribution and use in source and
*/ binary forms, with or without
*/ modification, are permitted provided
*/ that the following conditions
*/ are met:
*/
*/ Redistributions of source code must
*/ retain the above copyright
*/ notice, this list of conditions and
*/ the following disclaimer.
*/
Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the
distribution.

Neither the name of Texas Instruments Incorporated nor the names of
its contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
/*
 ******************************************
 ******************************************

ifndef __INTERRUPT_H__
deфине __INTERRUPT_H__

/******************************************
 ******************************************

// If building with a C++ compiler, make all of the definitions in this header have a C binding.

/******************************************
 ******************************************

#ifdef __cplusplus
extern "C"
{
#endif

/******************************************
 ******************************************

// A union that describes the entries of the vector table. The union is needed since the first entry is the stack pointer and the remainder are function pointers.

/******************************************
 ******************************************

typedef union {
    void (*pfnHandler)(void);
    unsigned long ulPtr;
}
uVectorEntry;

// Macro to generate an interrupt priority mask based on the number of bits of priority supported by the hardware.

#define INT_PRIORITY_MASK ((0xFF << (8 - NUM_PRIORITY_BITS)) & 0xFF)

// Interrupt priority levels

#define INT_PRIORITY_LVL_0 0x00
#define INT_PRIORITY_LVL_1 0x20
#define INT_PRIORITY_LVL_2 0x40
#define INT_PRIORITY_LVL_3 0x60
#define INT_PRIORITY_LVL_4 0x80
#define INT_PRIORITY_LVL_5 0xA0
#define INT_PRIORITY_LVL_6 0xC0
#define INT_PRIORITY_LVL_7 0xE0

// Prototypes for the APIs.

extern tBoolean IntMasterEnable(void);
extern tBoolean IntMasterDisable(void);
extern void IntVTableBaseSet(unsigned long ulVtableBase);
extern void IntRegister(unsigned long ulInterrupt, void (*pfnHandler)(void));
extern void IntUnregister(unsigned long ulInterrupt);
extern void IntPriorityGroupingSet(unsigned long ulBits);
extern unsigned long IntPriorityGroupingGet(void);
extern void IntRegister(unsigned long ulInterrupt,
unsigned char ucPriority);
extern long IntPriorityGet(unsigned long ulInterrupt);
extern void IntEnable(unsigned long ulInterrupt);
extern void IntDisable(unsigned long ulInterrupt);
extern void IntPendSet(unsigned long ulInterrupt);
extern void IntPendClear(unsigned long ulInterrupt);
extern void IntPriorityMaskSet(unsigned long ulPriorityMask);
extern unsigned long IntPriorityMaskGet(void);

//***********************************************************************
// Mark the end of the C bindings section for C++ compilers.
//***********************************************************************
#ifdef __cplusplus

}  // __cplusplus
#endif

#endif // __INTERRUPT_H__
pin.h

1 //**********************************************************************************
2 ******************************************************************************
3 //
4 //  pin.h
5 //  Defines and Macros for the pin mux module
6 //
7 //  Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
8 //
9 //
10 //  Redistribution and use in source and binary forms, with or without
11 //  modification, are permitted provided that the following conditions
12 //  are met:
13 //
14 //  Redistributions of source code must retain the above copyright
15 //  notice, this list of conditions and the following disclaimer.
16 //
Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
// ******************************************
//
// If building with a C++ compiler, make all of the definitions in this header
// have a C binding.
//
// ******************************************
#ifdef __cplusplus
extern "C"
{
#endif

//Macros Defining Pins
//(***********************************************************************
#define PIN_01 0x00000000
#define PIN_02 0x00000001
#define PIN_03 0x00000002
#define PIN_04 0x00000003
#define PIN_05 0x00000004
#define PIN_06 0x00000005
#define PIN_07 0x00000006
#define PIN_08 0x00000007
#define PIN_09 0x00000008
#define PIN_10 0x00000009
#define PIN_11 0x0000000A
#define PIN_12 0x0000000B
}(***********************************************************************
#endif

#define PIN_13 0x0000000C
#define PIN_14 0x0000000D
#define PIN_15 0x0000000E
#define PIN_16 0x0000000F
#define PIN_17 0x00000010
#define PIN_18 0x00000011
#define PIN_19 0x00000012
#define PIN_20 0x00000013
#define PIN_21 0x00000014
#define PIN_45 0x0000002C
#define PIN_46 0x0000002D
#define PIN_47 0x0000002E
#define PIN_48 0x0000002F
#define PIN_49 0x00000030
#define PIN_50 0x00000031
#define PIN_52 0x00000033
#define PIN_53 0x00000034
#define PIN_55 0x00000036
#define PIN_56 0x00000037
#define PIN_57 0x00000038
#define PIN_58 0x00000039
#define PIN_59 0x0000003A
#define PIN_60 0x0000003B
#define PIN_61 0x0000003C
#define PIN_62 0x0000003D
#define PIN_63 0x0000003E
#define PIN_64 0x0000003F

//******************************************
// Macros that can be used with
// PinConfigSet(), PinTypeGet(), PinStrengthGet()
//******************************************
#define PIN_MODE_0 0x00000000
#define PIN_MODE_1 0x00000001
#define PIN_MODE_2 0x00000002
#define PIN_MODE_3 0x00000003
#define PIN_MODE_4 0x00000004
#define PIN_MODE_5 0x00000005
#define PIN_MODE_6 0x00000006
#define PIN_MODE_7 0x00000007
#define PIN_MODE_8 0x00000008
#define PIN_MODE_9 0x00000009
#define PIN_MODE_10 0x0000000A
#define PIN_MODE_11 0x0000000B
#define PIN_MODE_12 0x0000000C
#define PIN_MODE_13 0x0000000D
#define PIN_MODE_14 0x0000000E
#define PIN_MODE_15 0x0000000F

// Note: PIN_MODE_255 is a dummy define for pinmux utility code generation
// PIN_MODE_255 should never be used in any user code.
#define PIN_MODE_255 0x000000FF

//******************************************
// Macros that can be used with PinDirModeSet() and returned from PinDirModeGet().
//******************************************
#define PIN_DIR_MODE_IN 0x00000C00 // Pin is input
#define PIN_DIR_MODE_OUT 0x00000800 // Pin is output
#define PIN_DIR_MODE_HW 0x00000000 // Pin is peripheral function

//******************************************
// Macros that can be used with PinConfigSet()

***********************************
# define PIN_STRENGTH_2MA 0x00000020
# define PIN_STRENGTH_4MA 0x00000040
# define PIN_STRENGTH_6MA 0x00000060

# define PIN_TYPE_STD 0x00000000
# define PIN_TYPE_STD_PU 0x00000100
# define PIN_TYPE_STD_PD 0x00000200

# define PIN_TYPE_OD 0x00000010
# define PIN_TYPE_OD_PU 0x00000110
# define PIN_TYPE_OD_PD 0x00000210
# define PIN_TYPE_ANALOG 0x10000000

// API Function prototypes

//******************************************
extern void PinModeSet(unsigned long ulPin, unsigned long ulPinMode);
extern void PinDirModeSet(unsigned long ulPin, unsigned long ulPinIO);
extern unsigned long PinDirModeGet(unsigned long ulPin);
extern unsigned long PinModeGet(unsigned long ulPin);
extern void PinConfigGet(unsigned long ulPin, unsigned long *pulPinStrength,
unsigned long *pulPinType);

extern void PinConfigSet(unsigned long ulPin, unsigned long ulPinStrength,
unsigned long ulPinType);

extern void PinTypeUART(unsigned long ulPin, unsigned long ulPinMode);
extern void PinTypeI2C(unsigned long ulPin, unsigned long ulPinMode);
extern void PinTypeSPI(unsigned long ulPin, unsigned long ulPinMode);
extern void PinTypeI2S(unsigned long ulPin, unsigned long ulPinMode);
extern void PinTypeTimer(unsigned long ulPin, unsigned long ulPinMode);
extern void PinTypeCamera(unsigned long ulPin, unsigned long ulPinMode);
extern void PinTypeGPIO(unsigned long ulPin, unsigned long ulPinMode,
tBoolean bOpenDrain);
extern void PinTypeADC(unsigned long ulPin, unsigned long ulPinMode);
extern void PinTypeSDHost(unsigned long ulPin, unsigned long ulPinMode);

#define __cplusplus

#endif

#endif //__PIN_H__
prcm.h

1  /******************************************************************************
2  ***********************************************************
3  //
4  //
5  //  prcm.h
6  //
7  //  Prototypes for the PRCM control driver.
8  //
9  //  Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
10 //
11 //
12 //  Redistribution and use in source and binary forms, with or without
13 //  modification, are permitted provided that the following conditions
14 //  are met:
15 //
16 //  Redistributions of source code must retain the above copyright
17 //  notice, this list of conditions and the following disclaimer.
18 //
19 //  Redistributions in binary form must
reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the
distribution.

Neither the name of Texas Instruments Incorporated nor the names of
its contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#ifndef __PRCM_H__
define __PRCM_H__
#endif

if building with a C++ compiler, make all
of the definitions in this header
have a C binding.

// Peripheral clock and reset control
registers

typedef struct _PRCM_PeripheralRegs_
{
    unsigned long ulClkReg;
    unsigned long ulRstReg;
} PRCM_PeriphRegs_t;
// Values that can be passed to
PRCMPeripheralEnable() and
// PRCMPeripheralDisable()

// Values that can be passed to
PRCMSRAMRetentionEnable() and
// PRCMSRAMRetentionDisable() as ulSramColSel.

// Values that can be passed to
PRCMSRAMRetentionEnable() and
// PRCMSRAMRetentionDisable() as
ulModeFlags.

// Values that can be passed to
PRCMLPDSWakeupSourceEnable(),
// PRCMLPDSWakeupCauseGet() and
PRCMLPDSWakeupSourceDisable().
#endif

#define PRCM_LPDS_HOST_IRQ 0x00000080
#define PRCM_LPDS_GPIO 0x00000010
#define PRCM_LPDS_TIMER 0x00000001

// Values that can be passed to PRCMLPDSWakeupGPIOSelect() as Type
#define PRCM_LPDS_LOW_LEVEL 0x00000002
#define PRCM_LPDS_HIGH_LEVEL 0x00000000
#define PRCM_LPDS_FALL_EDGE 0x00000001
#define PRCM_LPDS_RISE_EDGE 0x00000003

#define PRCM_LPDS_GPIO2 0x00000000
#define PRCM_LPDS_GPIO4 0x00000001
#define PRCM_LPDS_GPIO13 0x00000002
#define PRCM_LPDS_GPIO17 0x00000003
#define PRCM_LPDS_GPIO11 0x00000004
#define PRCM_LPDS_GPIO24 0x00000005
#define PRCM_LPDS_GPIO26 0x00000006

// Values that can be passed to PRCMHibernateWakeupSourceEnable(),
// PRCMHibernateWakeupSourceDisable().
#define PRCM_HIB_SLOW_CLK_CTR 0x00000001

// Values that can be passed to PRCMHibernateWakeUpGPIOSelect() as ulType

#define PRCM_HIB_LOW_LEVEL 0x00000000
#define PRCM_HIB_HIGH_LEVEL 0x00000001
#define PRCM_HIB_FALL_EDGE 0x00000002
#define PRCM_HIB_RISE_EDGE 0x00000003

// Values that can be passed to PRCMHibernateWakeupSourceEnable(),
// PRCMHibernateWakeupSourceDisable(),
// PRCMHibernateWakeUpGPIOSelect()

#define PRCM_HIB_GPIO2 0x00010000
#define PRCM_HIB_GPIO4 0x00020000
#define PRCM_HIB_GPIO13 0x00040000
#define PRCM_HIB_GPIO17 0x00080000
#define PRCM_HIB_GPIO11 0x00100000
#define PRCM_HIB_GPIO24 0x00200000
#define PRCM_HIB_GPIO26 0x00400000

// Values that will be returned from PRCMSysResetCauseGet().

#define PRCM_HIB_GPIO2 0x00010000
```c
#define PRCM_POWER_ON            0x00000000
#define PRCM_LPDS_EXIT           0x00000001
#define PRCM_CORE_RESET         0x00000003
#define PRCM_MCU_RESET          0x00000004
#define PRCM_WDT_RESET          0x00000005
#define PRCM_SOC_RESET          0x00000006
#define PRCM_HIB_EXIT           0x00000007

// Values that can be passed to PRCMHibernateWakeupCauseGet().
#define PRCM_HIB_WAKEUP_CAUSE_SLOW_CLOCK 0x00000002
#define PRCM_HIB_WAKEUP_CAUSE_GPIO     0x00000004

// Values that can be passed to PRCMSEnableInterrupt
#define PRCM_INT_SLOW_CLK_CTR       0x00004000

// Values that can be passed to PRCMPeripheralClkEnable(), PRCMPeripheralClkDisable(), PRCMPeripheralReset()
#define PRCM_CAMERA                  0x00000000
#define PRCM_I2S                     0x00000001
```
#define PRCM_SDHOST 0x00000002
#define PRCM_GSPI 0x00000003
#define PRCM_LSPI 0x00000004
#define PRCM_UDMA 0x00000005
#define PRCM_GPIOA0 0x00000006
#define PRCM_GPIOA1 0x00000007
#define PRCM_GPIOA2 0x00000008
#define PRCM_GPIOA3 0x00000009
#define PRCM_GPIOA4 0x0000000A
#define PRCM_WDT 0x0000000B
#define PRCM_UARTA0 0x0000000C
#define PRCM_UARTA1 0x0000000D
#define PRCM_TIMERA0 0x0000000E
#define PRCM_TIMERA1 0x0000000F
#define PRCM_TIMERA2 0x00000010
#define PRCM_TIMERA3 0x00000011
#define PRCM_DTHE 0x00000012
#define PRCM_SSPI 0x00000013
#define PRCM_I2CA0 0x00000014

// Note: PRCM_ADC is a dummy define for pinmux utility code generation
// PRCM_ADC should never be used in any user code.
#define PRCM_ADC 0x000000FF

// ****************************************
// API Function prototypes
// ****************************************
extern void PRCMMCUReset(tBoolean bIncludeSubsystem);
extern unsigned long PRCMSysResetCauseGet(void);
extern void PRCMPeripheralClkEnable(unsigned long ulPeripheral, unsigned long ulClkFlags);
extern void PRCMPeripheralClkDisable(unsigned long ulPeripheral, unsigned long ulClkFlags);
extern void PRCMPeripheralReset(unsigned long ulPeripheral);
extern tBoolean PRCMPeripheralStatusGet(unsigned long ulPeripheral);
extern void PRCMI2SClockFreqSet(unsigned long ulI2CClkFreq);
extern unsigned long PRCMPeripheralClockGet(unsigned long ulPeripheral);
extern void PRCMSleepEnter(void);
extern void PRCMSRAMRetentionEnable(unsigned long ulSramColSel, unsigned long ulFlags);
extern void PRCMSRAMRetentionDisable(unsigned long ulSramColSel, unsigned long ulFlags);
extern void PRCMLPDSRestoreInfoSet(unsigned long ulRestoreSP, unsigned long ulRestorePC);
extern void PRCMLPDEnter(void);
extern void PRCMLPDSIntervalSet(unsigned long ulTicks);

extern void PRCMLPDSWakeupSourceEnable(unsigned long ulPdswakeupSrc);

extern unsigned long PRCMLPDSWakeupCauseGet(void);

extern void PRCMLPDSWakeupGPIOSelect(unsigned long ulGPIOPin,
  unsigned long ulType);

extern void PRCMLPDSWakeupSourceDisable(unsigned long ulPdswakeupSrc);

extern void PRCMHibernateEnter(void);

extern void PRCMHibernateWakeupSourceEnable(unsigned long ulHIBwakeupSrc);

extern unsigned long PRCMHibernateWakeupCauseGet(void);

extern void PRCMHibernateWakeupGPIOSelect(unsigned long ulMultiGPIOBitMap,
  unsigned long ulType);

extern void PRCMHibernateWakeupSourceDisable(unsigned long ulHIBwakeupSrc);

extern void PRCMHibernateIntervalSet(unsigned long long ullTicks);

extern unsigned long long PRCMSlowClkCtrGet(void);
extern void PRCMSlowClkCtrFastGet(void);

extern void PRCMSlowClkCtrMatchSet(unsigned long long ullTicks);

extern unsigned long long PRCMSlowClkCtrMatchGet(void);

extern void PRCMOCRRegisterWrite(unsigned char ucIndex,
       unsigned long ulRegValue);

extern unsigned long PRCMOCRRegisterRead(unsigned char ucIndex);

extern void PRCMIntRegister(void (*pfnHandler)(void));

extern void PRCMIntUnregister(void);

extern void PRCMIntEnable(unsigned long ulIntFlags);

extern void PRCMIntDisable(unsigned long ulIntFlags);

extern unsigned long PRCMIntStatus(void);

extern void PRCMRTCInUseSet(void);

extern tBoolean PRCMRTCInUseGet(void);

extern void PRCMRTCSet(unsigned long ulSecs,
      unsigned short usMsec);

extern void PRCMRTCGet(unsigned long *ulSecs,
      unsigned short *usMsec);

extern void PRCMRTCMatchSet(unsigned long ulSecs,
      unsigned short usMsec);

extern void PRCMRTCMatchGet(unsigned long *ulSecs,
      unsigned short *usMsec);

extern void PRCMCC3200MCUInit(void);

extern unsigned long PRCMHBRegRead(unsigned long ulRegAddr);

extern void PRCMHBRegWrite(unsigned long ulRegAddr, unsigned long ulValue);

extern unsigned long
PRCMCameraFreqSet(unsigned char ulDivider,
                unsigned char ulWidth);

extern void PRCMLPDEnterKeepDebugIf(void);

//******************************************************
//	Mark	the	end	of	the	C	bindings	section
//
//
//
// Mark the end of the C bindings section for C++ compilers.
//******************************************************

#ifdef __cplusplus
}
#endif
#endif // __PRCM_H__
Macros to facilitate calling functions in the ROM.

Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
// THIS IS AN AUTO-GENERATED FILE. DO NOT EDIT BY HAND.

#ifndef __ROM_H__
#define __ROM_H__

// Pointers to the main API tables.

#define ROM_APITABLE ((unsigned long *)0x0000040C)
#define ROM_VERSION (ROM_APITABLE[0])
#define ROM_UARTTABLE ((unsigned long *)(ROM_APITABLE[1]))
#define ROM_TIMERTABLE ((unsigned long *)(ROM_APITABLE[2]))
#define ROM_WATCHDOGTABLE ((unsigned long *)(ROM_APITABLE[3]))
#define ROM_INTERRUPTTABLE ((unsigned long *)(ROM_APITABLE[4]))
#define ROM_UDMATABLE ((unsigned long *)(ROM_APITABLE[5]))
#define ROM_PRCMTABLE ((unsigned long *)(ROM_APITABLE[6]))
#define ROM_I2CTABLE ((unsigned long *)(ROM_APITABLE[7]))
#define ROM_SPITABLE ((unsigned long *)(ROM_APITABLE[8]))
#define ROM_CAMERATABLE ((unsigned long *)(ROM_APITABLE[9]))
#define ROM_FLASHTABLE ((unsigned long *)(ROM_APITABLE[10]))
#define ROM_PINTABLE ((unsigned long *)(ROM_APITABLE[11]))
#define ROM_SYSTICKTABLE ((unsigned long *)(ROM_APITABLE[12]))
#define ROM_UTILSTABLE ((unsigned long *)(ROM_APITABLE[13]))
#define ROM_I2STABLE ((unsigned long *)(ROM_APITABLE[14]))
#define ROM_HWSPINLOCKTABLE ((unsigned long *)(ROM_APITABLE[15]))
#define ROM_GPIOTABLE ((unsigned long *)(ROM_APITABLE[16]))
#define ROM_AESTABLE ((unsigned long *)(ROM_APITABLE[17]))
#define ROM_DESTABLE ((unsigned long *)(ROM_APITABLE[18]))
#define ROM_SHAMD5TABLE ((unsigned long *)(ROM_APITABLE[19]))
#define ROM_CRCTABLE ((unsigned long *)(ROM_APITABLE[20]))
#define ROM_SDHOSTTABLE ((unsigned long *)(ROM_APITABLE[21]))
#define ROM_ADCTABLE ((unsigned long *)(ROM_APITABLE[22]))
// Macros for calling ROM functions in the Interrupt API.

// *****************************************************
// *****************************************************
#if defined(TARGET_IS_CC3200)
#define ROM_IntEnable
  ((void (*)(unsigned long ulInterrupt))ROM_INTERRUPTTABLE[0])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_IntMasterEnable
  ((tBoolean (*)(void))ROM_INTERRUPTTABLE[1])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_IntMasterDisable
  ((tBoolean (*)(void))ROM_INTERRUPTTABLE[2])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_IntDisable
  ((void (*)(unsigned long ulInterrupt))ROM_INTERRUPTTABLE[3])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_IntPriorityGroupingSet
  ((void (*)(unsigned long ulBits))ROM_INTERRUPTTABLE[4])
#endif
#define ROM_IntPriorityGroupingGet \
((unsigned long (*)(void))ROM_INTERRUPTTABLE[5])

#if defined(TARGET_IS_CC3200)
#define ROM_IntPrioritySet \
((void (*)(unsigned long, unsigned char))(void (*)(unsigned long, unsigned char))ROM_INTERRUPTTABLE[6])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_IntPriorityGet \
((long (*)(unsigned long))(long (*)(unsigned long))ROM_INTERRUPTTABLE[7])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_IntPendSet \
((void (*)(unsigned long))(void (*)(unsigned long))ROM_INTERRUPTTABLE[8])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_IntPendClear \
((void (*)(unsigned long))(void (*)(unsigned long))ROM_INTERRUPTTABLE[9])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_IntPriorityMaskSet \
((void (*)(unsigned long))(void (*)(unsigned long))ROM_INTERRUPTTABLE[10])
#endif
// Macros for calling ROM functions in the Timer API.

#define ROM_TimerEnable

#define ROM_IntPriorityMaskGet ((unsigned long(*)(void))ROM_INTERRUPTTABLE[11])

#define ROM_IntRegister ((void(*)(unsigned long ulInterrupt,void(*)(void)))ROM_INTERRUPTTABLE[12])

#define ROM_IntUnregister ((void(*)(unsigned long ulInterrupt)))ROM_INTERRUPTTABLE[13])

#define ROM_IntVTableBaseSet ((void(*)(unsigned long ulVtableBase)))ROM_INTERRUPTTABLE[14])
((void (*)(unsigned long ulBase, unsigned long ulTimer))ROM_TIMERTABLE[0])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_TimerDisable

((void (*)(unsigned long ulBase, unsigned long ulTimer))ROM_TIMERTABLE[1])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_TimerConfigure

((void (*)(unsigned long ulBase, unsigned long ulConfig))ROM_TIMERTABLE[2])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_TimerControlLevel

((void (*)(unsigned long ulBase, unsigned long ulTimer, tBoolean bInvert))ROM_TIMERTABLE[3])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_TimerControlEvent

((void (*)(unsigned long ulBase, unsigned long ulTimer, ...


tBoolean bInvert))ROM_TIMERTABLE[3])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_TimerControlEvent

((void (*)(unsigned long ulBase, unsigned long ulTimer, ...


tBoolean bInvert))ROM_TIMERTABLE[3])
#endif
unsigned long ulTimer, \
unsigned long ulEvent))ROM_TIMERTABLE[4])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerControlStall
\n((void (*)(unsigned long ulBase, \
unsigned long ulTimer, \
tBoolean bStall))ROM_TIMERTABLE[5])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerPrescaleSet
\n((void (*)(unsigned long ulBase, \
unsigned long ulTimer, \
unsigned long ulValue))ROM_TIMERTABLE[6])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerPrescaleMatchSet
\n((unsigned long(*)(unsigned long ulBase, \
unsigned long ulTimer))ROM_TIMERTABLE[7])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerPrescaleMatchSet
unsigned long ulTimer,
unsigned long ulValue))ROM_TIMERTABLE[8])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerPrescaleMatchGet
((unsigned long (*)(unsigned long ulBase,
    unsigned long ulTimer))ROM_TIMERTABLE[9])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerLoadSet
((void (*)(unsigned long ulBase,
    unsigned long ulTimer,
    unsigned long ulValue))ROM_TIMERTABLE[10])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerLoadGet
((unsigned long (*)(unsigned long ulBase,
    unsigned long ulTimer))ROM_TIMERTABLE[11])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerValueGet
((unsigned long (*)(unsigned long ulBase,
    unsigned long ulTimer))ROM_TIMERTABLE[12])
#endif
unsigned long ulTimer))ROM_TIMERTABLE[12])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerMatchSet

((void (*)(unsigned long ulBase,
unsigned long ulTimer,
unsigned long ulValue))ROM_TIMERTABLE[13])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerMatchGet

((unsigned long (*)(unsigned long ulBase,
unsigned long ulTimer))ROM_TIMERTABLE[14])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerIntRegister

((void (*)(unsigned long ulBase,
unsigned long ulTimer,
void (*pfnHandler)(void)))ROM_TIMERTABLE[15])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_TimerIntUnregister

((void (*)(unsigned long ulBase,
unsigned long ulTimer,
void (*pfnHandler)(void)))ROM_TIMERTABLE[15])
#endif
ulTimer))ROM_TIMERTABLE[16])
  #endif
  #if defined(TARGET_IS_CC3200)
  #define ROM_TimerIntEnable
  \n  ((void (*)(unsigned long ulBase,
  \n  unsigned long ulIntFlags))ROM_TIMERTABLE[17])
  #endif
  #if defined(TARGET_IS_CC3200)
  #define ROM_TimerIntDisable
  \n  ((void (*)(unsigned long ulBase,
  \n  unsigned long ulIntFlags))ROM_TIMERTABLE[18])
  #endif
  #if defined(TARGET_IS_CC3200)
  #define ROM_TimerIntStatus
  \n  ((unsigned long (*)(unsigned long ulBase,
  \n  tBoolean bMasked))ROM_TIMERTABLE[19])
  #endif
  #if defined(TARGET_IS_CC3200)
  #define ROM_TimerIntClear
  \n  ((void (*)(unsigned long ulBase,
  \n  unsigned long ulIntFlags))ROM_TIMERTABLE[20])
  #endif

//**************************************************************************
***************************************************************************
 Macros for calling ROM functions in the UART API.

******************************************
******************************************
#if defined(TARGET_IS_CC3200)
#define ROM_UARTParityModeSet
   ((void (*)(unsigned long ulBase,
            unsigned long ulParity))ROM_UARTTABLE[0])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_UARTParityModeGet
   ((unsigned long (*)(unsigned long ulBase))ROM_UARTTABLE[1])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_UARTFIFOLevelSet
   ((void (*)(unsigned long ulBase,
            unsigned long ulTxLevel,
            unsigned long ulRxLevel))ROM_UARTTABLE[2])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_UARTFIFOLevelGet
   ((void (*)(unsigned long ulBase,
            unsigned long *pulTxLevel,
            unsigned long *pulRxLevel))ROM_UARTTABLE[3])
#endif
unsigned long *pulRxLevel))ROM_UARTTABLE[3])
#endif
#if defined(TARGET_IS_CC3200)
define ROM_UARTConfigSetExpClk
  ((void (*)(unsigned long ulBase,
    unsigned long ulUARTClk,
    unsigned long ulBaud,
    unsigned long ulConfig))ROM_UARTTABLE[4])
#endif
#if defined(TARGET_IS_CC3200)
define ROM_UARTConfigGetExpClk
  ((void (*)(unsigned long ulBase,
    unsigned long ulUARTClk,
    unsigned long *pulBaud,
    unsigned long *pulConfig))ROM_UARTTABLE[5])
#endif
#if defined(TARGET_IS_CC3200)
define ROM_UARTEnable
  ((void (*)(unsigned long ulBase))ROM_UARTTABLE[6])
#endif
#if defined(TARGET_IS_CC3200)
define ROM_UARTDisable
  ((void (*)(unsigned long ulBase))ROM_UARTTABLE[7])
#endif
ulBase))ROM_UARTTABLE[7])
#define ROM_UARTFIFOEnable
    ((void (*)(unsigned long ulBase))ROM_UARTTABLE[8])
#endif
#define ROM_UARTFIFODisable
    ((void (*)(unsigned long ulBase))ROM_UARTTABLE[9])
#endif
#define ROM_UARTCharsAvail
    ((tBoolean (*)(unsigned long ulBase))ROM_UARTTABLE[10])
#endif
#define ROM_UARTSpaceAvail
    ((tBoolean (*)(unsigned long ulBase))ROM_UARTTABLE[11])
#endif
#define ROM_UARTCharGetNonBlocking
    ((long (*)(unsigned long ulBase))ROM_UARTTABLE[12])
#endif
#define ROM_UARTCharGet
    ((long (*)(unsigned long ulBase))ROM_UARTTABLE[13])
337  #endif
338  #if defined(TARGET_IS_CC3200)
339  #define ROM_UARTCharPutNonBlocking
340     ((tBoolean (*)(unsigned long ulBase,
341                      unsigned char ucData))ROM_UARTTABLE[14])
342  #endif
343  #if defined(TARGET_IS_CC3200)
344  #define ROM_UARTCharPut
345     ((void (*)(unsigned long ulBase,
346                      unsigned char ucData))ROM_UARTTABLE[15])
347  #endif
348  #if defined(TARGET_IS_CC3200)
349  #define ROM_UARTBreakCtl
350     ((void (*)(unsigned long ulBase,
351                      tBoolean bBreakState))ROM_UARTTABLE[16])
352  #endif
353  #if defined(TARGET_IS_CC3200)
354  #define ROM_UARTBusy
355     ((tBoolean (*)(unsigned long ulBase))ROM_UARTTABLE[17])
356  #endif
357  #if defined(TARGET_IS_CC3200)
358  #define ROM_UARTIntRegister
359     ((void (*)(unsigned long ulBase,
360                      void(*pfnHandler)
(void)) ROM_UARTTABLE[18])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_UARTIntUnregister
  ((void (*)(unsigned long ulBase)) ROM_UARTTABLE[19])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_UARTIntEnable
  ((void (*)(unsigned long ulBase,
            unsigned long ulIntFlags)) ROM_UARTTABLE[20])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_UARTIntDisable
  ((void (*)(unsigned long ulBase,
            unsigned long ulIntFlags)) ROM_UARTTABLE[21])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_UARTIntStatus
  ((unsigned long (*)(unsigned long ulBase,
                       tBoolean bMasked)) ROM_UARTTABLE[22])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_UARTIntClear
  ((void (*)(unsigned long ulBase,
            unsigned long ulIntFlags)) ROM_UARTTABLE[23])
#endif
unsigned long ulIntFlags))ROM_UARTTABLE[23])

#endif
#if defined(TARGET_IS_CC3200)
#define ROM_UARTDMAEnable
    ((void (*)(unsigned long ulBase,
               unsigned long ulDMAFlags))ROM_UARTTABLE[24])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_UARTDMADisable
    ((void (*)(unsigned long ulBase,
               unsigned long ulDMAFlags))ROM_UARTTABLE[25])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_UARTRxErrorGet
    ((unsigned long (*)(unsigned long ulBase))ROM_UARTTABLE[26])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_UARTRxErrorClear
    ((void (*)(unsigned long ulBase))ROM_UARTTABLE[27])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_UARTModemControlSet
    ((void (*)(unsigned long ulBase,
               unsigned long ulIntFlags))ROM_UARTTABLE[23])
#endif
ulControl))ROM_UARTTABLE[28])
408  #endif
409  #if defined(TARGET_IS_CC3200)
410  #define ROM_UARTModemControlClear
  
  ((void (*)(unsigned long ulBase,
  
  unsigned long
  ulControl))ROM_UARTTABLE[29])
413  #endif
414  #if defined(TARGET_IS_CC3200)
415  #define ROM_UARTModemControlGet
  
  ((unsigned long (*)(unsigned long
  ulBase))ROM_UARTTABLE[30])
417  #endif
418  #if defined(TARGET_IS_CC3200)
419  #define ROM_UARTModemStatusGet
  
  ((unsigned long (*)(unsigned long
  ulBase))ROM_UARTTABLE[31])
421  #endif
422  #if defined(TARGET_IS_CC3200)
423  #define ROM_UARTFlowControlSet
  
  ((void (*)(unsigned long ulBase,
  
  unsigned long
  ulMode))ROM_UARTTABLE[32])
426  #endif
427  #if defined(TARGET_IS_CC3200)
428  #define ROM_UARTFlowControlGet
  
  ((unsigned long (*)(unsigned long
  ulBase))ROM_UARTTABLE[33])
430  #endif
431  #if defined(TARGET_IS_CC3200)
#define ROM_UARTTxIntModeSet
\( ((void (*)(unsigned long ulBase, \)
\( unsigned long \)
\( ulMode))\)ROM_UARTTABLE[34])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_UARTTxIntModeGet
\( ((unsigned long (*)(unsigned long \)
\( ulBase))\)ROM_UARTTABLE[35])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAChannelTransferSet
\( ((void (*)(unsigned long \)
\( ulChannelStructIndex, \)
\( unsigned long ulMode, \)
\( void *pvSrcAddr, \)
\( void *pvDstAddr, \)
\( unsigned long \)
\( ulTransferSize))\)ROM_UDMATABLE[0])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAEnable
    ((void (*)(void))ROM_UDMATABLE[1])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMADisable
    ((void (*)(void))ROM_UDMATABLE[2])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAErrorStatusGet
    ((unsigned long (*)(void))ROM_UDMATABLE[3])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAErrorStatusClear
    ((void (*)(void))ROM_UDMATABLE[4])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAChannelEnable
    ((void (*)(unsigned long ulChannelNum))ROM_UDMATABLE[5])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAChannelDisable
    ((void (*)(unsigned long ulChannelNum))ROM_UDMATABLE[6])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAChannelIsEnabled
    ((tBoolean (*)(unsigned long ulChannelNum))ROM_UDMATABLE[7])
#ifndef
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAControlBaseSet
    ((void (*)(void *pControlTable))ROM_UDMATABLE[8])
#endif
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAControlBaseGet
    ((void (*)(void))ROM_UDMATABLE[9])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAChannelRequest
    ((void (*)(unsigned long ulChannelNum))ROM_UDMATABLE[10])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAChannelAttributeEnable
    ((void (*)(unsigned long ulChannelNum, unsigned long ulAttr))ROM_UDMATABLE[11])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAChannelAttributeDisable
    ((void (*)(unsigned long ulChannelNum, unsigned long ulAttr))ROM_UDMATABLE[12])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_uDMAChannelAttributeGet
    //
#endif
506  ((unsigned long (*)(unsigned long
  ulChannelNum))ROM_UDMATABLE[13])
507  #endif
508  #if defined(TARGET_IS_CC3200)
509  #define ROM_uDMAChannelControlSet
510        ((void (*)(unsigned long
            ulChannelStructIndex,
            unsigned long
            ulControl))ROM_UDMATABLE[14])
511  #endif
512  #if defined(TARGET_IS_CC3200)
513  #define ROM_uDMAChannelSizeGet
514        ((unsigned long (*)(unsigned long
            ulChannelStructIndex))ROM_UDMATABLE[15])
515  #endif
516  #if defined(TARGET_IS_CC3200)
517  #define ROM_uDMAChannelModeGet
518        ((unsigned long (*)(unsigned long
            ulChannelStructIndex))ROM_UDMATABLE[16])
519  #endif
520  #if defined(TARGET_IS_CC3200)
521  #define ROM_uDMAIntStatus
522        ((unsigned long (*)(void))ROM_UDMATABLE[17])
523  #endif
524  #if defined(TARGET_IS_CC3200)
525  #define ROM_uDMAIntClear
526        ((void (*)(unsigned long
            ulChanMask))ROM_UDMATABLE[18])
527  #endif
528  #if defined(TARGET_IS_CC3200)
529  #define ROM_uDMAControlAlternateBaseGet
531    ((void (*)(void))ROM_UDMATABLE[19])
532  #endif
533  #if defined(TARGET_IS_CC3200)
534  #define ROM_uDMAChannelScatterGatherSet
535    ((void (*)(unsigned long
       ulChannelNum,
       unsigned ulTaskCount,
       void *pvTaskList,
       unsigned long
       ulIsPeriphSG))ROM_UDMATABLE[20])
536  #endif
537  #if defined(TARGET_IS_CC3200)
538  #define ROM_uDMAChannelAssign
539    ((void (*)(unsigned long
       ulMapping))ROM_UDMATABLE[21])
540  #endif
541  #if defined(TARGET_IS_CC3200)
542  #define ROM_uDMAIntRegister
543    ((void (*)(unsigned long
       ulIntChannel,
       void (*pfnHandler)(void)))ROM_UDMATABLE[22])
544  #endif
545  #if defined(TARGET_IS_CC3200)
546  #define ROM_uDMAIntUnregister
547    ((void (*)(unsigned long
       ulIntChannel))ROM_UDMATABLE[23])
548  #endif
549  #if defined(TARGET_IS_CC3200)
550  #define ROM_uDMAIntUnregister
551    ((void (*)(unsigned long
       ulIntChannel))ROM_UDMATABLE[23])
552  #endif
Macros for calling ROM functions in the Watchdog API.

```c
#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogIntClear
   ((void (*)(unsigned long ulBase))ROM_WATCHDOGTABLE[0])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogRunning
   ((tBoolean (*)(unsigned long ulBase))ROM_WATCHDOGTABLE[1])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogEnable
   ((void (*)(unsigned long ulBase))ROM_WATCHDOGTABLE[2])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogLock
   ((void (*)(unsigned long ulBase))ROM_WATCHDOGTABLE[3])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogUnlock
   ((void (*)(unsigned long ulBase))ROM_WATCHDOGTABLE[4])
#endif
```
#ifdef TARGET_IS_CC3200
#define ROM_WatchdogLockState
    \n    ((tBoolean (*)(unsigned long ulBase))ROM_WATCHDOGTAB[5])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogReloadSet
    \n    ((void (*)(unsigned long ulBase,
             unsigned long ulLoadVal))ROM_WATCHDOGTAB[6])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogReloadGet
    \n    ((unsigned long (*)(unsigned long ulBase))ROM_WATCHDOGTAB[7])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogValueGet
    \n    ((unsigned long (*)(unsigned long ulBase))ROM_WATCHDOGTAB[8])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogIntStatus
    \n    ((unsigned long (*)(unsigned long ulBase,
                          tBoolean bMasked))ROM_WATCHDOGTAB[10])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_WatchdogStallEnable
((void (*)(unsigned long ulBase))ROM_WATCHDOGTABLE[11]) #endif

#ifdef TARGET_IS_CC3200
#define ROM_WatchdogStallDisable
#endif

((void (*)(unsigned long ulBase))ROM_WATCHDOGTABLE[12]) #endif

#ifdef TARGET_IS_CC3200
#define ROM_WatchdogIntRegister
#endif

(void (*)(unsigned long ulBase, 
     void(*pfnHandler)
     (void)))ROM_WATCHDOGTABLE[13]) #endif

#ifdef TARGET_IS_CC3200
#define ROM_WatchdogIntUnregister
#endif

((void (*)(unsigned long ulBase))ROM_WATCHDOGTABLE[14])

//******************************************
***********************************
//
//
// Macros for calling ROM functions in the I2C API.
//
//******************************************
***********************************

#if defined(TARGET_IS_CC3200)
#define ROM_I2CIntRegister
#endif

(void (*)(uint32_t ui32Base,
\void(pfnHandler)
(void))\ROM_I2CTABLE[0])
\#endif
\#if defined(TARGET_IS_CC3200)
\#define ROM_I2CIntUnregister
\=((void (*)(uint32_t
ui32Base))\ROM_I2CTABLE[1])
\#endif
\#if defined(TARGET_IS_CC3200)
\#define ROM_I2CTxFIFOConfigSet
\=((void (*)(uint32_t	ui32Base,
ui32Config))\ROM_I2CTABLE[2])
\#endif
\#if defined(TARGET_IS_CC3200)
\#define ROM_I2CTxFIFOFlush
\=((void (*)(uint32_t	ui32Base))\ROM_I2CTABLE[3])
\#endif
\#if defined(TARGET_IS_CC3200)
\#define ROM_I2CRxFIFOConfigSet
\=((void (*)(uint32_t	ui32Base,
ui32Config))\ROM_I2CTABLE[4])
\#endif
\#if defined(TARGET_IS_CC3200)
\#define ROM_I2CRxFIFOFlush
\=((void (*)(uint32_t	ui32Base))\ROM_I2CTABLE[5])
#ifndef
# if defined(TARGET_IS_CC3200)
#define ROM_I2CFIFOSTatus
\
((uint32_t (*)(uint32_t
ui32Base))ROM_I2CTABLE[6])
#endif
# if defined(TARGET_IS_CC3200)
#define ROM_I2CFIFODataPut
\
((void (*)(uint32_t ui32Base,
\
uint8_t
ui8Data))ROM_I2CTABLE[7])
#endif
# if defined(TARGET_IS_CC3200)
#define ROM_I2CFIFODataPutNonBlocking
\
((uint32_t (*)(uint32_t ui32Base,
\
uint8_t
ui8Data))ROM_I2CTABLE[8])
#endif
# if defined(TARGET_IS_CC3200)
#define ROM_I2CFIFODataGet
\
((uint32_t (*)(uint32_t ui32Base))ROM_I2CTABLE[9])
#endif
# if defined(TARGET_IS_CC3200)
#define ROM_I2CFIFODataGetNonBlocking
\
((uint32_t (*)(uint32_t ui32Base,
\
uint8_t
*pui8Data))ROM_I2CTABLE[10])
#endif
#ifdef TARGET_IS_CC3200
#define ROM_I2CMasterBurstLengthSet
   ((void (*)(uint32_t ui32Base,
            
   uint8_t ui8Length))ROM_I2CTABLE[11])
#endif

#ifdef TARGET_IS_CC3200
#define ROM_I2CMasterBurstCountGet
   ((uint32_t (*)(uint32_t ui32Base))ROM_I2CTABLE[12])
#endif

#ifdef TARGET_IS_CC3200
#define ROM_I2CMasterGlitchFilterConfigSet
   ((void (*)(uint32_t ui32Base,
            
   uint32_t ui32Config))ROM_I2CTABLE[13])
#endif

#ifdef TARGET_IS_CC3200
#define ROM_I2CSlaveFIFOEnable
   ((void (*)(uint32_t ui32Base,
            
   uint32_t ui32Config))ROM_I2CTABLE[14])
#endif

#ifdef TARGET_IS_CC3200
#define ROM_I2CSlaveFIFODisable
   ((void (*)(uint32_t ui32Base))ROM_I2CTABLE[15])
#endif

#ifdef TARGET_IS_CC3200
#define ROM_I2CMasterBusBusy
\((bool (*)(uint32_t ui32Base))\)ROM_I2CTABLE[16])

#if defined(TARGET_IS_CC3200)
#define ROM_I2CMasterBusy
\((bool (*)(uint32_t ui32Base))\)ROM_I2CTABLE[17])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_I2CMasterControl
\((void (*)(uint32_t ui32Base, uint32_t ui32Cmd))\)ROM_I2CTABLE[18])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_I2CMasterDataGet
\((uint32_t (*)(uint32_t ui32Base))\)ROM_I2CTABLE[19])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_I2CMasterDataPut
\((void (*)(uint32_t ui32Base, uint8_t ui8Data))\)ROM_I2CTABLE[20])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_I2CMasterDisable
\((void (*)(uint32_t ui32Base))\)ROM_I2CTABLE[21])
#endif
ui32Base))ROM_I2CTABLE[21])

722  #endif
723  #if defined(TARGET_IS_CC3200)
724  #define ROM_I2CMasterEnable
725  ((void (*)(uint32_t
    ui32Base))ROM_I2CTABLE[22])
726  #endif
727  #if defined(TARGET_IS_CC3200)
728  #define ROM_I2CMasterErr
729  ((uint32_t (*)(uint32_t
    ui32Base))ROM_I2CTABLE[23])
730  #endif
731  #if defined(TARGET_IS_CC3200)
732  #define ROM_I2CMasterIntClear
733  ((void (*)(uint32_t
    ui32Base))ROM_I2CTABLE[24])
734  #endif
735  #if defined(TARGET_IS_CC3200)
736  #define ROM_I2CMasterIntDisable
737  ((void (*)(uint32_t
    ui32Base))ROM_I2CTABLE[25])
738  #endif
739  #if defined(TARGET_IS_CC3200)
740  #define ROM_I2CMasterIntEnable
741  ((void (*)(uint32_t
    ui32Base))ROM_I2CTABLE[26])
742  #endif
743  #if defined(TARGET_IS_CC3200)
744  #define ROM_I2CMasterIntStatus
745  ((bool (*)(uint32_t ui32Base,
bool bMasked))ROM_I2CTABLE[27])
#endif
#endif
#define ROM_I2CMasterIntEnableEx
((void (*)(uint32_t ui32Base,
  uint32_t ui32IntFlags))ROM_I2CTABLE[28])
#endif
#endif
#define ROM_I2CMasterIntDisableEx
((void (*)(uint32_t ui32Base,
  uint32_t ui32IntFlags))ROM_I2CTABLE[29])
#endif
#endif
#define ROM_I2CMasterIntStatusEx
((uint32_t (*)(uint32_t ui32Base,
  bool bMasked))ROM_I2CTABLE[30])
#endif
#endif
#define ROM_I2CMasterIntClearEx
((void (*)(uint32_t ui32Base,
  uint32_t ui32IntFlags))ROM_I2CTABLE[31])
#endif
#endif
#define ROM_I2CMasterTimeoutSet
((void (*)(uint32_t ui32Base,
        uint32_t ui32Value))ROM_I2CTABLE[32])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveACKOverride
((void (*)(uint32_t ui32Base,
        bool bEnable))ROM_I2CTABLE[33])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveACKValueSet
((void (*)(uint32_t ui32Base,
        bool bACK))ROM_I2CTABLE[34])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CMasterLineStateGet
((uint32_t (*)(uint32_t ui32Base))ROM_I2CTABLE[35])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CMasterSlaveAddrSet
((void (*)(uint32_t ui32Base,
        uint8_t ui8SlaveAddr,
        bool bReceive))ROM_I2CTABLE[36])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveDataGet
    ((uint32_t (*)(uint32_t
            ui32Base))ROM_I2CTABLE[37])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveDataPut
    ((void (*)(uint32_t	ui32Base,
            uint8_t
            ui8Data))ROM_I2CTABLE[38])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveDisable
    ((void (*)(uint32_t
            ui32Base))ROM_I2CTABLE[39])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveEnable
    ((void (*)(uint32_t
            ui32Base))ROM_I2CTABLE[40])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveInit
    ((void (*)(uint32_t	ui32Base,
            uint8_t
            ui8SlaveAddr))ROM_I2CTABLE[41])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveAddressSet
((void (*)(uint32_t ui32Base, uint8_t ui8AddrNum, uint8_t ui8SlaveAddr))ROM_I2CTABLE[42])

#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveIntClear
#endif

((void (*)(uint32_t ui32Base))ROM_I2CTABLE[43])

#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveIntDisable
#endif

((void (*)(uint32_t ui32Base))ROM_I2CTABLE[44])

#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveIntEnable
#endif

((void (*)(uint32_t ui32Base, uint32_t ui32IntFlags))ROM_I2CTABLE[45])

#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveIntDisableEx
#endif
((void (*)(uint32_t ui32Base,
  
uint32_t ui32IntFlags))ROM_I2CTABLE[47])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveIntEnableEx
  
((void (*)(uint32_t ui32Base,
  
uint32_t ui32IntFlags))ROM_I2CTABLE[48])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveIntStatus
  
((bool (*)(uint32_t ui32Base,
  
bool bMasked))ROM_I2CTABLE[49])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveIntStatusEx
  
((uint32_t (*)(uint32_t ui32Base,
  
bool bMasked))ROM_I2CTABLE[50])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CSlaveStatus
  
((uint32_t (*)(uint32_t ui32Base))ROM_I2CTABLE[51])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2CMasterInitExpClk
  

((void (*)(uint32_t ui32Base,
uint32_t ui32I2CClk,
bool bFast))ROM_I2CTABLE[52])

#if defined(TARGET_IS_CC3200)
#define ROM_SPIEnable
((void (*)(unsigned long ulBase))ROM_SPITABLE[0])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_SPIDisable
((void (*)(unsigned long ulBase))ROM_SPITABLE[1])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_SPIReset
((void (*)(unsigned long ulBase))ROM_SPITABLE[2])
#endif

#define ROM_SPIConfigSetExpClk
((void (*)(unsigned long ulBase,
unsigned long ulSPIClk,
unsigned long ulBitRate,
unsigned long ulMode,
unsigned long ulSubMode,
unsigned long ulConfig))ROM_SPITABLE[3])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPIDataGetNonBlocking
((long (*)(unsigned long ulBase,
unsigned long *pulData))ROM_SPITABLE[4])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPIDataGet
((void (*)(unsigned long ulBase,
unsigned long *pulData))ROM_SPITABLE[5])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPIDataPutNonBlocking
((long (*)(unsigned long ulBase,
unsigned long ulData))ROM_SPITABLE[6])
#endif
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_SPIDataPut
 
((void (*)(unsigned long ulBase,

 unsigned long ulData))ROM_SPITABLE[7])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_SPIFIFOEnable
 
((void (*)(unsigned long ulBase,

 unsigned long ulFlags))ROM_SPITABLE[8])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_SPIFIFODisable
 
((void (*)(unsigned long ulBase,

 unsigned long ulFlags))ROM_SPITABLE[9])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_SPIFIFOLevelSet
 
((void (*)(unsigned long ulBase,

 unsigned long ulTxLevel,

 unsigned long ulRxLevel))ROM_SPITABLE[10])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_SPIFIFOLevelGet
((void (*)(unsigned long ulBase,
  unsigned long *pulTxLevel,
  unsigned long *pulRxLevel))ROM_SPITABLE[11])

#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPIWordCountSet

((void (*)(unsigned long ulBase,
          unsigned long ulWordCount))ROM_SPITABLE[12])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPIIntRegister

((void (*)(unsigned long ulBase,
           void(*pfnHandler)(void)))ROM_SPITABLE[13])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPIIntUnregister

((void (*)(unsigned long ulBase))ROM_SPITABLE[14])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPIIntEnable

((void (*)(unsigned long ulBase,
           unsigned long ulIntFlags))ROM_SPITABLE[15])
#endif


#define ROM_SPIIntDisable
\n((void (*)(unsigned long ulBase,
  unsigned long ulIntFlags))ROM_SPITABLE[16])
#endif
#endif
#define ROM_SPIIntStatus
\n((unsigned long (*)(unsigned long ulBase,
    tBoolean bMasked))ROM_SPITABLE[17])
#endif
#endif
#define ROM_SPIIntClear
\n((void (*)(unsigned long ulBase,
  unsigned long ulIntFlags))ROM_SPITABLE[18])
#endif
#endif
#define ROM_SPIDmaEnable
\n((void (*)(unsigned long ulBase,
  unsigned long ulFlags))ROM_SPITABLE[19])
#endif
#endif
#define ROM_SPIDmaDisable
\n((void (*)(unsigned long ulBase,
unsigned long ulFlags))ROM_SPITABLE[20])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPICSEnable

((void (*)(unsigned long ulBase))ROM_SPITABLE[21])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPICSDisable

((void (*)(unsigned long ulBase))ROM_SPITABLE[22])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SPITransfer

((long (*)(unsigned long ulBase,
  unsigned char *ucDout,
  unsigned char *ucDin,
  unsigned long ulSize,
  unsigned long ulFlags))ROM_SPITABLE[23])
#endif

/******************************************
*s***********************************
//
// Macros for calling ROM functions in the CAM API.
//
#if defined(TARGET_IS_CC3200)
define ROM_CameraReset

((void (*)(unsigned long ulBase))ROM_CAMERATABLE[0])
#endif

#if defined(TARGET_IS_CC3200)
define ROM_CameraParamsConfig

((void (*)(unsigned long ulBase,
 unsigned long ulHSPol,
 unsigned long ulVSPol,
 unsigned long ulFlags))ROM_CAMERATABLE[1])
#endif

#if defined(TARGET_IS_CC3200)
define ROM_CameraXClkConfig

((void (*)(unsigned long ulBase,
 unsigned long ulCamClkIn,
 unsigned long ulXClk))ROM_CAMERATABLE[2])
#endif

#if defined(TARGET_IS_CC3200)
define ROM_CameraXClkSet

((void (*)(unsigned long ulBase,
 unsigned char bXClkFlags))ROM_CAMERATABLE[3])
#endif
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_CameraDMAEnable
\((void (*)(unsigned long ulBase))\text{ROM\_CAMERATABLE}[4])
#endif
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_CameraDMADisable
\((void (*)(unsigned long ulBase))\text{ROM\_CAMERATABLE}[5])
#endif
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_CameraThresholdSet
\((void (*)(unsigned long ulBase,
  
  unsigned long ulThreshold))\text{ROM\_CAMERATABLE}[6])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_CameraIntRegister
\((void (*)(unsigned long ulBase,
  
  void (*pfnHandler)(void)))\text{ROM\_CAMERATABLE}[7])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_CameraIntUnregister
\((void (*)(unsigned long ulBase))\text{ROM\_CAMERATABLE}[8])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_CameraIntEnable
((void (*)(unsigned long ulBase,
    unsigned long ulIntFlags))ROM_CAMERATABLE[9])
#endif
#endif
#define ROM_CameraIntDisable
((void (*)(unsigned long ulBase,
    unsigned long ulIntFlags))ROM_CAMERATABLE[10])
#endif
#endif
#define ROM_CameraIntStatus
((unsigned long (*)(unsigned long ulBase))ROM_CAMERATABLE[11])
#endif
#endif
#define ROM_CameraIntClear
((void (*)(unsigned long ulBase,
    unsigned long ulIntFlags))ROM_CAMERATABLE[12])
#endif
#endif
#define ROM_CameraCaptureStop
((void (*)(unsigned long ulBase,
    tBoolean bImmediate))ROM_CAMERATABLE[13])
#endif
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_CameraCaptureStart
\((void (*)(unsigned long ulBase))ROM_CAMERATABLE[14])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_CameraBufferRead
\((void (*)(unsigned long ulBase, unsigned long *pBuffer,
unsigned char ucSize))ROM_CAMERATABLE[15])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_FlashDisable
\((void (*)(void))ROM_FLASHTABLE[0])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_FlashErase
\((long (*)(unsigned long ulAddress))ROM_FLASHTABLE[1])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_FlashMassErase
\((long (*)(unsigned long ulAddress))ROM_FLASHTABLE[1])
#endif
1097    ((long (*)(void))ROM_FLASHTABLE[2])
1098    #endif
1099    #if defined(TARGET_IS_CC3200)
1100    #define ROM_FlashMassEraseNonBlocking
1101    ((void (*)(void))ROM_FLASHTABLE[3])
1102    #endif
1103    #if defined(TARGET_IS_CC3200)
1104    #define ROM_FlashEraseNonBlocking
1105    ((void (*)(unsigned long ulAddress))ROM_FLASHTABLE[4])
1106    #endif
1107    #if defined(TARGET_IS_CC3200)
1108    #define ROM_FlashProgram
1109    ((long (*)(unsigned long *pulData,
1110        unsigned long ulAddress,
1111        unsigned long ulCount))ROM_FLASHTABLE[5])
1112    #endif
1113    #if defined(TARGET_IS_CC3200)
1114    #define ROM_FlashProgramNonBlocking
1115    ((long (*)(unsigned long *pulData,
1116        unsigned long ulAddress,
1117        unsigned long ulCount))ROM_FLASHTABLE[6])
1118    #endif
1119    #if defined(TARGET_IS_CC3200)
1120    #define ROM_FlashIntRegister
#ifndef ROM_FLASHTABLE

// ROM Flash Table

// #define ROM_FlashIntUnregister

// ROM_FlashIntUnregister

// ROM_FlashIntEnable

// ROM_FlashIntDisable

// ROM_FlashIntStatus

// ROM_FlashIntClear

// ROM_FlashProtectGet

// endif

#endif

#if defined(TARGET_IS_CC3200)

// define ROM_FlashIntUnregister

// (void (*)(void))ROM_FLASHTABLE[7]

#endif

#if defined(TARGET_IS_CC3200)

// define ROM_FlashIntEnable

// (void (*)(unsigned long ulIntFlags))ROM_FLASHTABLE[9]

#endif

#if defined(TARGET_IS_CC3200)

// define ROM_FlashIntDisable

// (void (*)(unsigned long ulIntFlags))ROM_FLASHTABLE[10]

#endif

#if defined(TARGET_IS_CC3200)

// define ROM_FlashIntStatus

// ((unsigned long (*)(tBoolean bMasked))ROM_FLASHTABLE[11])

#endif

#if defined(TARGET_IS_CC3200)

// define ROM_FlashIntClear

// (void (*)(unsigned long ulIntFlags))ROM_FLASHTABLE[12]

#endif

#if defined(TARGET_IS_CC3200)

// define ROM_FlashProtectGet

// (tFlashProtection (*)(unsigned long ulAddress))ROM_FLASHTABLE[13]

#endif
// Macros for calling ROM functions in the Pin API.

#if defined(TARGET_IS_CC3200)
#define ROM_PinModeSet
   ((void (*)(unsigned long ulPin,
         unsigned long ulPinMode))ROM_PINTABLE[0])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PinDirModeSet
   ((void (*)(unsigned long ulPin,
         unsigned long ulPinIO))ROM_PINTABLE[1])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PinDirModeGet
   ((unsigned long (*)(unsigned long ulPin))ROM_PINTABLE[2])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PinModeGet
   ((unsigned long (*)(unsigned long ulPin))ROM_PINTABLE[3])
#endif
#ifdef TARGET_IS_CC3200
#define ROM_PinConfigGet
  ((void (*)(unsigned long ulPin,
             unsigned long *pulPinStrength,
             unsigned long *pulPinType))ROM_PINTABLE[4])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PinConfigSet
  ((void (*)(unsigned long ulPin,
             unsigned long ulPinStrength,
             unsigned long ulPinType))ROM_PINTABLE[5])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PinTypeUART
  ((void (*)(unsigned long ulPin,
             unsigned long ulPinMode))ROM_PINTABLE[6])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PinTypeI2C
  ((void (*)(unsigned long ulPin,
             unsigned long ulPinMode))ROM_PINTABLE[7])
#endif
#if defined(TARGET_IS_CC3200)
  #define ROM_PinTypeSPI
  ((void (*)(unsigned long ulPin,
            unsigned long ulPinMode))ROM_PINTABLE[8])
#endif
#if defined(TARGET_IS_CC3200)
  #define ROM_PinTypeI2S
  ((void (*)(unsigned long ulPin,
            unsigned long ulPinMode))ROM_PINTABLE[9])
#endif
#if defined(TARGET_IS_CC3200)
  #define ROM_PinTypeTimer
  ((void (*)(unsigned long ulPin,
            unsigned long ulPinMode))ROM_PINTABLE[10])
#endif
#if defined(TARGET_IS_CC3200)
  #define ROM_PinTypeCamera
  ((void (*)(unsigned long ulPin,
            unsigned long ulPinMode))ROM_PINTABLE[11])
#endif
#if defined(TARGET_IS_CC3200)
  #define ROM_PinTypeGPIO
  ((void (*)(unsigned long ulPin,
            unsigned long ulPinMode))ROM_PINTABLE[12])
#endif
unsigned long ulPinMode,
\
   tBoolean bOpenDrain))ROM_PINTABLE[12])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PinTypeADC
\
   ((void (*)(unsigned long ulPin,
   
   unsigned long ulPinMode))ROM_PINTABLE[13])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PinTypeSDHost
\
   ((void (*)(unsigned long ulPin,
   
   unsigned long ulPinMode))ROM_PINTABLE[14])
#endif

//******************************************
***********************************
//
// Macros for calling ROM functions in the
SYSTICK API.
//
//******************************************
***********************************
#if defined(TARGET_IS_CC3200)
#define ROM_SysTickEnable
\
   ((void (*)())ROM_SYSTICKTABLE[0])
#endif
#if defined(TARGET_IS_CC3200)
   ((void (*)())ROM_SYSTICKTABLE[0])
#endif
#define ROM_SysTickDisable
\ 
((void (*)(void))ROM_SYSTICKTABLE[1])

#define ROM_SysTickIntRegister
\ 
((void (*)(void (*)(void (*)(void)pnHandler)(void)))ROM_SYSTICKTABLE[2])

#define ROM_SysTickIntUnregister
\ 
((void (*)(void))ROM_SYSTICKTABLE[3])

#define ROM_SysTickIntEnable
\ 
((void (*)(void))ROM_SYSTICKTABLE[4])

#define ROM_SysTickIntDisable
\ 
((void (*)(void))ROM_SYSTICKTABLE[5])

#define ROM_SysTickPeriodSet
\ 
((void (*)(unsigned long ulPeriod))ROM_SYSTICKTABLE[6])

#define ROM_SysTickPeriodGet
// Macros for calling ROM functions in the UTILS API.
//
// Macros for calling ROM functions in the I2S API.
1289  ((void (*)(unsigned long ulBase,
1290       unsigned long ulMode))ROM_I2STABLE[0])
1291 #endif
1292 #if defined(TARGET_IS_CC3200)
1293 #define ROM_I2SDisable
1294  ((void (*)(unsigned long ulBase))ROM_I2STABLE[1])
1295 #endif
1296 #if defined(TARGET_IS_CC3200)
1297 #define ROM_I2SDataPut
1298  ((void (*)(unsigned long ulBase,
1299       unsigned long ulDataLine,
1300       unsigned long ulData))ROM_I2STABLE[2])
1301 #endif
1302 #if defined(TARGET_IS_CC3200)
1303 #define ROM_I2SDataPutNonBlocking
1304  ((long (*)(unsigned long ulBase,
1305       unsigned long ulDataLine,
1306       unsigned long ulData))ROM_I2STABLE[3])
1307 #endif
1308 #if defined(TARGET_IS_CC3200)
1309 #define ROM_I2SDataGet
1310  ((void (*)(unsigned long ulBase,
1311       unsigned long ulDataLine,
unsigned long *pulData))ROM_I2STABLE[4])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SDataGetNonBlocking
((long (*)(unsigned long ulBase,
       unsigned long ulDataLine,
       unsigned long *pulData))ROM_I2STABLE[5])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SConfigSetExpClk
((void (*)(unsigned long ulBase,
       unsigned long ulI2SClk,
       unsigned long ulBitClk,
       unsigned long ulConfig))ROM_I2STABLE[6])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2STxFIFOEnable
((void (*)(unsigned long ulBase,
       unsigned long ulTxLevel,
       unsigned long ulWordsPerTransfer))ROM_I2STABLE[7])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2STxFIFODisable
((void (*)(unsigned long ulBase))ROM_I2STABLE[8])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SRxFIFOEnable
((void (*)(unsigned long ulBase,
        unsigned long ulRxLevel,
        unsigned long ulWordsPerTransfer))ROM_I2STABLE[9])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SRxFIFODisable
((void (*)(unsigned long ulBase))ROM_I2STABLE[10])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2STxFIFOStatusGet
((unsigned long (*)(unsigned long ulBase))ROM_I2STABLE[11])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SRxFIFOStatusGet
((unsigned long (*)(unsigned long ulBase))ROM_I2STABLE[12])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SSerializerConfig
((void (*)(unsigned long ulBase,
unsigned long ulDataLine,
unsigned long ulSerMode,
unsigned long ulInActState))ROM_I2STABLE[13])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SIntEnable
((void(*)(unsigned long ulBase,
unsigned long ulIntFlags))ROM_I2STABLE[14])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SIntDisable
((void(*)(unsigned long ulBase,
unsigned long ulIntFlags))ROM_I2STABLE[15])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SIntStatus
((unsigned long(*)(unsigned long ulBase))ROM_I2STABLE[16])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SIntClear
((void(*)(unsigned long ulBase,
unsigned long ulIntFlags))ROM_I2STABLE[17])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SIntRegister

((void(*)(unsigned long ulBase,
     void (*pfnHandler)(void)))ROM_I2STABLE[18])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_I2SIntUnregister

((void(*)(unsigned long ulBase))ROM_I2STABLE[19])
#endif

//******************************************
***********************************
//
//  Macros for calling ROM functions in the
GPIO API.
//
***********************************
/***********************************

#if defined(TARGET_IS_CC3200)
#define ROM_GPIODirModeSet

((void(*)(unsigned long ulPort,
     unsigned char ucPins,
     unsigned long ulPinIO))ROM_GPIOTABLE[0])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_GPIODirModeGet

((unsigned long (*)(unsigned long ulPort, unsigned char ucPin))ROM_GPIOTABLE[1])

#endif

#if defined(TARGET_IS_CC3200)
#define ROM_GPIOIntTypeSet

((void (*)(unsigned long ulPort, unsigned char ucPins, unsigned long ulIntType))ROM_GPIOTABLE[2])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_GPIODMATriggerEnable

((void (*)(unsigned long ulPort))ROM_GPIOTABLE[3])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_GPIODMATriggerDisable

((void (*)(unsigned long ulPort))ROM_GPIOTABLE[4])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_GPIOIntTypeGet

((unsigned long (*)(unsigned long ulPort, unsigned char ucPin))ROM_GPIOTABLE[5])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_GPIOIntEnable

((void (*)(unsigned long ulPort,
    unsigned long ulIntFlags))ROM_GPIOTABLE[6])

#if defined(TARGET_IS_CC3200)
#define ROM_GPIOIntDisable
    ((void (*)(unsigned long ulPort,
    unsigned long ulIntFlags))ROM_GPIOTABLE[7])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_GPIOIntStatus
    ((long (*)(unsigned long ulPort,
    tBoolean bMasked))ROM_GPIOTABLE[8])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_GPIOIntClear
    ((void (*)(unsigned long ulPort,
    unsigned long ulIntFlags))ROM_GPIOTABLE[9])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_GPIOIntRegister
    ((void (*)(unsigned long ulPort,
    void (*pfnIntHandler)(void)))ROM_GPIOTABLE[10])
#endif


// Macros for calling ROM functions in the AES API.

#if defined(TARGET_IS_CC3200)

#define ROM_AESConfigSet

#endif

//******************************************
// Macros for calling ROM functions in the
// AES API.
//******************************************
((void (*)(uint32_t ui32Base,
   uint32_t ui32Config))ROM_AESTABLE[0])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESKey1Set
   ((void (*)(uint32_t ui32Base,
   uint8_t *pui8Key,
   uint32_t ui32Keysize))ROM_AESTABLE[1])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESKey2Set
   ((void (*)(uint32_t ui32Base,
   uint8_t *pui8Key,
   uint32_t ui32Keysize))ROM_AESTABLE[2])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESKey3Set
   ((void (*)(uint32_t ui32Base,
   uint8_t *pui8Key))ROM_AESTABLE[3])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESIVSet
   ((void (*)(uint32_t ui32Base,
1497   uint8_t *pui8IVdata))ROM_AESTABLE[4])
1498   #endif
1499   #if defined(TARGET_IS_CC3200)
1500   #define ROM_AESTagRead
\  
1501   ((void (*)(uint32_t ui32Base,
1502        uint8_t *pui8TagData))ROM_AESTABLE[5])
1503   #endif
1504   #if defined(TARGET_IS_CC3200)
1505   #define ROM_AESDataLengthSet
\  
1506   ((void (*)(uint32_t ui32Base,
1507        uint64_t ui64Length))ROM_AESTABLE[6])
1508   #endif
1509   #if defined(TARGET_IS_CC3200)
1510   #define ROM_AESAuthDataLengthSet
\  
1511   ((void (*)(uint32_t ui32Base,
1512        uint32_t ui32Length))ROM_AESTABLE[7])
1513   #endif
1514   #if defined(TARGET_IS_CC3200)
1515   #define ROM_AESDataReadNonBlocking
\  
1516   ((bool (*)(uint32_t ui32Base,
1517        uint8_t *pui8Dest,
1518        uint8_t ui8Length))ROM_AESTABLE[8])
#ifndef
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESDataRead
\((\text{void} (*) (\text{uint32}_t \text{ui32Base,})\text{uint8}_t *\text{pui8Dest,})\text{uint8}_t \text{ui8Length}))\text{ROM_AESDataRead}[9])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESDataWriteNonBlocking
\((\text{bool} (*) (\text{uint32}_t \text{ui32Base,})\text{uint8}_t *\text{pui8Src,})\text{uint8}_t \text{ui8Length}))\text{ROM_AESDataWrite}[10])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESDataWrite
\((\text{void} (*) (\text{uint32}_t \text{ui32Base,})\text{uint8}_t *\text{pui8Src,})\text{uint8}_t \text{ui8Length}))\text{ROM_AESDataWrite}[11])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESDataProcess
\((\text{bool} (*) (\text{uint32}_t \text{ui32Base,})\text{uint8}_t *\text{pui8Src,})\text{uint8}_t \text{ui8Length}))\text{ROM_AESDataProcess}[12])
#endif
1542        uint8_t *pui8Dest,
1543        uint32_t
1544            ui32Length))ROM_AESTABLE[12])
1545 #endif
1546 #if defined(TARGET_IS_CC3200)
1547 #define ROM_AESDataMAC
1548           ((bool (*)(uint32_t ui32Base,
1549             uint8_t *pui8Src,
1550             uint32_t ui32Length,
1551             uint8_t *
1552                 *pui8Tag))ROM_AESTABLE[13])
1553 #endif
1554 #if defined(TARGET_IS_CC3200)
1555 #define ROM_AESDataProcessAE
1556           ((bool (*)(uint32_t ui32Base,
1557             uint8_t *pui8Src,
1558             uint8_t *pui8Dest,
1559             uint32_t ui32Length,
1560             uint8_t *
1561                 *pui8AuthSrc,
1562             uint32_t ui32AuthLength,
1563             uint8_t *
1564                 *pui8Tag))ROM_AESTABLE[14])
1565 #endif
1566 #if defined(TARGET_IS_CC3200)
#define ROM_AESIntStatus
((uint32_t (*)(uint32_t ui32Base,
        bool bMasked))ROM_AESTABLE[15])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESIntEnable
((void (*)(uint32_t ui32Base,
        uint32_t ui32IntFlags))ROM_AESTABLE[16])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESIntDisable
((void (*)(uint32_t ui32Base,
        uint32_t ui32IntFlags))ROM_AESTABLE[17])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESIntClear
((void (*)(uint32_t ui32Base,
        uint32_t ui32IntFlags))ROM_AESTABLE[18])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_AESIntRegister
((void (*)(uint32_t ui32Base,
        void(*pfnHandler)
(void))ROM_AESTABLE[19])
1586  #endif
1587  #if defined(TARGET_IS_CC3200)
1588  #define ROM_AESIntUnregister
    \
1589  ((void (*)(uint32_t  
        ui32Base))ROM_AESTABLE[20])
1590  #endif
1591  #if defined(TARGET_IS_CC3200)
1592  #define ROM_AESDMAEnable
    \
1593  ((void (*)(uint32_t ui32Base,
    \
1594       uint32_t 
        ui32Flags))ROM_AESTABLE[21])
1595  #endif
1596  #if defined(TARGET_IS_CC3200)
1597  #define ROM_AESDMADisable
    \
1598  ((void (*)(uint32_t ui32Base,
    \
1599       uint32_t 
        ui32Flags))ROM_AESTABLE[22])
1600  #endif
1600  
1601  //********************************************************************************
1602  //********************************************************************************
1603  //
1604  // Macros for calling ROM functions in the
1605  // DES API.
1606  //
1607  //********************************************************************************
1608  //********************************************************************************
1609  #if defined(TARGET_IS_CC3200)
1610  #define ROM_DESConfigSet
    \
1611  ((void (*)(uint32_t ui32Base,

1610 | uint32_t
   | ui32Config))ROM_DESTABLE[0])
1611 | #endif
1612 | #if defined(TARGET_IS_CC3200)
1613 | #define ROM_DESDataRead
1614 |   ((void (*)(uint32_t ui32Base,
1615 |   uint8_t *pui8Dest,
1616 |   uint8_t
   |   ui8Length))ROM_DESTABLE[1])
1617 | #endif
1618 | #if defined(TARGET_IS_CC3200)
1619 | #define ROM_DESDataReadNonBlocking
1620 |   ((bool (*)(uint32_t ui32Base,
1621 |   uint8_t *pui8Dest,
1622 |   uint8_t
   |   ui8Length))ROM_DESTABLE[2])
1623 | #endif
1624 | #if defined(TARGET_IS_CC3200)
1625 | #define ROM_DESDataProcess
1626 |   ((bool (*)(uint32_t ui32Base,
1627 |   uint8_t *pui8Src,
1628 |   uint8_t *pui8Dest,
1629 |   uint32_t
   |   ui32Length))ROM_DESTABLE[3])
1630 | #endif
1631 | #if defined(TARGET_IS_CC3200)
#define ROM_DESDataWrite
    ((void (*)(uint32_t ui32Base,
        uint8_t *pui8Src,
        uint8_t ui8Length))ROM_DESTABLE[4])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESDataWriteNonBlocking
    ((bool (*)(uint32_t ui32Base,
        uint8_t *pui8Src,
        uint8_t ui8Length))ROM_DESTABLE[5])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESMADisable
    ((void (*)(uint32_t ui32Base,
        uint32_t ui32Flags))ROM_DESTABLE[6])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESDMAEnable
    ((void (*)(uint32_t ui32Base,
        uint32_t ui32Flags))ROM_DESTABLE[7])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESIntClear
((void(*)(uint32_t ui32Base,
   uint32_t ui32IntFlags))ROM_DESTABLE[8])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESIntDisable
((void(*)(uint32_t ui32Base,
   uint32_t ui32IntFlags))ROM_DESTABLE[9])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESIntEnable
((void(*)(uint32_t ui32Base,
   void(*pfnHandler)(void)))ROM_DESTABLE[10])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESIntRegister
((void(*)(uint32_t ui32Base,
   void(*pfnHandler)(void)))ROM_DESTABLE[11])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESIntStatus
((uint32_t(*)(uint32_t ui32Base,
   bool bMasked))ROM_DESTABLE[12])
}
#ifndef
#if defined(TARGET_IS_CC3200)
#define ROM_DESIntUnregister
\((void (*)(uint32_t ui32Base))ROM_DESTABLE[13])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESIVSet
\((bool (*)(uint32_t ui32Base,
\uint8_t *pui8IVdata))ROM_DESTABLE[14])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESKeySet
\((void (*)(uint32_t ui32Base,
\uint8_t *pui8Key))ROM_DESTABLE[15])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_DESDataLengthSet
\((void (*)(uint32_t ui32Base,
\uint32_t ui32Length))ROM_DESTABLE[16])
#endif

//******************************************
***********************************
//	Macros	for
calling	ROM	functions	in	the
SHAMD5	API.
#if defined(TARGET_IS_CC3200)
define ROM_SHAMD5ConfigSet
   ((void (*)(uint32_t ui32Base, uint32_t ui32Mode))ROM_SHAMD5TABLE[0])
#endif

#if defined(TARGET_IS_CC3200)
define ROM_SHAMD5DataProcess
   ((bool (*)(uint32_t ui32Base, uint8_t *pui8DataSrc, uint32_t ui32DataLength, uint8_t *pui8HashResult))ROM_SHAMD5TABLE[1])
#endif

#if defined(TARGET_IS_CC3200)
define ROM_SHAMD5DataWrite
   ((void (*)(uint32_t ui32Base, uint8_t *pui8Src))ROM_SHAMD5TABLE[2])
#endif

#if defined(TARGET_IS_CC3200)
define ROM_SHAMD5DataWriteNonBlocking
   ((bool (*)(uint32_t ui32Base, uint8_t pui8Src))ROM_SHAMD5TABLE[3])
#endif
#define ROM_SHAMD5DMADisable ((void (*)(uint32_t ui32Base))ROM_SHAMD5TABLE[4])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SHAMD5DMAEnable ((void (*)(uint32_t ui32Base))ROM_SHAMD5TABLE[5])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SHAMD5DataLengthSet ((void (*)(uint32_t ui32Base, uint32_t ui32Length))ROM_SHAMD5TABLE[6])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SHAMD5HMACKeySet ((void (*)(uint32_t ui32Base, uint8_t *pui8Src))ROM_SHAMD5TABLE[7])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SHAMD5HMACPPKeyGenerate ((void (*)(uint32_t ui32Base, uint8_t *pui8Key,
uint8_t (*pui8PPKey))ROM_SHAMD5TABLE[8])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SHAMD5HMACPPKeySet
\((void (*)(uint32_t ui32Base,
  
uint8_t (*pui8Src))ROM_SHAMD5TABLE[9])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SHAMD5HMACProcess
\((bool (*)(uint32_t ui32Base,
  
uint8_t *pui8DataSrc,
  
uint32_t ui32DataLength,
  
uint8_t *pui8HashResult))ROM_SHAMD5TABLE[10])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SHAMD5IntClear
\((void (*)(uint32_t ui32Base,
  
uint32_t ui32IntFlags))ROM_SHAMD5TABLE[11])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SHAMD5IntDisable
\((void (*)(uint32_t ui32Base,
  
uint32_t ui32IntFlags))ROM_SHAMD5TABLE[11])
#endif
ui32IntFlags))ROM_SHAMD5TABLE[12])
1770  #endif
1771  #if defined(TARGET_IS_CC3200)
1772  #define ROM_SHAMD5IntEnable
1773  ((void (*)(uint32_t ui32Base,
1774     uint32_t ui32IntFlags))ROM_SHAMD5TABLE[13])
1775  #endif
1776  #if defined(TARGET_IS_CC3200)
1777  #define ROM_SHAMD5IntRegister
1778  ((void (*)(uint32_t ui32Base,
1779     void(*pfnHandler)(void)))ROM_SHAMD5TABLE[14])
1780  #endif
1781  #if defined(TARGET_IS_CC3200)
1782  #define ROM_SHAMD5IntStatus
1783  ((uint32_t (*)(uint32_t ui32Base,
1784     bool bMasked))ROM_SHAMD5TABLE[15])
1785  #endif
1786  #if defined(TARGET_IS_CC3200)
1787  #define ROM_SHAMD5IntUnregister
1788  ((void (*)(uint32_t ui32Base))ROM_SHAMD5TABLE[16])
1789  #endif
1790  #if defined(TARGET_IS_CC3200)
1791  #define ROM_SHAMD5ResultRead
1792  ((void (*)(uint32_t ui32Base,
uint8_t (*pui8Dest))ROM_SHAMD5TABLE[17])

// Macros for calling ROM functions in the CRC API.

#if defined(TARGET_IS_CC3200)
#define ROM_CRCConfigSet
   ((void (*)(uint32_t ui32Base,
      uint32_t ui32CRCConfig))ROM_CRCTABLE[0])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_CRCDataProcess
   ((uint32_t (*)(uint32_t ui32Base,
      void *puiDataIn,
      uint32_t ui32DataLength,
      uint32_t ui32Config))ROM_CRCTABLE[1])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_CRCDataWrite
   ((void (*)(uint32_t ui32Base,
1816    uint32_t ROM_CRCTABLE[2])
1817    #endif
1818    #if defined(TARGET_IS_CC3200)
1819    #define ROM_CRCResultRead
1820    ((uint32_t (*)(uint32_t
    1821    ui32Base))ROM_CRCTABLE[3])
1822    #endif
1823    #if defined(TARGET_IS_CC3200)
1824    #define ROM_CRCSeedSet
1825    ((void (*)(uint32_t	ui32Base,
    1826    uint32_t ui32Seed))ROM_CRCTABLE[4])
1827    #endif
1828    /***********************************************************************
1829    ***********************************************************************
1830    // Macros for calling ROM functions in the
1831    SDHOST API.
1832    /***********************************************************************
1833    #if defined(TARGET_IS_CC3200)
1834    #define ROM_SDHostCmdReset
1835    ((void (*)(unsigned long
    1836    ulBase))ROM_SDHOSTTABLE[0])
1837    #endif
1838    #if defined(TARGET_IS_CC3200)
1839    #define ROM_SDHostInit
1840    ((void (*)(unsigned long

#define ROM_SDHostCmdSend

((long (*)(unsigned long ulBase,
         unsigned long ulCmd,
         unsigned ulArg))ROM_SDHOSTTABLE[2])

#define ROM_SDHostIntRegister

((void (*)(unsigned long ulBase,
           void (*pfnHandler)
           (void)))ROM_SDHOSTTABLE[3])

#define ROM_SDHostIntUnregister

((void (*)(unsigned long
           ulBase))ROM_SDHOSTTABLE[4])

#define ROM_SDHostIntEnable

((void (*)(unsigned long ulBase,
           unsigned long ulIntFlags))ROM_SDHOSTTABLE[5])

#define ROM_SDHostIntDisable
((void (*)(unsigned long ulBase,
  
unsigned long ulIntFlags))ROM_SDHOSTTABLE[6])

#define ROM_SDHostIntStatus

((unsigned long (*)(unsigned long ulBase))ROM_SDHOSTTABLE[7])

#define ROM_SDHostIntClear

((void (*)(unsigned long ulBase,
  
unsigned long ulIntFlags))ROM_SDHOSTTABLE[8])

#define ROM_SDHostRespStatus

((unsigned long (*)(unsigned long ulBase))ROM_SDHOSTTABLE[9])

#define ROM_SDHostRespGet

((void (*)(unsigned long ulBase,
  
unsigned long ulResponse[4]))ROM_SDHOSTTABLE[10])

#define ROM_SDHostBlockSizeSet


1887 | unsigned short
1888 | ulBlkSize))ROM_SDHOSTTABLE[11])
1889 | ifndef
1890 | ifdef TARGET_IS_CC3200
1891 | define ROM_SDHostBlockCountSet
1892 | ((void (*)(unsigned long ulBase,
1893 | unsigned short
1894 | ulBlkCount))ROM_SDHOSTTABLE[12])
1895 | endif
1896 | ifdef TARGET_IS_CC3200
1897 | define ROM_SDHostDataNonBlockingWrite
1898 | ((tBoolean (*)(unsigned long ulBase,
1899 | unsigned long
1900 | ulData))ROM_SDHOSTTABLE[13])
1901 | ifdef TARGET_IS_CC3200
1902 | define ROM_SDHostDataNonBlockingRead
1903 | ((tBoolean (*)(unsigned long ulBase,
1904 | unsigned long
1905 | *pulData))ROM_SDHOSTTABLE[14])
1906 | ifdef TARGET_IS_CC3200
1907 | define ROM_SDHostDataWrite
1908 | ((void (*)(unsigned long ulBase,
1909 | unsigned long
1910 | ulData))ROM_SDHOSTTABLE[15])
1911 | endif
1912 | ifdef TARGET_IS_CC3200
#define ROM_SDHostDataRead
\((void (*)(unsigned long ulBase,
       unsigned long *ulData))ROM_SDHOSTTABLE[16]\)
#endif
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_SDHostSetExpClk
\((void (*)(unsigned long ulBase,
            unsigned long ulSDHostClk,
            unsigned long ulCardClk))ROM_SDHOSTTABLE[17]\)
#endif

//******************************************
// Macros for calling ROM functions in the
// PRCM API.
//******************************************
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMPeripheralClkEnable
\((void (*)(unsigned long ulPeripheral, unsigned long ulClkFlags))ROM_PRCMTABLE[3])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMPeripheralClkDisable
\((void (*)(unsigned long ulPeripheral, unsigned long ulClkFlags))ROM_PRCMTABLE[4])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMPeripheralReset
\((void (*)(unsigned long ulPeripheral))ROM_PRCMTABLE[5])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMPeripheralStatusGet
\((tBoolean (*)(unsigned long ulPeripheral))ROM_PRCMTABLE[6])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMI2SClockFreqSet
\((void (*)(unsigned long ulI2CClkFreq))ROM_PRCMTABLE[7])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMPeripheralClockGet
1958  ((unsigned long (*)(unsigned long
     ulPeripheral))ROM_PRCMTABLE[8])
1959  #endif
1960  #if defined(TARGET_IS_CC3200)
1961  #define ROM_PRCMSleepEnter
1962  ((void (*)(void))ROM_PRCMTABLE[9])
1963  #endif
1964  #if defined(TARGET_IS_CC3200)
1965  #define ROM_PRCMDepSleepEnter
1966  ((void (*)(void))ROM_PRCMTABLE[10])
1967  #endif
1968  #if defined(TARGET_IS_CC3200)
1969  #define ROM_PRCMSRAMRetentionEnable
1970  ((void (*)(unsigned long
     ulSramColSel,
     unsigned long
     ulFlags))ROM_PRCMTABLE[11])
1971  #endif
1972  #if defined(TARGET_IS_CC3200)
1973  #define ROM_PRCMSRAMRetentionDisable
1974  ((void (*)(unsigned long
     ulSramColSel,
     unsigned long
     ulFlags))ROM_PRCMTABLE[12])
1975  #endif
1976  #if defined(TARGET_IS_CC3200)
1977  #define ROM_PRCMLPDEnter
1978  ((void (*)(void))ROM_PRCMTABLE[13])
1979  #endif
1980  #if defined(TARGET_IS_CC3200)
1981  #define ROM_PRCMLPDISIntervalSet
1984 \((\text{void} (*)\text{(unsigned long)}\text{ulTicks}))\text{ROM_PRCMTABLE}[14])
1985 \#endif
1986 \#if \text{defined(TARGET_IS_CC3200)}
1987 \#define \text{ROM_PRCMLPDSWakeupSourceEnable}
1988 \((\text{void} (*)\text{(unsigned long)}\text{ulLpdsWakeupSrc}))\text{ROM_PRCMTABLE}[15])
1989 \#endif
1990 \#if \text{defined(TARGET_IS_CC3200)}
1991 \#define \text{ROM_PRCMLPDSWakeupCauseGet}
1992 \((\text{unsigned long} (*)\text{(void)}))\text{ROM_PRCMTABLE}[16])
1993 \#endif
1994 \#if \text{defined(TARGET_IS_CC3200)}
1995 \#define \text{ROM_PRCMLPDSWakeupGPIOSelect}
1996 \((\text{void} *)\text{(unsigned long ulGPIOPin,)}\text{ulType}))\text{ROM_PRCMTABLE}[17])
1998 \#endif
1999 \#if \text{defined(TARGET_IS_CC3200)}
2000 \#define \text{ROM_PRCMLPDSWakeupSourceDisable}
2001 \((\text{void} *)\text{(unsigned long ulLpdsWakeupSrc)})\text{ROM_PRCMTABLE}[18])
2002 \#endif
2003 \#if \text{defined(TARGET_IS_CC3200)}
2004 \#define \text{ROM_PRCMHibernateEnter}
2005 \((\text{void} *)\text{(void)}))\text{ROM_PRCMTABLE}[19])
2006 \#endif
2007 \#if \text{defined(TARGET_IS_CC3200)}
2008 \#define \text{ROM_PRCMHibernateWakeupSourceEnable}
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMHibernateWakeupCauseGet
((unsigned long (*)(void))ROM_PRCMTABLE[20])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMHibernateWakeUpGPIOSelect
((void (*)(unsigned long ulMultiGPIOBitMap, unsigned long ulType))ROM_PRCMTABLE[21])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMHibernateWakeupSourceDisable
((void (*)(unsigned long ulHIBWakupSrc))ROM_PRCMTABLE[22])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMHibernateIntervalSet
((void (*)(unsigned long long ullTicks))ROM_PRCMTABLE[23])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMSlowClkCtrGet
((unsigned long long (*)(void))ROM_PRCMTABLE[24])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMHibernateWakeupCauseGet
((unsigned long (*)(void))ROM_PRCMTABLE[21])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMHibernateWakeUpGPIOSelect
((void (*)(unsigned long ulMultiGPIOBitMap, unsigned long ulType))ROM_PRCMTABLE[22])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMHibernateWakeupSourceDisable
((void (*)(unsigned long ulHIBWakupSrc))ROM_PRCMTABLE[23])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMHibernateIntervalSet
((void (*)(unsigned long long ullTicks))ROM_PRCMTABLE[24])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMSlowClkCtrGet
((unsigned long long (*)(void))ROM_PRCMTABLE[25])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMSlowClkCtrMatchSet
\((\text{void} (*)(\text{unsigned long long}
\text{ullTicks}))\text{ROM_PRCMTABLE}[26])

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMSlowClkCtrMatchGet
\((\text{unsigned long long} (*)
(\text{void}))\text{ROM_PRCMTABLE}[27])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMOCRRegisterWrite
\((\text{void} (*)(\text{unsigned char ucIndex},
\text{unsigned long ulRegValue}))\text{ROM_PRCMTABLE}[28])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMOCRRegisterRead
\((\text{unsigned long} (*)(\text{unsigned char ucIndex}))\text{ROM_PRCMTABLE}[29])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMIntRegister
\((\text{void} (*)(\text{void} (*pfnHandler)
(\text{void}))\text{ROM_PRCMTABLE}[30])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMIntUnregister
\((\text{void} (*)(\text{void}))\text{ROM_PRCMTABLE}[31])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMIntEnable
\((void (*)(unsigned long ulIntFlags))ROM_PRCMTABLE[32])

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMIntDisable
\((void (*)(unsigned long ulIntFlags))ROM_PRCMTABLE[33])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMIntStatus
\((unsigned long (*)(void))ROM_PRCMTABLE[34])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMRTCInUseSet
\((void (*)(void))ROM_PRCMTABLE[35])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMRTCInUseGet
\((tBoolean (*)(void))ROM_PRCMTABLE[36])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMRTCSet
\((void (*)(unsigned long ulSecs, unsigned short usMsec))ROM_PRCMTABLE[37])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_PRCMRTCTCGet
\n((void (*)(unsigned long *ulSecs,
\nunsigned short *usMsec))ROM_PRCMTABLE[38])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMRTCMatchSet
\n((void (*)(unsigned long ulSecs,
\nunsigned short usMsec))ROM_PRCMTABLE[39])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMRTCMatchGet
\n((void (*)(unsigned long *ulSecs,
\nunsigned short *usMsec))ROM_PRCMTABLE[40])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_PRCMLPDSRestoreInfoSet
\n((void (*)(unsigned long ulRestoreSP,
\nunsigned long ulRestorePC))ROM_PRCMTABLE[41])
#endif

/******************************************
******************************************
//
// Macros for calling ROM functions in the HWSPINLOCK API.
//******************************************
//******************************************

#if defined(TARGET_IS_CC3200)
#define ROM_HwSpinLockAcquire
\
    ((void (*)(uint32_t ui32LockID))ROM_HWSPINLOCKTABLE[0])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_HwSpinLockTryAcquire
\
    ((int32_t (*)(uint32_t ui32LockID,
\    uint32_t ui32Retry))ROM_HWSPINLOCKTABLE[1])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_HwSpinLockRelease
\
    ((void (*)(uint32_t ui32LockID))ROM_HWSPINLOCKTABLE[2])
#endif

#if defined(TARGET_IS_CC3200)
#define ROM_HwSpinLockTest
\
    ((uint32_t (*)(uint32_t ui32LockID,
\    bool bCurrentStatus))ROM_HWSPINLOCKTABLE[3])
#endif

//******************************************
//******************************************

// Macros for calling ROM functions in the ADC API.
#if defined(TARGET_IS_CC3200)
  #define ROM_ADCEnable
  ((void(*)(unsigned long ulBase))ROM_ADCTABLE[0])
#endif

#if defined(TARGET_IS_CC3200)
  #define ROM_ADCDisable
  ((void(*)(unsigned long ulBase))ROM_ADCTABLE[1])
#endif

#if defined(TARGET_IS_CC3200)
  #define ROM_ADCChannelEnable
  ((void(*)(unsigned long ulBase, unsigned long ulChannel))ROM_ADCTABLE[2])
#endif

#if defined(TARGET_IS_CC3200)
  #define ROM_ADCChannelDisable
  ((void(*)(unsigned long ulBase, unsigned long ulChannel))ROM_ADCTABLE[3])
#endif

#if defined(TARGET_IS_CC3200)
  #define ROM_ADCIntRegister
  ((void(*)(unsigned long ulBase, unsigned long ulChannel, 

  unsigned long ulChannel))ROM_ADCTABLE[4])
#endif

#if defined(TARGET_IS_CC3200)
  #define ROM_ADCChannelDisable
  ((void(*)(unsigned long ulBase, unsigned long ulChannel, 

  unsigned long ulChannel))ROM_ADCTABLE[5])
#endif
void (*pfnHandler)(void))ROM_ADCTABLE[4])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCIntUnregister
    ((void (*)(unsigned long ulBase, unsigned long ulChannel))ROM_ADCTABLE[5])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCIntEnable
    ((void (*)(unsigned long ulBase, unsigned long ulChannel, unsigned long ulIntFlags))ROM_ADCTABLE[6])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCIntDisable
    ((void (*)(unsigned long ulBase, unsigned long ulChannel, unsigned long ulIntFlags))ROM_ADCTABLE[7])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCIntStatus
    ((unsigned long (*)(unsigned long ulBase, unsigned long ulChannel, unsigned long ulIntFlags))ROM_ADCTABLE[8])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCIntRearrange
    ((void (*)(unsigned long ulBase, unsigned long ulChannel, unsigned long ulIntFlags))ROM_ADCTABLE[9])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCIntEnablePin
    ((void (*)(unsigned long ulBase, unsigned long ulChannel, unsigned long ulIntFlags))ROM_ADCTABLE[10])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCIntDisablePin
    ((void (*)(unsigned long ulBase, unsigned long ulChannel, unsigned long ulIntFlags))ROM_ADCTABLE[11])
#endif
unsigned long ulChannel))ROM_ADCTABLE[8])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCIntClear

(void (*)(unsigned long ulBase,
unsigned long ulChannel,
unsigned long ulIntFlags))ROM_ADCTABLE[9])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCDMAEnable

(void (*)(unsigned long ulBase,
unsigned long ulChannel))ROM_ADCTABLE[10])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCDMADisable

(void (*)(unsigned long ulBase,
unsigned long ulChannel))ROM_ADCTABLE[11])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCChannelGainSet

(void (*)(unsigned long ulBase,
unsigned long ulChannel, unsigned char
ucGain))ROM_ADCTABLE[12])
2199 #endif
2200 #if defined(TARGET_IS_CC3200)
2201 #define ROM_ADCChannleGainGet
2202 ((unsigned char (*)(unsigned long
2203 ulBase,
2204 unsigned long
2205 ulChannel))ROM_ADCTABLE[13])
2204 #endif
2205 #if defined(TARGET_IS_CC3200)
2206 #define ROM_ADCTimerConfig
2207 ((void (*)(unsigned long ulBase,
2208 unsigned long ulValue))ROM_ADCTABLE[14])
2209 #endif
2210 #if defined(TARGET_IS_CC3200)
2211 #define ROM_ADCTimerEnable
2212 ((void (*)(unsigned long ulBase))ROM_ADCTABLE[15])
2213 #endif
2214 #if defined(TARGET_IS_CC3200)
2215 #define ROM_ADCTimerDisable
2216 ((void (*)(unsigned long ulBase))ROM_ADCTABLE[16])
2217 #endif
2218 #if defined(TARGET_IS_CC3200)
2219 #define ROM_ADCTimerReset
2220 ((void (*)(unsigned long ulBase))ROM_ADCTABLE[17])
2221 #endif
2222 #if defined(TARGET_IS_CC3200)
#define ROM_ADCTimerValueGet
\((\text{unsigned long }*)(\text{unsigned long }ulBase))\text{ROM_ADCTABLE}[18])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCFIFOLvlGet
\((\text{unsigned char }*)(\text{unsigned long }ulBase, \text{unsigned long }ulChannel))\text{ROM_ADCTABLE}[19])
#endif
#if defined(TARGET_IS_CC3200)
#define ROM_ADCFIFORead
\((\text{unsigned long }*)(\text{unsigned long }ulBase, \text{unsigned long }ulChannel))\text{ROM_ADCTABLE}[20])
#endif
#endif // __ROM_H__

//******************************************
//******************************************
//		rom_map.h
//		Macros to facilitate calling functions in the ROM when they are available and in flash otherwise.
//
// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
//
// Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
//
// Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

//
//****************************************************************************
//*****************************************************************************
//
// THIS IS AN AUTO-GENERATED FILE. DO NOT EDIT BY HAND.
//
//****************************************************************************
//
#ifndef __ROM_MAP_H__
define __ROM_MAP_H__
#include "rom_patch.h"
#endif

// Macros for the Interrupt API.
//
//****************************************************************************
#endif
 ifndef ROM_IntEnable
#define MAP_IntEnable \  ROM_IntEnable
#else
#define MAP_IntEnable \  IntEnable
#endif
#ifndef ROM_IntMasterEnable
#define MAP_IntMasterEnable \  ROM_IntMasterEnable
#else
#define MAP_IntMasterEnable \  IntEnable
#endif
#ifndef ROM_IntMasterDisable
#define MAP_IntMasterDisable \ 
  ROM_IntMasterDisable
#else
#define MAP_IntMasterDisable \ 
  IntMasterDisable
#endif

#ifndef ROM_IntDisable
#define MAP_IntDisable \ 
  ROM_IntDisable
#else
#define MAP_IntDisable \ 
  IntDisable
#endif

#ifndef ROM_IntPriorityGroupingSet
#define MAP_IntPriorityGroupingSet \ 
  ROM_IntPriorityGroupingSet
#else
#define MAP_IntPriorityGroupingSet \ 
  IntPriorityGroupingSet
#endif

#ifndef ROM_IntPriorityGroupingGet
#define MAP_IntPriorityGroupingGet \ 
  ROM_IntPriorityGroupingGet
#else
#define MAP_IntPriorityGroupingGet \ 
  IntPriorityGroupingGet
#endif

#ifndef ROM_IntPrioritySet
#define MAP_IntPrioritySet \ 
  ROM_IntPrioritySet
#else
#define MAP_IntPrioritySet \ 
  IntPrioritySet
#endif
#ifdef ROM_IntPriorityGet
#define MAP_IntPriorityGet \    ROM_IntPriorityGet
#else
#define MAP_IntPriorityGet \    IntPriorityGet
#endif
#endif
#endif
#ifndef ROM_IntPendClear
#define MAP_IntPendClear \    IntPendClear
#endif
#endif
#endif
#ifndef ROM_IntPriorityMaskGet
#define MAP_IntPriorityMaskGet \    IntPriorityMaskGet
#endif
#endif
#endif
#endif
#endif
#endif
#ifndef ROM_IntRegister
#define MAP_IntRegister \
ROM_IntRegister

#define MAP_IntRegister \ 
    IntRegister
#endif

#ifdef ROM_IntUnregister
#define MAP_IntUnregister \ 
    ROM_IntUnregister
#else
#define MAP_IntUnregister \ 
    IntUnregister
#endif

#ifdef ROM_IntVTableBaseSet
#define MAP_IntVTableBaseSet \ 
    ROM_IntVTableBaseSet
#else
#define MAP_IntVTableBaseSet \ 
    IntVTableBaseSet
#endif

//******************************************
//
// Macros for the Timer API.
//
//******************************************

#ifdef ROM_TimerEnable
#define MAP_TimerEnable \ 
    ROM_TimerEnable
#else
#define MAP_TimerEnable \ 
    TimerEnable
#endif

#ifdef ROM_TimerDisable
#define MAP_TimerDisable \ 
    ROM_TimerDisable
#else
#define MAP_TimerDisable \ TimerDisable
#endif

#ifdef ROM_TimerConfigure
#define MAP_TimerConfigure \ ROM_TimerConfigure
#else
#define MAP_TimerConfigure \ TimerConfigure
#endif

#ifdef ROM_TimerControlLevel
#define MAP_TimerControlLevel \ ROM_TimerControlLevel
#else
#define MAP_TimerControlLevel \ TimerControlLevel
#endif

#ifdef ROM_TimerControlEvent
#define MAP_TimerControlEvent \ ROM_TimerControlEvent
#else
#define MAP_TimerControlEvent \ TimerControlEvent
#endif

#ifdef ROM_TimerControlStall
#define MAP_TimerControlStall \ ROM_TimerControlStall
#else
#define MAP_TimerControlStall \ TimerControlStall
#endif

#ifdef ROM_TimerPrescaleSet
#define MAP_TimerPrescaleSet \ ROM_TimerPrescaleSet
#else
#define MAP_TimerPrescaleSet \ TimerPrescaleSet
#endif
#ifndef ROM_TimerPrescaleGet
#define MAP_TimerPrescaleGet \  
    ROM_TimerPrescaleGet
#else
#define MAP_TimerPrescaleGet \  
    TimerPrescaleGet
#endif

#ifdef ROM_TimerPrescaleMatchSet
#define MAP_TimerPrescaleMatchSet \  
    ROM_TimerPrescaleMatchSet
#else
#define MAP_TimerPrescaleMatchSet \  
    TimerPrescaleMatchSet
#endif

#ifdef ROM_TimerPrescaleMatchGet
#define MAP_TimerPrescaleMatchGet \  
    ROM_TimerPrescaleMatchGet
#else
#define MAP_TimerPrescaleMatchGet \  
    TimerPrescaleMatchGet
#endif

#ifdef ROM_TimerLoadSet
#define MAP_TimerLoadSet \  
    ROM_TimerLoadSet
#else
#define MAP_TimerLoadSet \  
    TimerLoadSet
#endif

#ifdef ROM_TimerLoadGet
#define MAP_TimerLoadGet \  
    ROM_TimerLoadGet
#else
#define MAP_TimerLoadGet \  
    TimerLoadGet
#endif
#ifdef ROM_TimerValueGet
#define MAP_TimerValueGet \ 
ROM_TimerValueGet
#else
#define MAP_TimerValueGet \ 
TimerValueGet
#endif

#ifdef ROM_TimerMatchSet
#define MAP_TimerMatchSet \ 
ROM_TimerMatchSet
#else
#define MAP_TimerMatchSet \ 
TimerMatchSet
#endif

#ifdef ROM_TimerMatchGet
#define MAP_TimerMatchGet \ 
ROM_TimerMatchGet
#else
#define MAP_TimerMatchGet \ 
TimerMatchGet
#endif

#ifdef ROM_TimerIntRegister
#define MAP_TimerIntRegister \ 
ROM_TimerIntRegister
#else
#define MAP_TimerIntRegister \ 
TimerIntRegister
#endif

#ifdef ROM_TimerIntUnregister
#define MAP_TimerIntUnregister \ 
ROM_TimerIntUnregister
#else
#define MAP_TimerIntUnregister \ 
TimerIntUnregister
#endif

#ifdef ROM_TimerIntEnable
#ifndef ROM_TimerIntEnable
#define MAP_TimerIntEnable
    TimerIntEnable
#endif

#ifdef ROM_TimerIntDisable
#define MAP_TimerIntDisable
    TimerIntDisable
#else
#define MAP_TimerIntDisable
    TimerIntDisable
#endif

#ifdef ROM_TimerIntStatus
#define MAP_TimerIntStatus
    TimerIntStatus
#else
#define MAP_TimerIntStatus
    TimerIntStatus
#endif

#ifdef ROM_TimerIntClear
#define MAP_TimerIntClear
    TimerIntClear
#else
#define MAP_TimerIntClear
    TimerIntClear
#endif

#ifdef ROM_TimerDMAEventSet
#define MAP_TimerDMAEventSet
    TimerDMAEventSet
#else
#define MAP_TimerDMAEventSet
    TimerDMAEventSet
#endif

#ifdef ROM_TimerDMAEventGet
#define MAP_TimerDMAEventGet
    TimerDMAEventGet
#else
#define MAP_TimerDMAEventGet
    TimerDMAEventGet
#endif

#else
#define MAP_TimerIntEnable
    TimerIntEnable
#define MAP_TimerIntDisable
    TimerIntDisable
#define MAP_TimerIntStatus
    TimerIntStatus
#define MAP_TimerIntClear
    TimerIntClear
#define MAP_TimerDMAEventSet
    TimerDMAEventSet
#define MAP_TimerDMAEventGet
    TimerDMAEventGet
#endif
#define MAP_TimerDMAEventGet \  TimerDMAEventGet
#endif

//******************************************
//
// Macros for the UART API.
//
//******************************************

#ifdef ROM_UARTParityModeSet
#define MAP_UARTParityModeSet \  ROM_UARTParityModeSet
#else
#define MAP_UARTParityModeSet \  UARTParityModeSet
#endif

#ifdef ROM_UARTParityModeGet
#define MAP_UARTParityModeGet \  ROM_UARTParityModeGet
#else
#define MAP_UARTParityModeGet \  UARTParityModeGet
#endif

#ifdef ROM_UARTFIFOLevelSet
#define MAP_UARTFIFOLevelSet \  ROM_UARTFIFOLevelSet
#else
#define MAP_UARTFIFOLevelSet \  UARTFIFOLevelSet
#endif

#ifdef ROM_UARTFIFOLevelGet
#define MAP_UARTFIFOLevelGet \  ROM_UARTFIFOLevelGet
#else
#define MAP_UARTFIFOLevelGet \  UARTFIFOLevelGet
#endif
UARTFIFOLevelGet

#ifdef ROM_UARTConfigSetExpClk
#define MAP_UARTConfigSetExpClk \  ROM_UARTConfigSetExpClk
#else
#define MAP_UARTConfigSetExpClk \  UARTConfigSetExpClk
#endif

#ifdef ROM_UARTConfigGetExpClk
#define MAP_UARTConfigGetExpClk \  ROM_UARTConfigGetExpClk
#else
#define MAP_UARTConfigGetExpClk \  UARTConfigGetExpClk
#endif

#ifdef ROM_UARTEnable
#define MAP_UARTEnable \  ROM_UARTEnable
#else
#define MAP_UARTEnable \  UARTEnable
#endif

#ifdef ROM_UARTDisable
#define MAP_UARTDisable \  ROM_UARTDisable
#else
#define MAP_UARTDisable \  UARTDisable
#endif

#ifdef ROM_UARTFIFOEnable
#define MAP_UARTFIFOEnable \  ROM_UARTFIFOEnable
#else
#define MAP_UARTFIFOEnable \  UARTFIFOEnable
#endif
397  #ifdef ROM_UARTFIFODisable
398  #define MAP_UARTFIFODisable \
            ROM_UARTFIFODisable
399  #else
400  #define MAP_UARTFIFODisable \
            UARTFIFODisable
401  #endif
402  #ifdef ROM_UARTCharsAvail
403  #define MAP_UARTCharsAvail \
            ROM_UARTCharsAvail
404  #else
405  #define MAP_UARTCharsAvail \
            UARTCharsAvail
406  #endif
407  #ifdef ROM_UARTSpaceAvail
408  #define MAP_UARTSpaceAvail \
            UARTSpaceAvail
409  #endif
410  #ifdef ROM_UARTCharGetNonBlocking
411  #define MAP_UARTCharGetNonBlocking \
            ROM_UARTCharGetNonBlocking
412  #else
413  #define MAP_UARTCharGetNonBlocking \
            UARTCharGetNonBlocking
414  #endif
415  #ifdef ROM_UARTCharGet
416  #define MAP_UARTCharGet \
            UARTCharGet
417  #endif
418  #ifdef ROM_UARTCharPutNonBlocking
419  #define MAP_UARTCharPutNonBlocking \
            UARTCharPutNonBlocking
420  #endif
#ifdef ROM_UARTCharPutNonBlocking
#define MAP_UARTCharPutNonBlocking UARTCharPutNonBlocking
#endif

#ifdef ROM_UARTCharPut
#define MAP_UARTCharPut UARTCharPut
#else
#define MAP_UARTCharPut ROM_UARTCharPut
#endif

#ifdef ROM_UARTBreakCtl
#define MAP_UARTBreakCtl UARTBreakCtl
#else
#define MAP_UARTBreakCtl ROM_UARTBreakCtl
#endif

#ifdef ROM_UARTBusy
#define MAP_UARTBusy UARTBusy
#else
#define MAP_UARTBusy ROM_UARTBusy
#endif

#ifdef ROM_UARTIntRegister
#define MAP_UARTIntRegister UARTIntRegister
#else
#define MAP_UARTIntRegister ROM_UARTIntRegister
#endif

#ifdef ROM_UARTIntUnregister
#define MAP_UARTIntUnregister UARTIntRegister
#else
#define MAP_UARTIntUnregister ROM_UARTIntUnregister
#endif
#define MAP_UARTIntUnregister UARTIntUnregister
#endif

#ifdef ROM_UARTIntEnable
#define MAP_UARTIntEnable ROM_UARTIntEnable
#else
#define MAP_UARTIntEnable UARTIntEnable
#endif

#ifdef ROM_UARTIntDisable
#define MAP_UARTIntDisable ROM_UARTIntDisable
#else
#define MAP_UARTIntDisable UARTIntDisable
#endif

#ifdef ROM_UARTIntStatus
#define MAP_UARTIntStatus ROM_UARTIntStatus
#else
#define MAP_UARTIntStatus UARTIntStatus
#endif

#ifdef ROM_UARTIntClear
#define MAP_UARTIntClear ROM_UARTIntClear
#else
#define MAP_UARTIntClear UARTIntClear
#endif

#ifdef ROM_UARTDMAEnable
#define MAP_UARTDMAEnable ROM_UARTDMAEnable
#else
#define MAP_UARTDMAEnable UARTDMAEnable
#endif
#endif

#ifdef ROM_UARTDMADisable
#define MAP_UARTDMADisable \ 
    ROM_UARTDMADisable
#else
#define MAP_UARTDMADisable \ 
    UARTDMADisable
#endif

#ifdef ROM_UARTRxErrorGet
#define MAP_UARTRxErrorGet \ 
    ROM_UARTRxErrorGet
#else
#define MAP_UARTRxErrorGet \ 
    UARTRxErrorGet
#endif

#ifdef ROM_UARTRxErrorClear
#define MAP_UARTRxErrorClear \ 
    ROM_UARTRxErrorClear
#else
#define MAP_UARTRxErrorClear \ 
    UARTRxErrorClear
#endif

#ifdef ROM_UARTModemControlSet
#define MAP_UARTModemControlSet \ 
    ROM_UARTModemControlSet
#else
#define MAP_UARTModemControlSet \ 
    UARTModemControlSet
#endif

#ifdef ROM_UARTModemControlClear
#define MAP_UARTModemControlClear \ 
    ROM_UARTModemControlClear
#else
#define MAP_UARTModemControlClear \ 
    UARTModemControlClear
#endif

#ifdef ROM_UARTModemControlGet
#define MAP_UARTModemControlGet \  
    ROM_UARTModemControlGet
#define MAP_UARTModemStatusGet \  
    UARTModemControl
#define MAP_UARTFlowControlSet \  
    UARTFlowControl
#define MAP_UARTFlowControlGet \  
    UARTFlowControl
#define MAP_UARTTxIntModeSet \  
    UARTTxIntMode
#define MAP_UARTTxIntModeGet \  
    UARTTxIntMode

#if defined ROM_UARTModemStatusGet
#define MAP_UARTModemStatusGet \  
    UARTModemStatusGet
#endif
#if defined ROM_UARTFlowControlSet
#define MAP_UARTFlowControlSet \  
    UARTFlowControlSet
#endif
#if defined ROM_UARTFlowControlGet
#define MAP_UARTFlowControlGet \  
    UARTFlowControlGet
#endif
#if defined ROM_UARTTxIntModeSet
#define MAP_UARTTxIntModeSet \  
    UARTTxIntModeSet
#endif
#if defined ROM_UARTTxIntModeGet
#define MAP_UARTTxIntModeGet \  
    UARTTxIntModeGet
#endif
582 #else
583 #define MAP_UARTTxIntModeGet \
584 UARTTxIntModeGet
585 #endif
586
587 //********************************************************************************
588 // Macros for the uDMA API.
589 //********************************************************************************
590 #ifdef ROM_uDMAChannelTransferSet
591 #define MAP_uDMAChannelTransferSet \
592 ROM_uDMAChannelTransferSet
593 #else
594 #define MAP_uDMAChannelTransferSet \
595 uDMAChannelTransferSet
596 #endif
597
598 #ifdef ROM_uDMAEnable
599 #define MAP_uDMAEnable \
600 ROM_uDMAEnable
601 #else
602 #define MAP_uDMAEnable \
603 uDMAEnable
604 #endif
605
606 #ifdef ROM_uDMADisable
607 #define MAP_uDMADisable \
608 ROM_uDMADisable
609 #else
610 #define MAP_uDMADisable \
611 uDMADisable
612 #endif
613
614 #ifdef ROM_uDMAErrorStatusGet
615 #define MAP_uDMAErrorStatusGet \
616 ROM_uDMAErrorStatusGet
617 #else
618 #endif
#define MAP_uDMAErrorStatusGet \
uDMAErrorStatusGet
#endif

#ifdef ROM_uDMAErrorStatusClear
#define MAP_uDMAErrorStatusClear \
ROM_uDMAErrorStatusClear
#else
#define MAP_uDMAErrorStatusClear \
uDMAErrorStatusClear
#endif

#ifdef ROM_uDMAChannelEnable
#define MAP_uDMAChannelEnable \
ROM_uDMAChannelEnable
#else
#define MAP_uDMAChannelEnable \
uDMAChannelEnable
#endif

#ifdef ROM_uDMAChannelDisable
#define MAP_uDMAChannelDisable \
ROM_uDMAChannelDisable
#else
#define MAP_uDMAChannelDisable \
uDMAChannelDisable
#endif

#ifdef ROM_uDMAChannelIsEnabled
#define MAP_uDMAChannelIsEnabled \
ROM_uDMAChannelIsEnabled
#else
#define MAP_uDMAChannelIsEnabled \
uDMAChannelIsEnabled
#endif

#ifdef ROM_uDMAControlBaseSet
#define MAP_uDMAControlBaseSet \
ROM_uDMAControlBaseSet
#else
#define MAP_uDMAControlBaseSet \
uDMAControlBaseSet
#endif
#define MAP_uDMAChannelControlSet \ 
   ROM_uDMAChannelControlSet
#else
#define MAP_uDMAChannelControlSet \ 
   uDMAChannelControlSet
#endif
#ifdef ROM_uDMAChannelSizeGet
#define MAP_uDMAChannelSizeGet \ 
   ROM_uDMAChannelSizeGet
#else
#define MAP_uDMAChannelSizeGet \ 
   uDMAChannelSizeGet
#endif
#ifdef ROM_uDMAChannelModeGet
#define MAP_uDMAChannelModeGet \ 
   ROM_uDMAChannelModeGet
#else
#define MAP_uDMAChannelModeGet \ 
   uDMAChannelModeGet
#endif
#ifdef ROM_uDMAIntStatus
#define MAP_uDMAIntStatus \ 
   ROM_uDMAIntStatus
#else
#define MAP_uDMAIntStatus \ 
   uDMAIntStatus
#endif
#ifdef ROM_uDMAIntClear
#define MAP_uDMAIntClear \ 
   ROM_uDMAIntClear
#else
#define MAP_uDMAIntClear \ 
   uDMAIntClear
#endif
#ifdef ROM_uDMAControlAlternateBaseGet
#define MAP_uDMAControlAlternateBaseGet \ 
   ROM_uDMAControlAlternateBaseGet
#else
#else
#define MAP_uDMAControlAlternateBaseGet \ uDMAControlAlternateBaseGet
#endif

#ifdef ROM_uDMAChannelScatterGatherSet
#define MAP_uDMAChannelScatterGatherSet \ ROM_uDMAChannelScatterGatherSet
#else
#define MAP_uDMAChannelScatterGatherSet \ uDMAChannelScatterGatherSet
#endif

#ifdef ROM_uDMAChannelAssign
#define MAP_uDMAChannelAssign \ ROM_uDMAChannelAssign
#else
#define MAP_uDMAChannelAssign \ uDMAChannelAssign
#endif

#ifdef ROM_uDMAIntRegister
#define MAP_uDMAIntRegister \ ROM_uDMAIntRegister
#else
#define MAP_uDMAIntRegister \ uDMAIntRegister
#endif

#ifdef ROM_uDMAIntUnregister
#define MAP_uDMAIntUnregister \ ROM_uDMAIntUnregister
#else
#define MAP_uDMAIntUnregister \ uDMAIntUnregister
#endif

//******************************************
//**************************************************
//	Macros	for	the	Watchdog	API.
//******************************************
***********************************
#ifdef	ROM_WatchdogIntClear
#define	MAP_WatchdogIntClear	\
ROM_WatchdogIntClear
#else
#define	MAP_WatchdogIntClear	\
WatchdogIntClear
#endif
#ifdef	ROM_WatchdogRunning
#define	MAP_WatchdogRunning	\
ROM_WatchdogRunning
#else
#define	MAP_WatchdogRunning	\
WatchdogRunning
#endif
#ifdef	ROM_WatchdogEnable
#define	MAP_WatchdogEnable	\
ROM_WatchdogEnable
#else
#define	MAP_WatchdogEnable	\
WatchdogEnable
#endif
#ifdef	ROM_WatchdogLock
#define	MAP_WatchdogLock	\
ROM_WatchdogLock
#else
#define	MAP_WatchdogLock	\
WatchdogLock
#endif
#ifdef	ROM_WatchdogUnlock
#define	MAP_WatchdogUnlock	\
ROM_WatchdogUnlock
#else
#define	MAP_WatchdogUnlock	\
WatchdogUnlock
#endif
#ifdef ROM_WatchdogLockState
#define MAP_WatchdogLockState ROM_WatchdogLockState
#else
#define MAP_WatchdogLockState WatchdogLockState
#endif

#ifdef ROM_WatchdogReloadSet
#define MAP_WatchdogReloadSet ROM_WatchdogReloadSet
#else
#define MAP_WatchdogReloadSet WatchdogReloadSet
#endif

#ifdef ROM_WatchdogReloadGet
#define MAP_WatchdogReloadGet ROM_WatchdogReloadGet
#else
#define MAP_WatchdogReloadGet WatchdogReloadGet
#endif

#ifdef ROM_WatchdogValueGet
#define MAP_WatchdogValueGet ROM_WatchdogValueGet
#else
#define MAP_WatchdogValueGet WatchdogValueGet
#endif

#ifdef ROM_WatchdogIntStatus
#define MAP_WatchdogIntStatus ROM_WatchdogIntStatus
#else
#define MAP_WatchdogIntStatus WatchdogIntStatus
#endif

#ifdef ROM_WatchdogStallEnable
```c
#define MAP_WatchdogStallEnable ROM_WatchdogStallEnable
#else
#define MAP_WatchdogStallEnable WatchdogStallEnable
#endif
#endif
#define MAP_WatchdogStallDisable ROM_WatchdogStallDisable
#else
#define MAP_WatchdogStallDisable WatchdogStallDisable
#endif
#endif
#define MAP_WatchdogIntRegister ROM_WatchdogIntRegister
#else
#define MAP_WatchdogIntRegister WatchdogIntRegister
#endif
#endif
#define MAP_WatchdogIntUnregister ROM_WatchdogIntUnregister
#else
#define MAP_WatchdogIntUnregister WatchdogIntUnregister
#endif
#endif
//******************************************
// Macros for the I2C API.
//******************************************
#endif
#define MAP_I2CIntRegister ROM_I2CIntRegister
#endif
```
#ifdef ROM_I2CIntRegister
#define MAP_I2CIntRegister \  
   I2CIntRegister
#endif

#ifdef ROM_I2CIntUnregister
#define MAP_I2CIntUnregister \  
   I2CIntUnregister
#else
#define MAP_I2CIntUnregister \  
   I2CIntUnregister
#endif

#ifdef ROM_I2CTxFIFOConfigSet
#define MAP_I2CTxFIFOConfigSet \  
   I2CTxFIFOConfigSet
#else
#define MAP_I2CTxFIFOConfigSet \  
   I2CTxFIFOConfigSet
#endif

#ifdef ROM_I2CTxFIFOFlush
#define MAP_I2CTxFIFOFlush \  
   I2CTxFIFOFlush
#else
#define MAP_I2CTxFIFOFlush \  
   I2CTxFIFOFlush
#endif

#ifdef ROM_I2CRxFIFOConfigSet
#define MAP_I2CRxFIFOConfigSet \  
   I2CRxFIFOConfigSet
#else
#define MAP_I2CRxFIFOConfigSet \  
   I2CRxFIFOConfigSet
#endif

#ifdef ROM_I2CRxFIFOFlush
#define MAP_I2CRxFIFOFlush \  
   I2CRxFIFOFlush
#else
#define MAP_I2CRxFIFOFlush \  
   I2CRxFIFOFlush
#endif

#endif
#define MAP_I2CRxFIFOFlush    I2CRxFIFOFlush
#endif
#ifdef ROM_I2CFIFOStatus
#define MAP_I2CFIFOStatus    ROM_I2CFIFOStatus
#else
#define MAP_I2CFIFOStatus    I2CFIFOStatus
#endif
#ifdef ROM_I2CFIFODataPut
#define MAP_I2CFIFODataPut    ROM_I2CFIFODataPut
#else
#define MAP_I2CFIFODataPut    I2CFIFODataPut
#endif
#ifdef ROM_I2CFIFODataPutNonBlocking
#define MAP_I2CFIFODataPutNonBlocking    ROM_I2CFIFODataPutNonBlocking
#else
#define MAP_I2CFIFODataPutNonBlocking    I2CFIFODataPutNonBlocking
#endif
#ifdef ROM_I2CFIFODataGet
#define MAP_I2CFIFODataGet    ROM_I2CFIFODataGet
#else
#define MAP_I2CFIFODataGet    I2CFIFODataGet
#endif
#ifdef ROM_I2CFIFODataGetNonBlocking
#define MAP_I2CFIFODataGetNonBlocking    ROM_I2CFIFODataGetNonBlocking
#else
#define MAP_I2CFIFODataGetNonBlocking    I2CFIFODataGetNonBlocking
#endif
#endif
#ifdef ROM_I2CMasterBurstLengthSet
#define MAP_I2CMasterBurstLengthSet \  
       ROM_I2CMasterBurstLengthSet
#else
#define MAP_I2CMasterBurstLengthSet \  
       I2CMasterBurstLengthSet
#endif
#ifdef ROM_I2CMasterBurstCountGet
#define MAP_I2CMasterBurstCountGet \  
       ROM_I2CMasterBurstCountGet
#else
#define MAP_I2CMasterBurstCountGet \  
       I2CMasterBurstCountGet
#endif
#ifdef ROM_I2CMasterGlitchFilterConfigSet
#define MAP_I2CMasterGlitchFilterConfigSet \  
       ROM_I2CMasterGlitchFilterConfigSet
#else
#define MAP_I2CMasterGlitchFilterConfigSet \  
       I2CMasterGlitchFilterConfigSet
#endif
#ifdef ROM_I2CSlaveFIFOEnable
#define MAP_I2CSlaveFIFOEnable \  
       ROM_I2CSlaveFIFOEnable
#else
#define MAP_I2CSlaveFIFOEnable \  
       I2CSlaveFIFOEnable
#endif
#ifdef ROM_I2CSlaveFIFODisable
#define MAP_I2CSlaveFIFODisable \  
       ROM_I2CSlaveFIFODisable
#else
#define MAP_I2CSlaveFIFODisable \  
       I2CSlaveFIFODisable
#endif
#ifdef ROM_I2CMasterBusBusy
#define MAP_I2CMasterBusBusy \  
ROM_I2CMasterBusBusy

#else
#define MAP_I2CMasterBusBusy \  
I2CMasterBusBusy
#endif

#ifdef ROM_I2CMasterBusy
#define MAP_I2CMasterBusy \  
ROM_I2CMasterBusy
#else
#define MAP_I2CMasterBusy \  
I2CMasterBusy
#endif

#ifdef ROM_I2CMasterControl
#define MAP_I2CMasterControl \  
ROM_I2CMasterControl
#else
#define MAP_I2CMasterControl \  
I2CMasterControl
#endif

#ifdef ROM_I2CMasterDataGet
#define MAP_I2CMasterDataGet \  
ROM_I2CMasterDataGet
#else
#define MAP_I2CMasterDataGet \  
I2CMasterDataGet
#endif

#ifdef ROM_I2CMasterDataPut
#define MAP_I2CMasterDataPut \  
ROM_I2CMasterDataPut
#else
#define MAP_I2CMasterDataPut \  
I2CMasterDataPut
#endif

#ifdef ROM_I2CMasterDisable
#define MAP_I2CMasterDisable \  
ROM_I2CMasterDisable
#else
#define MAP_I2CMasterDisable \    I2CMasterDisable
#endif

#ifdef ROM_I2CMasterEnable
#define MAP_I2CMasterEnable \    ROM_I2CMasterEnable
#else
#define MAP_I2CMasterEnable \    I2CMasterEnable
#endif

#ifdef ROM_I2CMasterErr
#define MAP_I2CMasterErr \    ROM_I2CMasterErr
#else
#define MAP_I2CMasterErr \    I2CMasterErr
#endif

#ifdef ROM_I2CMasterIntClear
#define MAP_I2CMasterIntClear \    ROM_I2CMasterIntClear
#else
#define MAP_I2CMasterIntClear \    I2CMasterIntClear
#endif

#ifdef ROM_I2CMasterIntDisable
#define MAP_I2CMasterIntDisable \    ROM_I2CMasterIntDisable
#else
#define MAP_I2CMasterIntDisable \    I2CMasterIntDisable
#endif

#ifdef ROM_I2CMasterIntEnable
#define MAP_I2CMasterIntEnable \    ROM_I2CMasterIntEnable
#else
#define MAP_I2CMasterIntEnable \    I2CMasterIntEnable
#endif
I2CMasterIntEnable

#ifdef ROM_I2CMasterIntStatus
#define MAP_I2CMasterIntStatus \       
  ROM_I2CMasterIntStatus
#else
#define MAP_I2CMasterIntStatus \       
  I2CMasterIntStatus
#endif

#ifdef ROM_I2CMasterIntEnableEx
#define MAP_I2CMasterIntEnableEx \     
  ROM_I2CMasterIntEnableEx
#else
#define MAP_I2CMasterIntEnableEx \     
  I2CMasterIntEnableEx
#endif

#ifdef ROM_I2CMasterIntDisableEx
#define MAP_I2CMasterIntDisableEx \    
  ROM_I2CMasterIntDisableEx
#else
#define MAP_I2CMasterIntDisableEx \    
  I2CMasterIntDisableEx
#endif

#ifdef ROM_I2CMasterIntStatusEx
#define MAP_I2CMasterIntStatusEx \     
  ROM_I2CMasterIntStatusEx
#else
#define MAP_I2CMasterIntStatusEx \     
  I2CMasterIntStatusEx
#endif

#ifdef ROM_I2CMasterIntClearEx
#define MAP_I2CMasterIntClearEx \      
  ROM_I2CMasterIntClearEx
#else
#define MAP_I2CMasterIntClearEx \      
  I2CMasterIntClearEx
#endif

#ifdef ROM_I2CMasterIntEnable
#define MAP_I2CMasterIntEnable \       
  ROM_I2CMasterIntEnable
#else
#define MAP_I2CMasterIntEnable \       
  I2CMasterIntEnable
#endif
#ifdef ROM_I2CMasterTimeoutSet
#define MAP_I2CMasterTimeoutSet \  
    ROM_I2CMasterTimeoutSet
#else
#define MAP_I2CMasterTimeoutSet \  
    I2CMasterTimeoutSet
#endif

#ifdef ROM_I2CSlaveACKOverride
#define MAP_I2CSlaveACKOverride \ 
    ROM_I2CSlaveACKOverride
#else
#define MAP_I2CSlaveACKOverride \ 
    I2CSlaveACKOverride
#endif

#ifdef ROM_I2CSlaveACKValueSet
#define MAP_I2CSlaveACKValueSet \ 
    ROM_I2CSlaveACKValueSet
#else
#define MAP_I2CSlaveACKValueSet \ 
    I2CSlaveACKValueSet
#endif

#ifdef ROM_I2CMasterLineStateGet
#define MAP_I2CMasterLineStateGet \ 
    ROM_I2CMasterLineStateGet
#else
#define MAP_I2CMasterLineStateGet \ 
    I2CMasterLineStateGet
#endif

#ifdef ROM_I2CMasterSlaveAddrSet
#define MAP_I2CMasterSlaveAddrSet \ 
    ROM_I2CMasterSlaveAddrSet
#else
#define MAP_I2CMasterSlaveAddrSet \ 
    I2CMasterSlaveAddrSet
#endif

#ifdef ROM_I2CSlaveDataGet
#define MAP_I2CSlaveDataGet \ 
    ROM_I2CSlaveDataGet
#else
#define MAP_I2CSlaveDataGet \ 
    I2CSlaveDataGet
#endif
#else
#define MAP_I2CSlaveDataGet
I2CSlaveDataGet
#endif
#else
#define MAP_I2CSlaveDataPut
I2CSlaveDataPut
#endif
#endif
#define ROM_I2CSlaveDataPut
I2CSlaveDataPut
#endif
#endif
#define ROM_I2CSlaveDisable
I2CSlaveDisable
#endif
#else
#define MAP_I2CSlaveDisable
I2CSlaveDisable
#endif
#endif
#define ROM_I2CSlaveEnable
I2CSlaveEnable
#endif
#endif
#define MAP_I2CSlaveEnable
I2CSlaveEnable
#endif
#endif
#define ROM_I2CSlaveInit
I2CSlaveInit
#endif
#endif
#define MAP_I2CSlaveInit
I2CSlaveInit
#endif
#endif
#define ROM_I2CSlaveAddressSet
I2CSlaveAddressSet
#endif
#else
#define MAP_I2CSlaveAddressSet
I2CSlaveAddressSet
#endif
#endif
#define MAP_I2CSlaveAddressSet
I2CSlaveAddressSet
#endif

#ifndef ROM_I2CSlaveIntClear
#define MAP_I2CSlaveIntClear
ROM_I2CSlaveIntClear
#else
#define MAP_I2CSlaveIntClear
I2CSlaveIntClear
#endif

#ifndef ROM_I2CSlaveIntDisable
#define MAP_I2CSlaveIntDisable
ROM_I2CSlaveIntDisable
#else
#define MAP_I2CSlaveIntDisable
I2CSlaveIntDisable
#endif

#ifndef ROM_I2CSlaveIntEnable
#define MAP_I2CSlaveIntEnable
ROM_I2CSlaveIntEnable
#else
#define MAP_I2CSlaveIntEnable
I2CSlaveIntEnable
#endif

#ifndef ROM_I2CSlaveIntClearEx
#define MAP_I2CSlaveIntClearEx
ROM_I2CSlaveIntClearEx
#else
#define MAP_I2CSlaveIntClearEx
I2CSlaveIntClearEx
#endif

#ifndef ROM_I2CSlaveIntDisableEx
#define MAP_I2CSlaveIntDisableEx
ROM_I2CSlaveIntDisableEx
#else
#define MAP_I2CSlaveIntDisableEx
I2CSlaveIntDisableEx
#endif
#endif

#ifndef ROM_I2CSlaveIntEnableEx
#define MAP_I2CSlaveIntEnableEx \  
  ROM_I2CSlaveIntEnableEx
#endif

#ifndef ROM_I2CSlaveIntStatus
#define MAP_I2CSlaveIntStatus \  
  I2CSlaveIntStatus
#endif

#ifndef ROM_I2CSlaveIntStatusEx
#define MAP_I2CSlaveIntStatusEx \  
  ROM_I2CSlaveIntStatusEx
#endif

#ifndef ROM_I2CSlaveStatus
#define MAP_I2CSlaveStatus \  
  I2CSlaveStatus
#endif

#ifndef ROM_I2CMasterInitExpClk
#define MAP_I2CMasterInitExpClk \  
  ROM_I2CMasterInitExpClk
#endif

#endif
Macros for the SPI API.
#else
#define MAP_SPIIDataGetNonBlocking
SPIIDataGetNonBlocking
#endif

#endif
#define MAP_SPIIDataGet
SPIIDataGet
#endif

#ifndef ROM_SPIDataPutNonBlocking
#define MAP_SPIIDataPutNonBlocking
SPIIDataPutNonBlocking
#endif

#endif
#define MAP_SPIIDataPut
SPIIDataPut
#endif

#ifndef ROM_SPIFIFOEnable
#define MAP_SPIFIFOEnable
SPIFIFOEnable
#endif

#ifndef ROM_SPIFIFODisable
#define MAP_SPIFIFODisable
SPIFIFOEnable
#endif

#endif
#define MAP_SPIFIFODisable SPIFIFODisable
#endif

#ifdef ROM_SPIFIFOLevelSet
#define MAP_SPIFIFOLevelSet ROM_SPIFIFOLevelSet
#else
#define MAP_SPIFIFOLevelSet SPIFIFOLevelSet
#endif

#ifdef ROM_SPIFIFOLevelGet
#define MAP_SPIFIFOLevelGet ROM_SPIFIFOLevelGet
#else
#define MAP_SPIFIFOLevelGet SPIFIFOLevelGet
#endif

#ifdef ROM_SPIWordCountSet
#define MAP_SPIWordCountSet ROM_SPIWordCountSet
#else
#define MAP_SPIWordCountSet SPIWordCountSet
#endif

#ifdef ROM_SPIIntRegister
#define MAP_SPIIntRegister ROM_SPIIntRegister
#else
#define MAP_SPIIntRegister SPIIntRegister
#endif

#ifdef ROM_SPIIntUnregister
#define MAP_SPIIntUnregister ROM_SPIIntUnregister
#else
#define MAP_SPIIntUnregister SPIIntUnregister
#endif
#ifdef ROM_SPIIntEnable
#define MAP_SPIIntEnable \  
  ROM_SPIIntEnable
#else
#define MAP_SPIIntEnable \  
  SPIIntEnable
#endif

#ifdef ROM_SPIIntDisable
#define MAP_SPIIntDisable \  
  ROM_SPIIntDisable
#else
#define MAP_SPIIntDisable \  
  SPIIntDisable
#endif

#ifdef ROM_SPIIntStatus
#define MAP_SPIIntStatus \  
  ROM_SPIIntStatus
#else
#define MAP_SPIIntStatus \  
  SPIIntStatus
#endif

#ifdef ROM_SPIIntClear
#define MAP_SPIIntClear \  
  ROM_SPIIntClear
#else
#define MAP_SPIIntClear \  
  SPIIntClear
#endif

#ifdef ROM_SPIDmaEnable
#define MAP_SPIDmaEnable \  
  ROM_SPIDmaEnable
#else
#define MAP_SPIDmaEnable \  
  SPIDmaEnable
#endif

#ifdef ROM_SPIDmaDisable
#define MAP_SPIDmaDisable
// Macros for the CAM API.

//******************************************
//
// Macros for the CAM API.

#ifdef ROM_CameraReset
# define MAP_CameraReset \
ROM_CameraReset
#endif

//******************************************
ROM_CameraReset

#else
#define MAP_CameraReset \
CameraReset
#endif

#ifdef ROM_CameraParamsConfig
#define MAP_CameraParamsConfig \
ROM_CameraParamsConfig
#else
#define MAP_CameraParamsConfig \
CameraParamsConfig
#endif

#ifdef ROM_CameraXClkConfig
#define MAP_CameraXClkConfig \
ROM_CameraXClkConfig
#else
#define MAP_CameraXClkConfig \
CameraXClkConfig
#endif

#ifdef ROM_CameraXClkSet
#define MAP_CameraXClkSet \
ROM_CameraXClkSet
#else
#define MAP_CameraXClkSet \
CameraXClkSet
#endif

#ifdef ROM_CameraDMAEnable
#define MAP_CameraDMAEnable \
ROM_CameraDMAEnable
#else
#define MAP_CameraDMAEnable \
CameraDMAEnable
#endif

#ifdef ROM_CameraDMADisable
#define MAP_CameraDMADisable \
ROM_CameraDMADisable
#else
#define MAP_CameraDMADisable \
CameraDMADisable
#endif

#else
#define MAP_CameraDMADisable \
CameraDMADisable
#endif
#define MAP_CameraDMADisable \
CameraDMADisable
#endif

#else
#define MAP_CameraThresholdSet \
CameraThresholdSet
#endif

#else
#define MAP_CameraThresholdSet \
CameraThresholdSet
#endif

#else
#define MAP_CameraIntRegister \
CameraIntRegister
#endif

#else
#define MAP_CameraIntRegister \
CameraIntRegister
#endif

#else
#define MAP_CameraIntUnregister \
CameraIntUnregister
#endif

#else
#define MAP_CameraIntUnregister \
CameraIntUnregister
#endif

#else
#define MAP_CameraIntEnable \
CameraIntEnable
#endif

#else
#define MAP_CameraIntEnable \
CameraIntEnable
#endif

#else
#define MAP_CameraIntDisable \
CameraIntDisable
#ifndef ROM_CameraIntStatus
#define MAP_CameraIntStatus
    ROM_CameraIntStatus
#else
#define MAP_CameraIntStatus
    CameraIntStatus
#endif

#ifdef ROM_CameraIntClear
#define MAP_CameraIntClear
    ROM_CameraIntClear
#else
#define MAP_CameraIntClear
    CameraIntClear
#endif

#ifdef ROM_CameraCaptureStop
#define MAP_CameraCaptureStop
    ROM_CameraCaptureStop
#else
#define MAP_CameraCaptureStop
    CameraCaptureStop
#endif

#ifdef ROM_CameraBufferRead
#define MAP_CameraBufferRead
    ROM_CameraBufferRead
#else
#define MAP_CameraBufferRead
    CameraBufferRead
#endif
Macros for the FLASH API.
#else
#define MAP_FlashEraseNonBlocking \
FlashEraseNonBlocking
#endif
#endif
#elif ROM_FlashProgram
#define MAP_FlashProgram \
ROM_FlashProgram
#endif
#endif
#endif
#elif ROM_FlashProgramNonBlocking
#define MAP_FlashProgramNonBlocking \
FlashProgramNonBlocking
#endif
#endif
#endif
#elif ROM_FlashIntRegister
#define MAP_FlashIntRegister \
FlashIntRegister
#endif
#endif
#elif ROM_FlashIntUnregister
#define MAP_FlashIntUnregister \
FlashIntUnregister
#endif
#endif
#elif ROM_FlashIntEnable
#define MAP_FlashIntEnable \
ROM_FlashIntEnable
#endif
#endif
1606  #define MAP_FlashIntEnable    
1607     FlashIntEnable
1608  #endif
1609  #ifdef ROM_FlashIntDisable
1610  #define MAP_FlashIntDisable    
1611     ROM_FlashIntDisable
1612  #else
1613  #define MAP_FlashIntDisable    
1614     FlashIntDisable
1615  #endif
1616  #ifdef ROM_FlashIntStatus
1617  #define MAP_FlashIntStatus    
1618     ROM_FlashIntStatus
1619  #else
1620  #define MAP_FlashIntStatus    
1621     FlashIntStatus
1622  #endif
1623  #ifdef ROM_FlashIntClear
1624  #define MAP_FlashIntClear    
1625     ROM_FlashIntClear
1626  #else
1627  #define MAP_FlashIntClear    
1628     FlashIntClear
1629  #endif
1630  #ifdef ROM_FlashProtectGet
1631  #define MAP_FlashProtectGet    
1632     ROM_FlashProtectGet
1633  #else
1634  #define MAP_FlashProtectGet    
1635     FlashProtectGet
1636  #endif
1637
1638  /***************************************************************************/
1639  /***************************************************************************/
1640  //
1641  // Macros for the Pin API.
1642  //
#ifdef ROM_PinModeSet
#define MAP_PinModeSet ROM_PinModeSet
#else
#define MAP_PinModeSet PinModeSet
#endif

#ifndef ROM_PinDirModeSet
#define MAP_PinDirModeSet ROM_PinDirModeSet
#else
#define MAP_PinDirModeSet PinDirModeSet
#endif

#ifndef ROM_PinDirModeGet
#define MAP_PinDirModeGet ROM_PinDirModeGet
#else
#define MAP_PinDirModeGet PinDirModeGet
#endif

#ifndef ROM_PinModeGet
#define MAP_PinModeGet ROM_PinModeGet
#else
#define MAP_PinModeGet PinModeGet
#endif

#ifndef ROM_PinConfigGet
#define MAP_PinConfigGet ROM_PinConfigGet
#else
#define MAP_PinConfigGet PinConfigGet
#endif
#ifdef ROM_PinConfigSet
#define MAP_PinConfigSet \  
   ROM_PinConfigSet
#else
#define MAP_PinConfigSet \  
   PinConfigSet
#endif
#ifdef ROM_PinTypeUART
#define MAP_PinTypeUART \  
   ROM_PinTypeUART
#else
#define MAP_PinTypeUART \  
   PinTypeUART
#endif
#ifdef ROM_PinTypeI2C
#define MAP_PinTypeI2C \  
   ROM_PinTypeI2C
#else
#define MAP_PinTypeI2C \  
   PinTypeI2C
#endif
#ifdef ROM_PinTypeSPI
#define MAP_PinTypeSPI \  
   ROM_PinTypeSPI
#else
#define MAP_PinTypeSPI \  
   PinTypeSPI
#endif
#ifdef ROM_PinTypeI2S
#define MAP_PinTypeI2S \  
   ROM_PinTypeI2S
#else
#define MAP_PinTypeI2S \  
   PinTypeI2S
#endif
#ifdef ROM_PinTypeTimer
#ifdef ROM_PinTypeTimer
#define MAP_PinTypeTimer PinTypeTimer
#endif

#ifdef ROM_PinTypeCamera
#define MAP_PinTypeCamera ROM_PinTypeCamera
#else
#define MAP_PinTypeCamera PinTypeCamera
#endif

#ifdef ROM_PinTypeGPIO
#define MAP_PinTypeGPIO ROM_PinTypeGPIO
#else
#define MAP_PinTypeGPIO PinTypeGPIO
#endif

#ifdef ROM_PinTypeADC
#define MAP_PinTypeADC ROM_PinTypeADC
#else
#define MAP_PinTypeADC PinTypeADC
#endif

#ifdef ROM_PinTypeSDHost
#define MAP_PinTypeSDHost ROM_PinTypeSDHost
#else
#define MAP_PinTypeSDHost PinTypeSDHost
#endif

//******************************************
***********************************
//
Macros for the SYSTICK API.

ifdef ROM_SysTickEnable
#define MAP_SysTickEnable ROM_SysTickEnable
#else
#define MAP_SysTickEnable SysTickEnable
#endif

ifdef ROM_SysTickDisable
#define MAP_SysTickDisable ROM_SysTickDisable
#else
#define MAP_SysTickDisable SysTickDisable
#endif

ifdef ROM_SysTickIntRegister
#define MAP_SysTickIntRegister ROM_SysTickIntRegister
#else
#define MAP_SysTickIntRegister SysTickIntRegister
#endif

ifdef ROM_SysTickIntUnregister
#define MAP_SysTickIntUnregister ROM_SysTickIntUnregister
#else
#define MAP_SysTickIntUnregister SysTickIntUnregister
#endif

ifdef ROM_SysTickIntEnable
#define MAP_SysTickIntEnable ROM_SysTickIntEnable
#else
#define MAP_SysTickIntEnable SysTickIntEnable
#endif
SysTickIntEnable
#endif
#endif
#define MAP_SysTickIntDisable \
ROM_SysTickIntDisable
#else
#define MAP_SysTickIntDisable \
SysTickIntDisable
#endif
#endif
#ifdef ROM_SysTickPeriodSet
#define MAP_SysTickPeriodSet \
ROM_SysTickPeriodSet
#else
#define MAP_SysTickPeriodSet \
SysTickPeriodSet
#endif
#endif
#ifdef ROM_SysTickPeriodGet
#define MAP_SysTickPeriodGet \
ROM_SysTickPeriodGet
#else
#define MAP_SysTickPeriodGet \
SysTickPeriodGet
#endif
#endif
#ifdef ROM_SysTickValueGet
#define MAP_SysTickValueGet \
ROM_SysTickValueGet
#else
#define MAP_SysTickValueGet \
SysTickValueGet
#endif
#endif
//******************************************
***********************************
//	Macros	for	the	UTILS	API.
//
//******************************************
ifdef ROM_UtilsDelay
#define MAP_UtilsDelay \  
  ROM_UtilsDelay
else
#define MAP_UtilsDelay \  
  UtilsDelay
endif

//******************************************
//
// Macros for the I2S API.
//
//******************************************

ifdef ROM_I2SEnable
#define MAP_I2SEnable \  
  ROM_I2SEnable
else
#define MAP_I2SEnable \  
  I2SEnable
endif

ifdef ROM_I2SDisable
#define MAP_I2SDisable \  
  ROM_I2SDisable
else
#define MAP_I2SDisable \  
  I2SDisable
endif

ifdef ROM_I2SDataPut
#define MAP_I2SDataPut \  
  ROM_I2SDataPut
else
#define MAP_I2SDataPut \  
  I2SDataPut
endif
#ifdef ROM_I2SDataPutNonBlocking
#define MAP_I2SDataPutNonBlocking ROM_I2SDataPutNonBlocking
#else
#define MAP_I2SDataPutNonBlocking I2SDataPutNonBlocking
#endif

#ifdef ROM_I2SDataGet
#define MAP_I2SDataGet ROM_I2SDataGet
#else
#define MAP_I2SDataGet I2SDataGet
#endif

#ifdef ROM_I2SDataGetNonBlocking
#define MAP_I2SDataGetNonBlocking ROM_I2SDataGetNonBlocking
#else
#define MAP_I2SDataGetNonBlocking I2SDataGetNonBlocking
#endif

#ifdef ROM_I2SConfigSetExpClk
#define MAP_I2SConfigSetExpClk ROM_I2SConfigSetExpClk
#else
#define MAP_I2SConfigSetExpClk I2SConfigSetExpClk
#endif

#ifdef ROM_I2STxFIFOEnable
#define MAP_I2STxFIFOEnable ROM_I2STxFIFOEnable
#else
#define MAP_I2STxFIFOEnable I2STxFIFOEnable
#endif

#ifdef ROM_I2STxFIFODisable
#define MAP_I2STxFIFODisable ROM_I2STxFIFODisable
#else
#define MAP_I2STxFIFODisable I2STxFIFODisable
#endif
#ifndef ROM_I2STxFIFODisable
#define MAP_I2STxFIFODisable I2STxFIFODisable
#endif

#ifdef ROM_I2SRxFIFOEnable
#define MAP_I2SRxFIFOEnable ROM_I2SRxFIFOEnable
#else
#define MAP_I2SRxFIFOEnable I2SRxFIFOEnable
#endif

#ifdef ROM_I2SRxFIFODisable
#define MAP_I2SRxFIFODisable ROM_I2SRxFIFODisable
#else
#define MAP_I2SRxFIFODisable I2SRxFIFODisable
#endif

#ifdef ROM_I2STxFIFOStatusGet
#define MAP_I2STxFIFOStatusGet I2STxFIFOStatusGet
#else
#define MAP_I2STxFIFOStatusGet I2STxFIFOStatusGet
#endif

#ifdef ROM_I2SRxFIFOStatusGet
#define MAP_I2SRxFIFOStatusGet I2SRxFIFOStatusGet
#else
#define MAP_I2SRxFIFOStatusGet I2SRxFIFOStatusGet
#endif

#ifdef ROM_I2SSerializerConfig
#define MAP_I2SSerializerConfig ROM_I2SSerializerConfig
#else
#define MAP_I2SSerializerConfig ROM_I2SSerializerConfig
#endif

#ifndef ROM_I2STxFIFODisable
#define MAP_I2STxFIFODisable I2STxFIFODisable
#endif

#ifndef ROM_I2SRxFIFOEnable
#define MAP_I2SRxFIFOEnable I2SRxFIFOEnable
#endif
#define MAP_I2SSerializerConfig I2SSerializerConfig
#endif
#ifdef ROM_I2SIntEnable
#define MAP_I2SIntEnable
ROM_I2SIntEnable
#else
#define MAP_I2SIntEnable
I2SIntEnable
#endif
#ifdef ROM_I2SIntDisable
#define MAP_I2SIntDisable
ROM_I2SIntDisable
#else
#define MAP_I2SIntDisable
I2SIntDisable
#endif
#ifdef ROM_I2SIntStatus
#define MAP_I2SIntStatus
ROM_I2SIntStatus
#else
#define MAP_I2SIntStatus
I2SIntStatus
#endif
#ifdef ROM_I2SIntClear
#define MAP_I2SIntClear
ROM_I2SIntClear
#else
#define MAP_I2SIntClear
I2SIntClear
#endif
#ifdef ROM_I2SIntRegister
#define MAP_I2SIntRegister
ROM_I2SIntRegister
#else
#define MAP_I2SIntRegister
I2SIntRegister
// Macros for the GPIO API.

//******************************************
//
// Macros for the GPIO API.
//******************************************

#ifdef ROM_I2SIntUnregister
#define MAP_I2SIntUnregister \  
ROM_I2SIntUnregister
#else
#define MAP_I2SIntUnregister \  
I2SIntUnregister
#endif

//******************************************
//
// Macros for the GPIO API.
//******************************************

#ifdef ROM_GPIODirModeSet
#define MAP_GPIODirModeSet \ 
ROM_GPIODirModeSet
#else
#define MAP_GPIODirModeSet \ 
GPIODirModeSet
#endif

#ifdef ROM_GPIOIntTypeSet
#define MAP_GPIOIntTypeSet \ 
ROM_GPIOIntTypeSet
#else
#define MAP_GPIOIntTypeSet \ 
GPIOIntTypeSet
#endif
#ifdef ROM_GPIODMATriggerEnable
#define MAP_GPIODMATriggerEnable \
ROM_GPIODMATriggerEnable
#else
#define MAP_GPIODMATriggerEnable \
GPIODMATriggerEnable
#endif

#ifdef ROM_GPIODMATriggerDisable
#define MAP_GPIODMATriggerDisable \
ROM_GPIODMATriggerDisable
#else
#define MAP_GPIODMATriggerDisable \
GPIODMATriggerDisable
#endif

#ifdef ROM_GPIOIntTypeGet
#define MAP_GPIOIntTypeGet \
ROM_GPIOIntTypeGet
#else
#define MAP_GPIOIntTypeGet \
GPIOIntTypeGet
#endif

#ifdef ROM_GPIOIntEnable
#define MAP_GPIOIntEnable \
ROM_GPIOIntEnable
#else
#define MAP_GPIOIntEnable \
GPIOIntEnable
#endif

#ifdef ROM_GPIOIntDisable
#define MAP_GPIOIntDisable \
ROM_GPIOIntDisable
#else
#define MAP_GPIOIntDisable \
GPIOIntDisable
#endif

#ifdef ROM_GPIOIntStatus
ROM_GPIOIntStatus
#else
#define MAP_GPIOIntStatus \
    GPIOIntStatus
#endif
#endif
#define MAP_GPIOIntClear \
    GPIOIntClear
#endif
#endif
#define MAP_GPIOIntRegister \
    GPIOIntRegister
#endif
#endif
#define MAP_GPIOIntUnregister \
    GPIOIntUnregister
#endif
#endif
#define MAP_GPIOPinRead \
    GPIOPinRead
#endif
#endif
#define MAP_GPIOPinWrite \
    GPIOPinWrite
#endif
#else
// Macros for the AES API.

//******************************************
//
// Macros for the AES API.
//
//******************************************

#ifdef ROM_AESConfigSet
#define MAP_AESConfigSet ROM_AESConfigSet
#else
#define MAP_AESConfigSet AESConfigSet
#endif

#ifdef ROM_AESKey1Set
#define MAP_AESKey1Set ROM_AESKey1Set
#else
#define MAP_AESKey1Set AESKey1Set
#endif

#ifdef ROM_AESKey2Set
#define MAP_AESKey2Set ROM_AESKey2Set
#else
#define MAP_AESKey2Set AESKey2Set
#endif

#ifdef ROM_AESKey3Set
#define MAP_AESKey3Set ROM_AESKey3Set
#else
#define MAP_AESKey3Set AESKey3Set
#endif
AESKey3Set
#endif
#endif
#define MAP_AESIVSet \  ROM_AESIVSet
#else
#define MAP_AESIVSet \  AESIVSet
#endif
#endif
#define MAP_AESTagRead \  AESTagRead
#else
#define MAP_AESTagRead \  AESTagRead
#endif
#endif
#define MAP_AESDataLengthSet \  AESDataLengthSet
#else
#define MAP_AESDataLengthSet \  AESDataLengthSet
#endif
#endif
#define MAP_AESAuthDataLengthSet \  AESAuthDataLengthSet
#else
#define MAP_AESAuthDataLengthSet \  AESAuthDataLengthSet
#endif
#endif
#define MAP_AESDataReadNonBlocking \  AESDataReadNonBlocking
#else
#define MAP_AESDataReadNonBlocking \  AESDataReadNonBlocking
#endif
#ifdef ROM_AESDataRead
#define MAP_AESDataRead
  ROM_AESDataRead
#else
#define MAP_AESDataRead
  AESDataRead
#endif

#ifdef ROM_AESDataWriteNonBlocking
#define MAP_AESDataWriteNonBlocking
  ROM_AESDataWriteNonBlocking
#else
#define MAP_AESDataWriteNonBlocking
  AESDataWriteNonBlocking
#endif

#ifdef ROM_AESDataWrite
#define MAP_AESDataWrite
  ROM_AESDataWrite
#else
#define MAP_AESDataWrite
  AESDataWrite
#endif

#ifdef ROM_AESDataProcess
#define MAP_AESDataProcess
  ROM_AESDataProcess
#else
#define MAP_AESDataProcess
  AESDataProcess
#endif

#ifdef ROM_AESDataMAC
#define MAP_AESDataMAC
  ROM_AESDataMAC
#else
#define MAP_AESDataMAC
  AESDataMAC
#endif

#ifdef ROM_AESDataProcessAE
#define MAP_AESDataProcessAE
#endif
ROM_AESDataProcessAE
#else
#define MAP_AESDataProcessAE \
    AESDataProcessAE
#endif
#ifdef ROM_AESIntStatus
#define MAP_AESIntStatus \
    ROM_AESIntStatus
#else
#define MAP_AESIntStatus \
    AESIntStatus
#endif
#ifdef ROM_AESIntEnable
#define MAP_AESIntEnable \
    ROM_AESIntEnable
#else
#define MAP_AESIntEnable \
    AESIntEnable
#endif
#ifdef ROM_AESIntDisable
#define MAP_AESIntDisable \
    ROM_AESIntDisable
#else
#define MAP_AESIntDisable \
    AESIntDisable
#endif
#ifdef ROM_AESIntClear
#define MAP_AESIntClear \
    ROM_AESIntClear
#else
#define MAP_AESIntClear \
    AESIntClear
#endif
#ifdef ROM_AESIntRegister
#define MAP_AESIntRegister \
    ROM_AESIntRegister
#else
//******************************************
// Macros for the DES API.
//******************************************
#ifdef ROM_DESConfigSet
#define MAP_DESConfigSet \ 
    ROM_DESConfigSet
#else
#define MAP_DESConfigSet \ 
    DESConfigSet
#endif

//ifdef ROM_AESIntUnregister
#define MAP_AESIntUnregister \ 
    AESIntUnregister
#endif

//ifdef ROM_AESDMAEnable
#define MAP_AESDMAEnable \ 
    AESDMAEnable
#endif

//ifdef ROM_AESDMADisable
#define MAP_AESDMADisable \ 
    AESDMADisable
#endif

//***********************************************************************
//***********************************************************************
DESConfigSet

#ifndef ROM_DESDataRead
#define MAP_DESDataRead \n    ROM_DESDataRead
#else
#define MAP_DESDataRead \n    DESDataRead
#endif

#ifndef ROM_DESDataReadNonBlocking
#define MAP_DESDataReadNonBlocking \n    ROM_DESDataReadNonBlocking
#else
#define MAP_DESDataReadNonBlocking \n    DESDataReadNonBlocking
#endif

#ifndef ROM_DESDataProcess
#define MAP_DESDataProcess \n    ROM_DESDataProcess
#else
#define MAP_DESDataProcess \n    DESDataProcess
#endif

#ifndef ROM_DESDataWrite
#define MAP_DESDataWrite \n    ROM_DESDataWrite
#else
#define MAP_DESDataWrite \n    DESDataWrite
#endif

#ifndef ROM_DESDataWriteNonBlocking
#define MAP_DESDataWriteNonBlocking \n    ROM_DESDataWriteNonBlocking
#else
#define MAP_DESDataWriteNonBlocking \n    DESDataWriteNonBlocking
#endif
#ifdef ROM_DESDMADisable
#define MAP_DESDMADisable \  
ROM_DESDMADisable
#else
#define MAP_DESDMADisable \  
DESDMADisable
#endif

#ifdef ROM_DESDMAEnable
#define MAP_DESDMAEnable \  
ROM_DESDMAEnable
#else
#define MAP_DESDMAEnable \  
DESDMAEnable
#endif

#ifdef ROM_DESIntClear
#define MAP_DESIntClear \  
ROM_DESIntClear
#else
#define MAP_DESIntClear \  
DESIntClear
#endif

#ifdef ROM_DESIntDisable
#define MAP_DESIntDisable \  
ROM_DESIntDisable
#else
#define MAP_DESIntDisable \  
DESIntDisable
#endif

#ifdef ROM_DESIntEnable
#define MAP_DESIntEnable \  
ROM_DESIntEnable
#else
#define MAP_DESIntEnable \  
DESIntEnable
#endif

#ifdef ROM_DESIntRegister
ROM_DESIntRegister

#define MAP_DESIntRegister \ DESIntRegister

#else
#endif

#ifndef ROM_DESIntStatus
#define MAP_DESIntStatus \ ROM_DESIntStatus
#else
#define MAP_DESIntStatus \ DESIntStatus
#endif

#ifndef ROM_DESIntUnregister
#define MAP_DESIntUnregister \ ROM_DESIntUnregister
#else
#define MAP_DESIntUnregister \ DESIntUnregister
#endif

#ifndef ROM_DESIVSet
#define MAP_DESIVSet \ ROM_DESIVSet
#else
#define MAP_DESIVSet \ DESIVSet
#endif

#ifndef ROM_DESKeySet
#define MAP_DESKeySet \ ROM_DESKeySet
#else
#define MAP_DESKeySet \ DESKeySet
#endif

#ifndef ROM_DESDataLengthSet
#define MAP_DESDataLengthSet \ ROM_DESDataLengthSet
#else
#define MAP_DESDataLengthSet \ DESDataLengthSet
#endif

#else
// Macros for the SHAMD5 API.

#ifndef ROM_SHAMD5ConfigSet
#define MAP_SHAMD5ConfigSet \    
    ROM_SHAMD5ConfigSet
#endif

#ifndef ROM_SHAMD5DataProcess
#define MAP_SHAMD5DataProcess \    
    ROM_SHAMD5DataProcess
#endif

#ifndef ROM_SHAMD5DataWrite
#define MAP_SHAMD5DataWrite \    
    ROM_SHAMD5DataWrite
#endif

#ifndef ROM_SHAMD5DataWriteNonBlocking
#define MAP_SHAMD5DataWriteNonBlocking \    
    ROM_SHAMD5DataWriteNonBlocking
#endif
SHAMD5DataWriteNonBlocking
#endif
#ifdef ROM_SHAMD5DMADisable
#define MAP_SHAMD5DMADisable \ ROM_SHAMD5DMADisable
#else
#define MAP_SHAMD5DMADisable \ SHAMD5DMADisable
#endif
#ifdef ROM_SHAMD5DMAEnable
#define MAP_SHAMD5DMAEnable \ ROM_SHAMD5DMAEnable
#else
#define MAP_SHAMD5DMAEnable \ SHAMD5DMAEnable
#endif
#ifdef ROM_SHAMD5DataLengthSet
#define MAP_SHAMD5DataLengthSet \ ROM_SHAMD5DataLengthSet
#else
#define MAP_SHAMD5DataLengthSet \ SHAMD5DataLengthSet
#endif
#ifdef ROM_SHAMD5HMACKeySet
#define MAP_SHAMD5HMACKeySet \ ROM_SHAMD5HMACKeySet
#else
#define MAP_SHAMD5HMACKeySet \ SHAMD5HMACKeySet
#endif
#ifdef ROM_SHAMD5HMACCPPKeyGenerate
#define MAP_SHAMD5HMACCPPKeyGenerate \ ROM_SHAMD5HMACCPPKeyGenerate
#else
#define MAP_SHAMD5HMACCPPKeyGenerate \ SHAMD5HMACCPPKeyGenerate
#endif
```c
#ifdef ROM_SHAMD5HMACPPKeySet
#define MAP_SHAMD5HMACPPKeySet
        ROM_SHAMD5HMACPPKeySet
#else
#define MAP_SHAMD5HMACPPKeySet
        SHAMD5HMACPPKeySet
#endif

#ifdef ROM_SHAMD5HMACProcess
#define MAP_SHAMD5HMACProcess
        ROM_SHAMD5HMACProcess
#else
#define MAP_SHAMD5HMACProcess
        SHAMD5HMACProcess
#endif

#ifdef ROM_SHAMD5IntClear
#define MAP_SHAMD5IntClear
        ROM_SHAMD5IntClear
#else
#define MAP_SHAMD5IntClear
        SHAMD5IntClear
#endif

#ifdef ROM_SHAMD5IntDisable
#define MAP_SHAMD5IntDisable
        ROM_SHAMD5IntDisable
#else
#define MAP_SHAMD5IntDisable
        SHAMD5IntDisable
#endif

#ifdef ROM_SHAMD5IntEnable
#define MAP_SHAMD5IntEnable
        ROM_SHAMD5IntEnable
#else
#define MAP_SHAMD5IntEnable
        SHAMD5IntEnable
#endif

#ifdef ROM_SHAMD5IntRegister
#define MAP_SHAMD5IntRegister
```
#define MAP_SHAMD5IntRegister \ SHAMD5IntRegister
#endif

#ifdef ROM_SHAMD5IntStatus
#define MAP_SHAMD5IntStatus \ ROM_SHAMD5IntStatus
#else
#define MAP_SHAMD5IntStatus \ SHAMD5IntStatus
#endif

#ifdef ROM_SHAMD5IntUnregister
#define MAP_SHAMD5IntUnregister \ ROM_SHAMD5IntUnregister
#else
#define MAP_SHAMD5IntUnregister \ SHAMD5IntUnregister
#endif

#ifdef ROM_SHAMD5ResultRead
#define MAP_SHAMD5ResultRead \ ROM_SHAMD5ResultRead
#else
#define MAP_SHAMD5ResultRead \ SHAMD5ResultRead
#endif

//**************************************************
// Macros for the CRC API.
//**************************************************
#ifdef ROM_CRCConfigSet
#define MAP_CRCConfigSet \ ROM_CRCConfigSet
#else
#define MAP_CRCConfigSet \ CRCConfigSet
#endif

//**************************************************
#ifdef MAP_CRCConfigSet
  CRCConfigSet
#endif

#define MAP_CRCDataProcess \
  ROM_CRCDataProcess
#elif ROM_CRCDataProcess
#define MAP_CRCDataProcess \
  CRCDataProcess
#else
#define MAP_CRCDataProcess \
  ROM_CRCDataWrite
#elif ROM_CRCDataWrite
#define MAP_CRCDataWrite \
  CRCDataWrite
#else
#define MAP_CRCDataWrite \
  ROM_CRCResultRead
#elif ROM_CRCResultRead
#define MAP_CRCResultRead \
  CRCResultRead
#else
#define MAP_CRCResultRead \
  ROM_CRCSeedSet
#elif ROM_CRCSeedSet
#define MAP_CRCSeedSet \
  CRCSeedSet
#else
#endif

//******************************************
***********************************
//
// Macros for the SDHOST API.
```c
#ifndef ROM_SDHostCmdReset
#define MAP_SDHostCmdReset \  ROM_SDHostCmdReset
#else
#define MAP_SDHostCmdReset \  SDHostCmdReset
#endif

#ifndef ROM_SDHostInit
#define MAP_SDHostInit \  ROM_SDHostInit
#else
#define MAP_SDHostInit \  SDHostInit
#endif

#ifndef ROM_SDHostCmdSend
#define MAP_SDHostCmdSend \  ROM_SDHostCmdSend
#else
#define MAP_SDHostCmdSend \  SDHostCmdSend
#endif

#ifndef ROM_SDHostIntRegister
#define MAP_SDHostIntRegister \  ROM_SDHostIntRegister
#else
#define MAP_SDHostIntRegister \  SDHostIntRegister
#endif

#ifndef ROM_SDHostIntUnregister
#define MAP_SDHostIntUnregister \  ROM_SDHostIntUnregister
#else
#define MAP_SDHostIntUnregister \  SDHostIntUnregister
#endif
```
#define MAP_SDHostRespGet \  
ROM_SDHostRespGet
#else
#define MAP_SDHostRespGet \  
SDHostRespGet
#endif
#ifndef ROM_SDHostBlockSizeSet
#define MAP_SDHostBlockSizeSet \  
ROM_SDHostBlockSizeSet
#else
#define MAP_SDHostBlockSizeSet \  
SDHostBlockSizeSet
#endif
#ifndef ROM_SDHostBlockCountSet
#define MAP_SDHostBlockCountSet \  
ROM_SDHostBlockCountSet
#else
#define MAP_SDHostBlockCountSet \  
SDHostBlockCountSet
#endif
#ifndef ROM_SDHostDataNonBlockingWrite
#define MAP_SDHostDataNonBlockingWrite \  
ROM_SDHostDataNonBlockingWrite
#else
#define MAP_SDHostDataNonBlockingWrite \  
SDHostDataNonBlockingWrite
#endif
#ifndef ROM_SDHostDataNonBlockingRead
#define MAP_SDHostDataNonBlockingRead \  
ROM_SDHostDataNonBlockingRead
#else
#define MAP_SDHostDataNonBlockingRead \  
SDHostDataNonBlockingRead
#endif
#ifndef ROM_SDHostDataWrite
#define MAP_SDHostDataWrite \  
ROM_SDHostDataWrite
#ifdef ROM_SDHostDataRead
#define MAP_SDHostDataRead SDHostDataRead
#endif

#elif
#define MAP_SDHostDataWrite SDHostDataWrite
#endif

#ifdef ROM_SDHostSetExpClk
#define MAP_SDHostSetExpClk ROM_SDHostSetExpClk
#else
#define MAP_SDHostSetExpClk SDHostSetExpClk
#endif

#ifdef ROM_PRCMMCUReset
#define MAP_PRCMMCUReset ROM_PRCMMCUReset
#else
#define MAP_PRCMMCUReset PRCMMCUReset
#endif

#ifdef ROM_PRCMSysResetCauseGet
#define MAP_PRCMSysResetCauseGet ROM_PRCMSysResetCauseGet
#else
#define MAP_PRCMSysResetCauseGet PRCMSysResetCauseGet
#endif

// Macros for the PRCM API.
#define MAP_PRCMSysResetCauseGet
    PRCMSysResetCauseGet
#endif

#ifdef ROM_PRCMPeripheralClkEnable
#define MAP_PRCMPeripheralClkEnable
    ROM_PRCMPeripheralClkEnable
#else
#define MAP_PRCMPeripheralClkEnable
    PRCMPeripheralClkEnable
#endif

#ifdef ROM_PRCMPeripheralClkDisable
#define MAP_PRCMPeripheralClkDisable
    ROM_PRCMPeripheralClkDisable
#else
#define MAP_PRCMPeripheralClkDisable
    PRCMPeripheralClkDisable
#endif

#ifdef ROM_PRCMPeripheralReset
#define MAP_PRCMPeripheralReset
    ROM_PRCMPeripheralReset
#else
#define MAP_PRCMPeripheralReset
    PRCMPeripheralReset
#endif

#ifdef ROM_PRCMPeripheralStatusGet
#define MAP_PRCMPeripheralStatusGet
    ROM_PRCMPeripheralStatusGet
#else
#define MAP_PRCMPeripheralStatusGet
    PRCMPeripheralStatusGet
#endif

#ifdef ROM_PRCMI2SClockFreqSet
#define MAP_PRCMI2SClockFreqSet
    ROM_PRCMI2SClockFreqSet
#else
#define MAP_PRCMI2SClockFreqSet
    PRCMI2SClockFreqSet
#endif
ifdef ROM_PRCMPeripheralClockGet
#define MAP_PRCMPeripheralClockGet \     ROM_PRCMPeripheralClockGet
#else
#define MAP_PRCMPeripheralClockGet \     PRCMPeripheralClockGet
#endif

ifdef ROM_PRCMSleepEnter
#define MAP_PRCMSleepEnter \     ROM_PRCMSleepEnter
#else
#define MAP_PRCMSleepEnter \     PRCMSleepEnter
#endif

ifdef ROM_PRCMDeepSleepEnter
#define MAP_PRCMDeepSleepEnter \     ROM_PRCMDeepSleepEnter
#else
#define MAP_PRCMDeepSleepEnter \     PRCMDeepSleepEnter
#endif

ifdef ROM_PRCMSRAMRetentionEnable
#define MAP_PRCMSRAMRetentionEnable \     ROM_PRCMSRAMRetentionEnable
#else
#define MAP_PRCMSRAMRetentionEnable \     PRCMSRAMRetentionEnable
#endif

ifdef ROM_PRCMSRAMRetentionDisable
#define MAP_PRCMSRAMRetentionDisable \     PRCMSRAMRetentionDisable
#else
#define MAP_PRCMSRAMRetentionDisable \     PRCMSRAMRetentionDisable
#endif

ifdef ROM_PRCMLPDSEnter
#define MAP_PRCMLPDEnter \  
ROM_PRCMLPDEnter
#else
#define MAP_PRCMLPDEnter \  
PRCMLPDEnter
#endif
#ifdef ROM_PRCMLPDSIntervalSet
#define MAP_PRCMLPDSIntervalSet \  
ROM_PRCMLPDSIntervalSet
#else
#define MAP_PRCMLPDSIntervalSet \  
PRCMLPDSIntervalSet
#endif
#ifdef ROM_PRCMLPDSWakeupSourceEnable
#define MAP_PRCMLPDSWakeupSourceEnable \  
ROM_PRCMLPDSWakeupSourceEnable
#else
#define MAP_PRCMLPDSWakeupSourceEnable \  
PRCMLPDSWakeupSourceEnable
#endif
#ifdef ROM_PRCMLPDSWakeupCauseGet
#define MAP_PRCMLPDSWakeupCauseGet \  
ROM_PRCMLPDSWakeupCauseGet
#else
#define MAP_PRCMLPDSWakeupCauseGet \  
PRCMLPDSWakeupCauseGet
#endif
#ifdef ROM_PRCMLPDSWakeUpGPIOSelect
#define MAP_PRCMLPDSWakeUpGPIOSelect \  
ROM_PRCMLPDSWakeUpGPIOSelect
#else
#define MAP_PRCMLPDSWakeUpGPIOSelect \  
PRCMLPDSWakeUpGPIOSelect
#endif
#ifdef ROM_PRCMLPDSWakeupSourceDisable
#define MAP_PRCMLPDSWakeupSourceDisable \  
ROM_PRCMLPDSWakeupSourceDisable
#else
#define MAP_PRCMLPDSWakeupSourceDisable \  PRCMLPDSWakeupSourceDisable
#endif

#define MAP_PRCMHibernateEnter \  PRCMHibernateEnter
#endif

#define MAP_PRCMHibernateWakeupSourceEnable \  PRCMHibernateWakeupSourceEnable
#endif

#define MAP_PRCMHibernateWakeupCauseGet \  PRCMHibernateWakeupCauseGet
#endif

#define MAP_PRCMHibernateWakeUpGPIOSelect \  PRCMHibernateWakeUpGPIOSelect
#endif

#endif

#define MAP_PRCMHibernateWakeupSourceDisable \  PRCMHibernateWakeupSourceDisable
}
ROM_PRCMHibernateWakeupSourceDisable

#define MAP_PRCMHibernateWakeupSourceDisable

PRCMHibernateWakeupSourceDisable

#endif

#endif

#define MAP_PRCMHibernateIntervalSet

PRCMHibernateIntervalSet

#endif

#endif

#define MAP_PRCMSlowClkCtrGet

PRCMSlowClkCtrGet

#endif

#endif

#define MAP_PRCMSlowClkCtrMatchSet

PRCMSlowClkCtrMatchSet

#endif

#endif

#define MAP_PRCMSlowClkCtrMatchGet

PRCMSlowClkCtrMatchGet

#endif

#endif

#define MAP_PRCMOCRRegisterWrite

ROM_PRCMOCRRegisterWrite
#else
#define MAP_PRCMOCRRegisterWrite \  
PRCMOCRRegisterWrite
#endif

#ifdef ROM_PRCMOCRRegisterRead
#define MAP_PRCMOCRRegisterRead \  
ROM_PRCMOCRRegisterRead
#else
#define MAP_PRCMOCRRegisterRead \  
PRCMOCRRegisterRead
#endif

#ifdef ROM_PRCMIntRegister
#define MAP_PRCMIntRegister \  
ROM_PRCMIntRegister
#else
#define MAP_PRCMIntRegister \  
PRCMIntRegister
#endif

#ifdef ROM_PRCMIntUnregister
#define MAP_PRCMIntUnregister \  
ROM_PRCMIntUnregister
#else
#define MAP_PRCMIntUnregister \  
PRCMIntUnregister
#endif

#ifdef ROM_PRCMIntEnable
#define MAP_PRCMIntEnable \  
ROM_PRCMIntEnable
#else
#define MAP_PRCMIntEnable \  
PRCMIntEnable
#endif

#ifdef ROM_PRCMIntDisable
#define MAP_PRCMIntDisable \  
ROM_PRCMIntDisable
#else
#define MAP_PRCMIntDisable \  
PRCMIntDisable
#endif
PRCMIntDisable

#ifdef ROM_PRCMIntStatus
#define MAP_PRCMIntStatus \ 
ROM_PRCMIntStatus
#else
#define MAP_PRCMIntStatus \ 
PRCMIntStatus
#endif

#ifdef ROM_PRCMRTCInUseSet
#define MAP_PRCMRTCInUseSet \ 
ROM_PRCMRTCInUseSet
#else
#define MAP_PRCMRTCInUseSet \ 
PRCMRTCInUseSet
#endif

#ifdef ROM_PRCMRTCInUseGet
#define MAP_PRCMRTCInUseGet \ 
ROM_PRCMRTCInUseGet
#else
#define MAP_PRCMRTCInUseGet \ 
PRCMRTCInUseGet
#endif

#ifdef ROM_PRCMRTCSet
#define MAP_PRCMRTCSet \ 
ROM_PRCMRTCSet
#else
#define MAP_PRCMRTCSet \ 
PRCMRTCSet
#endif

ifdef ROM_PRCMRTCGet
#define MAP_PRCMRTCGet \ 
ROM_PRCMRTCGet
#else
#define MAP_PRCMRTCGet \ 
PRCMRTCGet
#endif

#endif
#ifdef ROM_PRCMRTCMatchSet
#define MAP_PRCMRTCMatchSet
   ROM_PRCMRTCMatchSet
#else
#define MAP_PRCMRTCMatchSet
   PRCMRTCMatchSet
#endif

#ifdef ROM_PRCMRTCMatchGet
#define MAP_PRCMRTCMatchGet
   ROM_PRCMRTCMatchGet
#else
#define MAP_PRCMRTCMatchGet
   PRCMRTCMatchGet
#endif

#ifdef ROM_PRCMLPDSRestoreInfoSet
#define MAP_PRCMLPDSRestoreInfoSet
   ROM_PRCMLPDSRestoreInfoSet
#else
#define MAP_PRCMLPDSRestoreInfoSet
   PRCMLPDSRestoreInfoSet
#endif

#ifdef ROM_PRCMHIBRegRead
#define MAP_PRCMHIBRegRead
   ROM_PRCMHIBRegRead
#else
#define MAP_PRCMHIBRegRead
   PRCMHIBRegRead
#endif

#ifdef ROM_PRCMHIBRegWrite
#define MAP_PRCMHIBRegWrite
   ROM_PRCMHIBRegWrite
#else
#define MAP_PRCMHIBRegWrite
   PRCMHIBRegWrite
#endif
Macros for the HWSPINLOCK API.

ifdef ROM_HwSpinLockAcquire
#define MAP_HwSpinLockAcquire ROM_HwSpinLockAcquire
#else
#define MAP_HwSpinLockAcquire HwSpinLockAcquire
#endif

ifdef ROM_HwSpinLockTryAcquire
#define MAP_HwSpinLockTryAcquire ROM_HwSpinLockTryAcquire
#else
#define MAP_HwSpinLockTryAcquire HwSpinLockTryAcquire
#endif

ifdef ROM_HwSpinLockRelease
#define MAP_HwSpinLockRelease ROM_HwSpinLockRelease
#else
#define MAP_HwSpinLockRelease HwSpinLockRelease
#endif

ifdef ROM_HwSpinLockTest
#define MAP_HwSpinLockTest ROM_HwSpinLockTest
#else
#define MAP_HwSpinLockTest HwSpinLockTest
#endif
Macros for the ADC API.

```c
#ifdef ROM_ADCEnable
#define MAP_ADCEnable \    ROM_ADCEnable
#else
#define MAP_ADCEnable \    ADCEnable
#endif

#ifdef ROM_ADCDisable
#define MAP_ADCDisable \    ROM_ADCDisable
#else
#define MAP_ADCDisable \    ADCDisable
#endif

#ifdef ROM_ADCChannelEnable
#define MAP_ADCChannelEnable \    ROM_ADCChannelEnable
#else
#define MAP_ADCChannelEnable \    ADCChannelEnable
#endif

#ifdef ROM_ADCChannelDisable
#define MAP_ADCChannelDisable \    ROM_ADCChannelDisable
#else
#define MAP_ADCChannelDisable \    ADCChannelDisable
#endif

#ifdef ROM_ADCIntRegister
#define MAP_ADCIntRegister \
#endif
```
3056 #else
3057 #define MAP_ADCIntRegister \  
3059 ADCIntRegister
3060 #endif
3061 #ifdef ROM_ADCIntUnregister
3062 #define MAP_ADCIntUnregister \  
3063 ROM_ADCIntUnregister
3064 #else
3065 #define MAP_ADCIntUnregister \  
3067 ADCIntUnregister
3068 #endif
3069 #ifdef ROM_ADCIntEnable
3070 #define MAP_ADCIntEnable \  
3072 ROM_ADCIntEnable
3074 #else
3076 #define MAP_ADCIntEnable \  
3078 ADCIntEnable
3080 #endif
3082 #ifdef ROM_ADCIntDisable
3083 #define MAP_ADCIntDisable \  
3085 ROM_ADCIntDisable
3088 #else
3090 #define MAP_ADCIntDisable \  
3092 ADCIntDisable
3094 #endif
3095 #ifdef ROM_ADCIntStatus
3097 #define MAP_ADCIntStatus \  
3099 ROM_ADCIntStatus
3098 #endif
3099 #ifdef ROM_ADCIntClear
3102 #define MAP_ADCIntClear \  
3105 ROM_ADCIntClear
3106 #else
#define MAP_ADCIntClear \    ADCIntClear
#endif

#ifdef ROM_ADCDMAEnable
#define MAP_ADCDMAEnable \    ROM_ADCDMAEnable
#else
#define MAP_ADCDMAEnable \    ADCDMAEnable
#endif

#ifdef ROM_ADCDMADisable
#define MAP_ADCDMADisable \    ROM_ADCDMADisable
#else
#define MAP_ADCDMADisable \    ADCDMADisable
#endif

#ifdef ROM_ADCChannelGainSet
#define MAP_ADCChannelGainSet \    ROM_ADCChannelGainSet
#else
#define MAP_ADCChannelGainSet \    ADCChannelGainSet
#endif

#ifdef ROM_ADCChannelGainGet
#define MAP_ADCChannelGainGet \    ROM_ADCChannelGainGet
#else
#define MAP_ADCChannelGainGet \    ADCChannelGainGet
#endif

#ifdef ROM_ADCTimerConfig
#define MAP_ADCTimerConfig \    ROM_ADCTimerConfig
#else
#define MAP_ADCTimerConfig \    ADCTimerConfig
#endif
```c
#ifdef ROM_ADCTimerEnable
#define MAP_ADCTimerEnable ROM_ADCTimerEnable
#else
#define MAP_ADCTimerEnable ADCTimerEnable
#endif

#ifdef ROM_ADCTimerDisable
#define MAP_ADCTimerDisable ROM_ADCTimerDisable
#else
#define MAP_ADCTimerDisable ADCTimerDisable
#endif

#ifdef ROM_ADCTimerReset
#define MAP_ADCTimerReset ROM_ADCTimerReset
#else
#define MAP_ADCTimerReset ADCTimerReset
#endif

#ifdef ROM_ADCFIFOLvlGet
#define MAP_ADCFIFOLvlGet ROM_ADCFIFOLvlGet
#else
#define MAP_ADCFIFOLvlGet ADCFIFOLvlGet
#endif

#ifdef ROM_ADCFIFORead
```
#define MAP_ADCFIFORead \ ROM_ADCFIFORead
#endif

#define MAP_ADCFIFORead \ ADCFIFORead
#endif // __ROM_MAP_H__

//******************************************
***********************************
//
3 //    rom_patch.h
4 //
5 //  Macros to facilitate patching driverlib
   API's in the ROM.
6 //
7 //  Copyright (C) 2014 Texas Instruments
   Incorporated - http://www.ti.com/
8 //
9 //
10 //  Redistribution and use in source and
      binary forms, with or without
11 //  modification, are permitted provided
      that the following conditions
12 //  are met:
13 //
14 //    Redistributions of source code must
      retain the above copyright
15 //    notice, this list of conditions and
      the following disclaimer.
16 //
Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the
distribution.

Neither the name of Texas Instruments Incorporated nor the names of
its contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
List of API's in the ROM that need to be patched.

For e.g. to patch ROM_UARTCharPut add the line #undef ROM_UARTCharPut

#undef ROM_ADCIntClear
#undef ROM_IntEnable
#undef ROM_IntDisable
#undef ROM_IntPendSet
#undef ROM_PRCMHibernateWakeUpGPIOSelect
#undef ROM_SDHostCardErrorMaskSet
#undef ROM_SDHostCardErrorMaskGet
#undef ROM_TimerConfigure
#undef ROM_TimerDMAEventSet
#undef ROM_TimerDMAEventGet
#undef ROM_SDHostDataNonBlockingWrite
#undef ROM_SDHostDataWrite
#undef ROM_SDHostDataRead
#undef ROM_SDHostDataNonBlockingRead
#undef ROM_PRCMSysResetCauseGet
#undef ROM_PRCMPeripheralClkEnable
#undef ROM_PRCMLPDSWakeUpGPIOSelect
#undef ROM_PRCMHibernateWakeupSourceEnable
#undef ROM_PRCMHibernateWakeupSourceDisable
#undef ROM_PRCMHibernateWakeupCauseGet
#undef ROM_PRCMHibernateIntervalSet
#undef ROM_PRCMHibernateWakeupGPIOSelect
#undef ROM_PRCMHibernateEnter
#undef ROM_PRCMSlowClkCtrGet
#undef ROM_PRCMSlowClkCtrMatchSet
#undef ROM_PRCMSlowClkCtrMatchGet
#undef ROM_PRCMOCRRegisterWrite
#undef ROM_PRCMOCRRegisterRead
#undef ROM_PRCMIntEnable
#undef ROM_PRCMIntDisable
#undef ROM_PRCMRTCInUseSet
#undef ROM_PRCMRTCInUseGet
#undef ROM_PRCMRTCSet
#undef ROM_PRCMRTCGet
#undef ROM_PRCMRTCMatchSet
#undef ROM_PRCMRTCMatchGet
#undef ROM_PRCMPeripheralClkDisable
#undef ROM_PRCMPeripheralReset
#undef ROM_PRCMPeripheralStatusGet
#undef ROM_SPIConfigSetExpClk
#undef ROM_AESDataProcess
#undef ROM_DESDataProcess
#undef ROM_I2SEnable
#undef ROM_I2SConfigSetExpClk
#undef ROM_PinConfigSet
#undef ROM_PRCMLPDSEnter
#undef ROM_PRCMCC3200MCUInit
#undef ROM_SDHostIntStatus
#undef ROM_SDHostBlockCountSet
#undef ROM_UARTModemControlSet
#undef ROM_UARTModemControlClear
#undef ROM_CameraXClkSet
#undef ROM_PRCMMCUReset
#undef ROM_SPIIdmaDisable
#undef ROM_PRCMSRAMRetentionEnable
#undef ROM_PRCMSRAMRetentionDisable
#undef ROM_PRCMDeepSleepSleepEnter
---

### sdhost.h

---

```c
//********************************************************************************
//**sdhost.h**
//**Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/**
//**Redistribution and use in source and binary forms, with or without**
//**modification, are permitted provided that the following conditions**
//**are met:**
//**Redistributions of source code must retain the above copyright**
//**notice, this list of conditions and the following disclaimer.**
//**Redistributions in binary form must**
```
reproduce the above copyright
notice, this list of conditions and
the following disclaimer in the
documentation and/or other materials
provided with the
distribution.

Neither the name of Texas Instruments
Incorporated nor the names of
its contributors may be used to
endorse or promote products derived
from this software without specific
prior written permission.

THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
#ifndef __SDHOST_H__
#define __SDHOST_H__

// If building with a C++ compiler, make all of the definitions in this header have a C binding.

#ifdef __cplusplus
extern "C"
{
#endif

// Values that can be passed to SDHostRespGet().

#define SDHOST_RESP_10 0x00000003
#define SDHOST_RESP_32 0x00000002
#define SDHOST_RESP_54 0x00000001
#define SDHOST_RESP_76 0x00000000

// Values that can be passed to
SDHostIntEnable(), SDHostIntDisable(),
// SDHostIntClear(), and returned from SDHostIntStatus().

//******************************************************************************
// Definitions
//******************************************************************************
#define SDHOST_INT_CC 0x00000001
#define SDHOST_INT_TC 0x00000002
#define SDHOST_INT_BWR 0x00000010
#define SDHOST_INT_BRR 0x00000020
#define SDHOST_INT_ERRI 0x00008000
#define SDHOST_INT_CTO 0x00010000
#define SDHOST_INT_CEB 0x00040000
#define SDHOST_INT_DTO 0x00100000
#define SDHOST_INT_DCRC 0x00200000
#define SDHOST_INT_DEB 0x00400000
#define SDHOST_INT_CERR 0x10000000
#define SDHOST_INT_BADA 0x20000000
#define SDHOST_INT_DMARD 0x40000000
#define SDHOST_INT_DMAWR 0x80000000

//******************************************************************************
// Values that can be passed to SDHostCmdSend().
//******************************************************************************
#define SDHOST_CMD_0 0x00000000
#define SDHOST_CMD_1 0x01000000
#define SDHOST_CMD_2 0x02000000
#define SDHOST_CMD_3 0x03000000
#define SDHOST_CMD_4 0x04000000
#define SDHOST_CMD_5 0x05000000
#define SDHOST_CMD_6 0x06000000
#define SDHOST_CMD_7 0x07000000
#define SDHOST_CMD_8 0x08000000
#define SDHOST_CMD_9 0x09000000
#define SDHOST_CMD_10 0x0A000000
<table>
<thead>
<tr>
<th>Line</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>#define SDHOST_CMD_11</td>
<td>0xB000000</td>
</tr>
<tr>
<td>99</td>
<td>#define SDHOST_CMD_12</td>
<td>0xC000000</td>
</tr>
<tr>
<td>100</td>
<td>#define SDHOST_CMD_13</td>
<td>0xD000000</td>
</tr>
<tr>
<td>101</td>
<td>#define SDHOST_CMD_14</td>
<td>0xE000000</td>
</tr>
<tr>
<td>102</td>
<td>#define SDHOST_CMD_15</td>
<td>0xF000000</td>
</tr>
<tr>
<td>103</td>
<td>#define SDHOST_CMD_16</td>
<td>0x10000000</td>
</tr>
<tr>
<td>104</td>
<td>#define SDHOST_CMD_17</td>
<td>0x11000000</td>
</tr>
<tr>
<td>105</td>
<td>#define SDHOST_CMD_18</td>
<td>0x12000000</td>
</tr>
<tr>
<td>106</td>
<td>#define SDHOST_CMD_19</td>
<td>0x13000000</td>
</tr>
<tr>
<td>107</td>
<td>#define SDHOST_CMD_20</td>
<td>0x14000000</td>
</tr>
<tr>
<td>108</td>
<td>#define SDHOST_CMD_21</td>
<td>0x15000000</td>
</tr>
<tr>
<td>109</td>
<td>#define SDHOST_CMD_22</td>
<td>0x16000000</td>
</tr>
<tr>
<td>110</td>
<td>#define SDHOST_CMD_23</td>
<td>0x17000000</td>
</tr>
<tr>
<td>111</td>
<td>#define SDHOST_CMD_24</td>
<td>0x18000000</td>
</tr>
<tr>
<td>112</td>
<td>#define SDHOST_CMD_25</td>
<td>0x19000000</td>
</tr>
<tr>
<td>113</td>
<td>#define SDHOST_CMD_26</td>
<td>0x1A000000</td>
</tr>
<tr>
<td>114</td>
<td>#define SDHOST_CMD_27</td>
<td>0x1B000000</td>
</tr>
<tr>
<td>115</td>
<td>#define SDHOST_CMD_28</td>
<td>0x1C000000</td>
</tr>
<tr>
<td>116</td>
<td>#define SDHOST_CMD_29</td>
<td>0x1D000000</td>
</tr>
<tr>
<td>117</td>
<td>#define SDHOST_CMD_30</td>
<td>0x1E000000</td>
</tr>
<tr>
<td>118</td>
<td>#define SDHOST_CMD_31</td>
<td>0x1F000000</td>
</tr>
<tr>
<td>119</td>
<td>#define SDHOST_CMD_32</td>
<td>0x20000000</td>
</tr>
<tr>
<td>120</td>
<td>#define SDHOST_CMD_33</td>
<td>0x21000000</td>
</tr>
<tr>
<td>121</td>
<td>#define SDHOST_CMD_34</td>
<td>0x22000000</td>
</tr>
<tr>
<td>122</td>
<td>#define SDHOST_CMD_35</td>
<td>0x23000000</td>
</tr>
<tr>
<td>123</td>
<td>#define SDHOST_CMD_36</td>
<td>0x24000000</td>
</tr>
<tr>
<td>124</td>
<td>#define SDHOST_CMD_37</td>
<td>0x25000000</td>
</tr>
<tr>
<td>125</td>
<td>#define SDHOST_CMD_38</td>
<td>0x26000000</td>
</tr>
<tr>
<td>126</td>
<td>#define SDHOST_CMD_39</td>
<td>0x27000000</td>
</tr>
<tr>
<td>127</td>
<td>#define SDHOST_CMD_40</td>
<td>0x28000000</td>
</tr>
<tr>
<td>128</td>
<td>#define SDHOST_CMD_41</td>
<td>0x29000000</td>
</tr>
<tr>
<td>129</td>
<td>#define SDHOST_CMD_42</td>
<td>0x2A000000</td>
</tr>
<tr>
<td>130</td>
<td>#define SDHOST_CMD_43</td>
<td>0x2B000000</td>
</tr>
<tr>
<td>131</td>
<td>#define SDHOST_CMD_44</td>
<td>0x2C000000</td>
</tr>
<tr>
<td>132</td>
<td>#define SDHOST_CMD_45</td>
<td>0x2D000000</td>
</tr>
<tr>
<td>133</td>
<td>#define SDHOST_CMD_46</td>
<td>0x2E000000</td>
</tr>
<tr>
<td>134</td>
<td>#define SDHOST_CMD_47</td>
<td>0x2F000000</td>
</tr>
</tbody>
</table>
135  #define SDHOST_CMD_48  0x30000000
136  #define SDHOST_CMD_49  0x31000000
137  #define SDHOST_CMD_50  0x32000000
138  #define SDHOST_CMD_51  0x33000000
139  #define SDHOST_CMD_52  0x34000000
140  #define SDHOST_CMD_53  0x35000000
141  #define SDHOST_CMD_54  0x36000000
142  #define SDHOST_CMD_55  0x37000000
143  #define SDHOST_CMD_56  0x38000000
144  #define SDHOST_CMD_57  0x39000000
145  #define SDHOST_CMD_58  0x3A000000
146  #define SDHOST_CMD_59  0x3B000000
147  #define SDHOST_CMD_60  0x3C000000
148  #define SDHOST_CMD_61  0x3D000000
149  #define SDHOST_CMD_62  0x3E000000
150  #define SDHOST_CMD_63  0x3F000000
151
152  //*************************************************************************************
153  // Values that can be logically ORed with ulCmd parameter for SDHostCmdSend().
154  //*************************************************************************************
155  #define SDHOST_MULTI_BLK  0x00000022
156  #define SDHOST_DMA_EN  0x00000001
157  #define SDHOST_WR_CMD  0x00200000
158  #define SDHOST_RD_CMD  0x00200010
159  #define SDHOST_RESP_LEN_136  0x00010000
160  #define SDHOST_RESP_LEN_48  0x00020000
161  #define SDHOST_RESP_LEN_48B  0x00030000
162
163  //*************************************************************************************
164  // API Function prototypes
165  //
extern void SDHostCmdReset(unsigned long ulBase);
extern void SDHostInit(unsigned long ulBase);
extern long SDHostCmdSend(unsigned long ulBase, unsigned long ulCmd,
                          unsigned ulArg);
extern void SDHostIntRegister(unsigned long ulBase, void (*pfnHandler)(void));
extern void SDHostIntUnregister(unsigned long ulBase);
extern void SDHostIntEnable(unsigned long ulBase, unsigned long ulIntFlags);
extern void SDHostIntDisable(unsigned long ulBase, unsigned long ulIntFlags);
extern unsigned long SDHostIntStatus(unsigned long ulBase);
extern void SDHostIntClear(unsigned long ulBase, unsigned long ulIntFlags);
extern void SDHostCardErrorMaskSet(unsigned long ulBase,
                                    unsigned long ulErrMask);
extern unsigned long SDHostCardErrorMaskGet(unsigned long ulBase);
extern void SDHostSetExpClk(unsigned long ulBase, unsigned long ulSDHostClk,
                            unsigned long ulCardClk);
extern void SDHostRespGet(unsigned long ulBase, unsigned long ulResponse[4]);
extern void SDHostBlockSizeSet(unsigned long ulBase, unsigned short ulBlkSize);
extern void SDHostBlockCountSet(unsigned long ulBase,
                                 unsigned long ulBlkCnt);
extern void SDHostRespGet(unsigned long ulBase, unsigned long ulResponse[4]);
extern void SDHostCmdSend(unsigned long ulBase, unsigned long ulCmd,
                          unsigned ulArg);
extern void SDHostIntRegister(unsigned long ulBase, void (*pfnHandler)(void));
extern void SDHostIntUnregister(unsigned long ulBase);
extern void SDHostIntEnable(unsigned long ulBase, unsigned long ulIntFlags);
extern void SDHostIntDisable(unsigned long ulBase, unsigned long ulIntFlags);
extern unsigned long SDHostIntStatus(unsigned long ulBase);
extern void SDHostIntClear(unsigned long ulBase, unsigned long ulIntFlags);
extern void SDHostCardErrorMaskSet(unsigned long ulBase,
                                    unsigned long ulErrMask);
extern unsigned long SDHostCardErrorMaskGet(unsigned long ulBase);
extern void SDHostSetExpClk(unsigned long ulBase, unsigned long ulSDHostClk,
                            unsigned long ulCardClk);
extern void SDHostRespGet(unsigned long ulBase, unsigned long ulResponse[4]);
extern void SDHostBlockSizeSet(unsigned long ulBase, unsigned short ulBlkSize);
extern void SDHostBlockCountSet(unsigned long ulBase,
                                 unsigned long ulBlkCnt);
short ulBlkCount;

extern tBoolean SDHostDataNonBlockingWrite(unsigned long ulBase,
  unsigned long ulData);

extern tBoolean SDHostDataNonBlockingRead(unsigned long ulBase,
  unsigned long *pulData);

extern void SDHostDataWrite(unsigned long ulBase, unsigned long ulData);

extern void SDHostDataRead(unsigned long ulBase, unsigned long *ulData);

//******************************************
// Mark the end of the C bindings section
// for C++ compilers.
//******************************************

#ifndef __cplusplus
}
#endif
#endif // __SDHOST_H__

Generated on Thu Feb 18 2016 13:22:02 for CC3200 Peripheral Driver
Library User's Guide by  doxygen 1.8.11
//******************************************
***********************************
///		shamd5.h
///		Defines and Macros for the SHA/MD5.
///
/// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
///
/// Redistribution and use in source and binary forms, with or without
/// modification, are permitted provided that the following conditions
/// are met:
///
/// Redistributions of source code must retain the above copyright
/// notice, this list of conditions and the following disclaimer.
///
/// Redistributions in binary form must
reproduce the above copyright

18    // notice, this list of conditions and
the following disclaimer in the
19    // documentation and/or other materials
provided with the
20    // distribution.
21    //
22    // Neither the name of Texas Instruments
Incorporated nor the names of
23    // its contributors may be used to
endorse or promote products derived
24    // from this software without specific
prior written permission.
25    //
26    // THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS
27    // "AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
28    // LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR
29    // A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE COPYRIGHT
30    // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,
31    // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT
32    // LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE,
33    // DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY
34    // THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
35    // (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE
36    // OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
37    //
#ifndef __DRIVERLIB_SHAMD5_H__
declare __DRIVERLIB_SHAMD5_H__

//******************************************
***********************************
#
#	If building with a C++ compiler, make all of the definitions in this header
#	have a C binding.
#
#
//******************************************
***********************************

#ifdef __cplusplus
extern "C"
{
#endif

//******************************************
***********************************
#
#	The following defines are used to specify the algorithm in use in the
#	SHA/MD5 module.
#
#
//******************************************
***********************************
#define SHAMD5_ALGO_MD5	0x00000018
// MD5
#define SHAMD5_ALGO_SHA1	0x0000001a
// SHA-1
#define SHAMD5_ALGO_SHA224	0x0000001c
// SHA-224
#define SHAMD5_ALGO_SHA256	0x0000001e
// SHA-256

#endif // ifndef __cplusplus
#define SHAMD5_ALGO_HMAC_MD5 0x00000000
// HMAC-MD5
#define SHAMD5_ALGO_HMAC_SHA1 0x00000002
// HMAC-SHA-1
#define SHAMD5_ALGO_HMAC_SHA224 0x00000004
// HMAC-SHA-224
#define SHAMD5_ALGO_HMAC_SHA256 0x00000006
// HMAC-SHA-256

// The following defines are used to represent the different interrupt sources in SHAMD5IntEnable(), SHAMD5IntDisable(), SHAMD5GetIntStatus(), and SHAMD5BlockOnIntStatus() functions.

#define SHAMD5_INT_CONTEXT_READY
0x00000008
#define SHAMD5_INT_PARTHASH_READY
0x00000004
#define SHAMD5_INT_INPUT_READY
0x00000002
#define SHAMD5_INT_OUTPUT_READY
0x00000001
#define SHAMD5_INT_DMA_CONTEXT_IN
0x00010000
#define SHAMD5_INT_DMA_DATA_IN
0x00020000
#define SHAMD5_INT_DMA_CONTEXT_OUT
0x00040000

//
extern void SHAMD5ConfigSet(uint32_t ui32Base, uint32_t ui32Mode);
extern bool SHAMD5DataProcess(uint32_t ui32Base, uint8_t *pui8DataSrc, uint32_t ui32DataLength, uint8_t *pui8HashResult);
extern void SHAMD5DataWrite(uint32_t ui32Base, uint8_t *pui8Src);
extern bool SHAMD5DataWriteNonBlocking(uint32_t ui32Base, uint8_t *pui8Src);
extern void SHAMD5DMADisable(uint32_t ui32Base);
extern void SHAMD5DMAEnable(uint32_t ui32Base);
extern void SHAMD5DataLengthSet(uint32_t ui32Base, uint32_t ui32Length);
extern void SHAMD5HMACKeySet(uint32_t ui32Base, uint8_t *pui8Src);
extern void SHAMD5HMACPPKeyGenerate(uint32_t ui32Base, uint8_t *pui8Key, uint8_t *pui8PPKey);
extern void SHAMD5HMACPPKeySet(uint32_t ui32Base, uint8_t *pui8Src);
extern bool SHAMD5HMACProcess(uint32_t ui32Base, uint8_t *pui8DataSrc, uint32_t ui32DataLength, uint8_t *pui8HashResult);
extern void SHAMD5IntClear(uint32_t ui32Base, uint32_t ui32IntFlags);
extern void SHAMD5IntDisable(uint32_t ui32Base, uint32_t ui32IntFlags);
extern void SHAMD5IntEnable(uint32_t ui32Base, uint32_t ui32IntFlags);
extern void SHAMD5IntRegister(uint32_t ui32Base, void(*pfnHandler)(void));
extern void SHAMD5IntUnregister(uint32_t ui32Base);
extern void SHAMD5IntStatus(uint32_t ui32Base, bool bMasked);
extern void SHAMD5ResultRead(uint32_t ui32Base, uint8_t *pui8Dest);

//******************************************
// Mark the end of the C bindings section
for C++ compilers.
//******************************************
#ifdef __cplusplus
}
#endif
#endif // __DRIVERLIB_SHAMD5_H__

//spi.h

// Defines and Macros for the SPI.

// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

// Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

// Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

// Redistributions in binary form must
reproduce the above copyright
notice, this list of conditions and
the following disclaimer in the
documentation and/or other materials
provided with the
distribution.

Neither the name of Texas Instruments
Incorporated nor the names of
its contributors may be used to
der endorse or promote products derived
from this software without specific
prior written permission.

THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
//******************************************
***********************************
#ifndef __SPI_H__
#define __SPI_H__

//******************************************
***********************************
//
// If building with a C++ compiler, make all
// of the definitions in this header
// have a C binding.
//
//******************************************
***********************************

#ifdef __cplusplus
extern "C"
{
#endif

//******************************************
***********************************
// Values that can be passed to
// SPIConfigSetExpClk() as ulMode parameter
//******************************************
***********************************
#define SPI_MODE_MASTER 0x00000000
#define SPI_MODE_SLAVE 0x00000004

//******************************************
***********************************
// Values that can be passed to
// SPIConfigSetExpClk() as ulSubMode parameter
//******************************************
***********************************
#define SPI_SUB_MODE_0 0x00000000
#define SPI_SUB_MODE_1 0x00000001
// Values that can be passed to SPIConfigSetExpClk() as ulConfigFlags parameter

// Values that can be passed to SPIFIFOEnable() and SPIFIFODisable()

// Values that can be passed to SPIDMAEnable() and SPIDMADisable()
#define SPI_RX_DMA        0x00008000
#define SPI_TX_DMA        0x00004000

// Values that can be passed to 
SPIIntEnable(), SPIIntDisable(),
// SPIIntClear() or returned from 
SPIStatus()

#define SPI_INT_DMATX     0x20000000
#define SPI_INT_DMARX     0x10000000
#define SPI_INT_EOW       0x00020000
#define SPI_INT_WKS       0x00010000
#define SPI_INT_RX_OVRFLOW 0x00000008
#define SPI_INT_RX_FULL   0x00000004
#define SPI_INT_TX_UDRFLOW 0x00000002
#define SPI_INT_TX_EMPTY  0x00000001

// Values that can be passed to 
SPITransfer()

#define SPI_CS_ENABLE     0x00000001
#define SPI_CS_DISABLE    0x00000002

// API Function prototypes

//
extern void SPIEnable(unsigned long ulBase);
extern void SPIDisable(unsigned long ulBase);
extern void SPIReset(unsigned long ulBase);
extern void SPIConfigSetExpClk(unsigned long ulBase, unsigned long ulSPIClk,
                                unsigned long ulBitRate, unsigned long ulMode,
                                unsigned long ulSubMode, unsigned long ulConfig);
extern long SPIDataGetNonBlocking(unsigned long ulBase,
                                   unsigned long *pulData);
extern void SPIDataGet(unsigned long ulBase, unsigned long *pulData);
extern long SPIDataPutNonBlocking(unsigned long ulBase,
                                   unsigned long ulData);
extern void SPIDataPut(unsigned long ulBase, unsigned long ulData);
extern void SPIFIFOEnable(unsigned long ulBase, unsigned long ulFlags);
extern void SPIFIFODisable(unsigned long ulBase, unsigned long ulFlags);
extern void SPIFIFOLevelSet(unsigned long ulBase, unsigned long ulTxLevel,
                             unsigned long ulRxLevel);
extern void SPIFIFOLevelGet(unsigned long ulBase, unsigned long *pulTxLevel,
                             unsigned long *pulRxLevel);
extern void SPIWordCountSet(unsigned long ulBase, unsigned long ulWordCount);
extern void SPIIntRegister(unsigned long ulBase, void(*pfnHandler)(void));
extern void SPIIntUnregister(unsigned long ulBase);
ulBase);
141| extern void SPIIntEnable(unsigned long ulBase, unsigned long ulIntFlags);
142| extern void SPIIntDisable(unsigned long ulBase, unsigned long ulIntFlags);
143| extern unsigned long SPIIntStatus(unsigned long ulBase, tBoolean bMasked);
144| extern void SPIIntClear(unsigned long ulBase, unsigned long ulIntFlags);
145| extern void SPIDmaEnable(unsigned long ulBase, unsigned long ulFlags);
146| extern void SPIDmaDisable(unsigned long ulBase, unsigned long ulFlags);
147| extern void SPICSEnable(unsigned long ulBase);
148| extern void SPICSDisable(unsigned long ulBase);
149| extern long SPITransfer(unsigned long ulBase, unsigned char *ucDout,
150|                        unsigned char *ucDin,
151|                        unsigned long ulSize,
152|                        unsigned long ulFlags);
153
154| //*******************************************************************************
155|*******************************************************************************
156| //
157| // Mark the end of the C bindings section for C++ compilers.
158| //
159| #ifdef __cplusplus
160| }
161| #endif
162
163| #endif // __SPI_H__
systick.h

1 //***************************************************************
   ***************************************************
2 //
3 //  systick.h
4 //
5 //  Prototypes for the SysTick driver.
6 //
7 //  Copyright (C) 2014 Texas Instruments
   Incorporated - http://www.ti.com/
8 //
9 //
10 //  Redistribution and use in source and
   binary forms, with or without
11 //  modification, are permitted provided
   that the following conditions
12 //  are met:
13 //
14 //  Redistributions of source code must
   retain the above copyright
15 //  notice, this list of conditions and
   the following disclaimer.
16 //
17 //  Redistributions in binary form must
reproduce the above copyright

notice, this list of conditions and
the following disclaimer in the
documentation and/or other materials
provided with the
distribution.

Neither the name of Texas Instruments
Incorporated nor the names of
its contributors may be used to
endorse or promote products derived
from this software without specific
prior written permission.

THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE)
 ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
 ifndef __SYSTICK_H__
define __SYSTICK_H__

 // If building with a C++ compiler, make all of the definitions in this header have a C binding.

 ifndef __cplusplus
 extern "C"
 {
 #endif

 extern void SysTickEnable(void);
 extern void SysTickDisable(void);
 extern void SysTickIntRegister(void (*pfnHandler)(void));
 extern void SysTickIntUnregister(void);
 extern void SysTickIntEnable(void);
 extern void SysTickIntDisable(void);
 extern void SysTickPeriodSet(unsigned long ulPeriod);
 extern unsigned long SysTickPeriodGet(void);
extern unsigned long SysTickValueGet(void);

//******************************************************************************
// Mark the end of the C bindings section
// for C++ compilers.
//******************************************************************************

#ifdef __cplusplus
}
#endif
#endif // __SYSTICK_H__

#endif // __cplusplus

#undef __cplusplus

#ifndef __cplusplus
#endif // __SYSTICK_H__
timer.h

1 //*******************************************************************************
2 *******************************************************************************
3 //
4 // timer.h
5 //
6 // Prototypes for the timer module
7 //
8 // Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
9 //
10 // Redistribution and use in source and binary forms, with or without
11 // modification, are permitted provided that the following conditions
12 // are met:
13 //
14 // Redistributions of source code must retain the above copyright
15 // notice, this list of conditions and the following disclaimer.
16 //
17 // Redistributions in binary form must
reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the
distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//******************************************

ifndef __TIMER_H__
define __TIMER_H__
//******************************************

// If building with a C++ compiler, make all of the definitions in this header have a C binding.
//******************************************

#ifdef __cplusplus
extern "C"
{
define TIMER_CFG_ONE_SHOT							0x00000021
// Full-width one-shot timer
define TIMER_CFG_ONE_SHOT_UP				0x00000031
// Full-width one-shot up-count
define TIMER_CFG_PERIODIC							0x00000022
// Full-width periodic timer

define TIMER_CFG_PERIODIC_UP				0x00000032
// Full-width periodic up-count

define TIMER_CFG_PERIODIC_DOWN				0x00000002
// Full-width periodic down-count

define TIMER_CFG_PERIODIC_NONPERSISTENT		0x00000001
// Full-width periodic non-persistent
#endif

// Values that can be passed to TimerConfigure as the ulConfig parameter.

//******************************************
#define TIMER_CFG_PERIODIC_UP 0x00000032
   // Full-width periodic up-count

   // timer

#define TIMER_CFG_SPLIT_PAIR 0x04000000
   // Two half-width timers

#define TIMER_CFG_A_ONE_SHOT 0x00000021
   // Timer A one-shot timer

#define TIMER_CFG_A_ONE_SHOT_UP 0x00000031
   // Timer A one-shot up-count timer

#define TIMER_CFG_A_PERIODIC 0x00000022
   // Timer A periodic timer

#define TIMER_CFG_A_PERIODIC_UP 0x00000032
   // Timer A periodic up-count timer

#define TIMER_CFG_A_CAP_COUNT 0x00000003
   // Timer A event counter

#define TIMER_CFG_A_CAP_COUNT_UP 0x00000013
   // Timer A event up-counter

#define TIMER_CFG_A_CAP_TIME 0x00000007
   // Timer A event timer

#define TIMER_CFG_A_CAP_TIME_UP 0x00000017
   // Timer A event up-count timer

#define TIMER_CFG_A_PWM 0x0000000A
   // Timer A PWM output

#define TIMER_CFG_B_ONE_SHOT 0x00002100
   // Timer B one-shot timer

#define TIMER_CFG_B_ONE_SHOT_UP 0x00003100
   // Timer B one-shot up-count timer

#define TIMER_CFG_B_PERIODIC 0x00002200
   // Timer B periodic timer

#define TIMER_CFG_B_PERIODIC_UP 0x00003200
   // Timer B periodic up-count timer

#define TIMER_CFG_B_CAP_COUNT 0x00000300
   // Timer B event counter

#define TIMER_CFG_B_CAP_COUNT_UP 0x00001300
   // Timer B event up-counter


```c
# define TIMER_CFG_B_CAP_TIME 0x00000700
   // Timer B event timer
# define TIMER_CFG_B_CAP_TIME_UP 0x00001700
   // Timer B event up-count timer
# define TIMER_CFG_B_PWM 0x00000A00
   // Timer B PWM output

// ******************************************
// Values that can be passed to
// TimerIntEnable, TimerIntDisable, and
// TimerIntClear as the ulIntFlags
// parameter, and returned from TimerIntStatus.
// ******************************************

# define TIMER_TIMB_DMA 0x00000200
   // TimerB DMA Done interrupt
# define TIMER_TIMB_MATCH 0x00000800
   // TimerB match interrupt
# define TIMER_CAPB_EVENT 0x00000400
   // CaptureB event interrupt
# define TIMER_CAPB_MATCH 0x00000200
   // CaptureB match interrupt
# define TIMER_TIMB_TIMEOUT 0x00000100
   // TimerB time out interrupt
# define TIMER_TIMA_DMA 0x00000020
   // TimerA DMA Done interrupt
# define TIMER_TIMA_MATCH 0x00000010
   // TimerA match interrupt
# define TIMER_CAPA_EVENT 0x00000004
   // CaptureA event interrupt
# define TIMER_CAPA_MATCH 0x00000002
   // CaptureA match interrupt
# define TIMER_TIMA_TIMEOUT 0x00000001
```
// TimerA time out interrupt

//**************************************************************
// Values that can be passed to TimerControlEvent as the ulEvent parameter.
//**************************************************************
#define TIMER_EVENT_POS_EDGE 0x00000000
  // Count positive edges
#define TIMER_EVENT_NEG_EDGE 0x00000404
  // Count negative edges
#define TIMER_EVENT_BOTH_EDGES 0x00000C0C
  // Count both edges

//**************************************************************
// Values that can be passed to most of the timer APIs as the ulTimer parameter.
//**************************************************************
#define TIMER_A 0x000000ff
  // Timer A
#define TIMER_B 0x0000ff00
  // Timer B
#define TIMER_BOTH 0x0000ffff
  // Timer Both

//**************************************************************
// Values that can be passed to TimerSynchronize as the ulTimers parameter.

// ****************************************
// Values that can be passed to TimerDMAEventSet() or returned from TimerDMAEventGet().

#define TIMER_0A_SYNC 0x00000001 // Synchronize Timer 0A
#define TIMER_0B_SYNC 0x00000002 // Synchronize Timer 0B
#define TIMER_1A_SYNC 0x00000004 // Synchronize Timer 1A
#define TIMER_1B_SYNC 0x00000008 // Synchronize Timer 1B
#define TIMER_2A_SYNC 0x00000010 // Synchronize Timer 2A
#define TIMER_2B_SYNC 0x00000020 // Synchronize Timer 2B
#define TIMER_3A_SYNC 0x00000040 // Synchronize Timer 3A
#define TIMER_3B_SYNC 0x00000080 // Synchronize Timer 3B

#define TIMER_DMA_MODEMATCH_B 0x00000800
#define TIMER_DMA_CAPEVENT_B 0x00000400
#define TIMER_DMA_CAPMATCH_B 0x00000200
#define TIMER_DMA_TIMEOUT_B 0x00000100
#define TIMER_DMA_MODEMATCH_A 0x00000010
#define TIMER_DMA_CAPEVENT_A 0x00000004
#define TIMER_DMA_CAPMATCH_A 0x00000002
#define TIMER_DMA_TIMEOUT_A 0x00000001

// Prototypes for the APIs.
extern void TimerEnable(unsigned long ulBase, unsigned long ulTimer);
extern void TimerDisable(unsigned long ulBase, unsigned long ulTimer);
extern void TimerConfigure(unsigned long ulBase, unsigned long ulConfig);
extern void TimerControlLevel(unsigned long ulBase, unsigned long ulTimer,
                                tBoolean bInvert);
extern void TimerControlEvent(unsigned long ulBase, unsigned long ulTimer,
                                unsigned long ulEvent);
extern void TimerControlStall(unsigned long ulBase, unsigned long ulTimer,
                               tBoolean bStall);
extern void TimerPrescaleSet(unsigned long ulBase, unsigned long ulTimer,
                              unsigned long ulValue);
extern unsigned long TimerPrescaleGet(unsigned long ulBase,
                                       unsigned long ulTimer);
extern void TimerPrescaleMatchSet(unsigned long ulBase, unsigned long ulTimer,
    unsigned long ulValue);

extern unsigned long TimerPrescaleMatchGet(unsigned long ulBase,
    unsigned long ulTimer);

extern void TimerLoadSet(unsigned long ulBase, unsigned long ulTimer,
    unsigned long ulValue);

extern unsigned long TimerLoadGet(unsigned long ulBase, unsigned long ulTimer);

extern unsigned long TimerValueGet(unsigned long ulBase,
    unsigned long ulTimer);

extern void TimerValueSet(unsigned long ulBase, unsigned long ulTimer,
    unsigned long ulValue);

extern void TimerMatchSet(unsigned long ulBase, unsigned long ulTimer,
    unsigned long ulValue);

extern unsigned long TimerMatchGet(unsigned long ulBase,
    unsigned long ulTimer);

extern void TimerIntRegister(unsigned long ulBase, unsigned long ulTimer,
    void (*pfnHandler)(void));

extern void TimerIntUnregister(unsigned long ulBase, unsigned long ulTimer);
extern void TimerIntEnable(unsigned long ulBase, unsigned long ulIntFlags);

extern void TimerIntDisable(unsigned long ulBase, unsigned long ulIntFlags);

extern unsigned long TimerIntStatus(unsigned long ulBase, tBoolean bMasked);

extern void TimerIntClear(unsigned long ulBase, unsigned long ulIntFlags);

extern void TimerDMAEventSet(unsigned long ulBase, unsigned long ulDMAEvent);

extern unsigned long TimerDMAEventGet(unsigned long ulBase);

//******************************************************************************
****
// Mark the end of the C bindings section for C++ compilers.
//******************************************************************************
uart.h

/***************************************************************************/
/***************************************************************************/

// uart.h

// Defines and Macros for the UART.

// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:

// Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.

// Redistributions in binary form must
reproduce the above copyright

// notice, this list of conditions and
the following disclaimer in the
documentation and/or other materials
provided with the
distribution.

// Neither the name of Texas Instruments
Incorporated nor the names of
its contributors may be used to
endorse or promote products derived
from this software without specific
prior written permission.

// THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```c
#ifndef __UART_H__
#define __UART_H__

// If building with a C++ compiler, make all of the definitions in this header have a C binding.

#if defined __cplusplus
extern "C"
{
#endif

#define UART_INT_DMATX 0x20000 // DMA Tx Done interrupt Mask
#define UART_INT_DMARX 0x10000 // DMA Rx Done interrupt Mask
#define UART_INT_EOT 0x800   // End of transfer interrupt Mask

} //
```

// Values that can be passed to UARTIntEnable, UARTIntDisable, and UARTIntClear as the ulIntFlags parameter, and returned from UARTIntStatus.

```
```c
#define UART_INT_OE 0x400
// Overrun Error Interrupt Mask
#define UART_INT_BE 0x200
// Break Error Interrupt Mask
#define UART_INT_PE 0x100
// Parity Error Interrupt Mask
#define UART_INT_FE 0x080
// Framing Error Interrupt Mask
#define UART_INT_RT 0x040
// Receive Timeout Interrupt Mask
#define UART_INT_TX 0x020
// Transmit Interrupt Mask
#define UART_INT_RX 0x010
// Receive Interrupt Mask
#define UART_INT_CTS 0x002
// CTS Modem Interrupt Mask

// Values that can be passed to UARTConfigSetExpClk as the ulConfig parameter
// and returned by UARTConfigGetExpClk in the pulConfig parameter.
// Additionally, the UART_CONFIG_PAR_* subset can be passed to
// UARTParityModeSet as the ulParity parameter, and are returned by
// UARTParityModeGet.

#define UART_CONFIG_WLEN_MASK 0x00000060
// Mask for extracting word length
#define UART_CONFIG_WLEN_8 0x00000060
// 8 bit data
```
#define UART_CONFIG_WLEN_7 0x00000040  // 7 bit data
#define UART_CONFIG_WLEN_6 0x00000020  // 6 bit data
#define UART_CONFIG_WLEN_5 0x00000000  // 5 bit data
#define UART_CONFIG_STOP_MASK 0x00000008  // Mask for extracting stop bits
#define UART_CONFIG_STOP_ONE 0x00000000  // One stop bit
#define UART_CONFIG_STOP_TWO 0x00000008  // Two stop bits
#define UART_CONFIG_PAR_MASK 0x00000086  // Mask for extracting parity
#define UART_CONFIG_PAR_NONE 0x00000000  // No parity
#define UART_CONFIG_PAR_EVEN 0x00000006  // Even parity
#define UART_CONFIG_PAR_ODD 0x00000002  // Odd parity
#define UART_CONFIG_PAR_ONE 0x00000082  // Parity bit is one
#define UART_CONFIG_PAR_ZERO 0x00000086  // Parity bit is zero

// ******************************************
// Values that can be passed to UARTFIFO\nLevelSet as the ulTxLevel parameter and
// returned by UARTFIFO\nLevelGet in the pulTxLevel.
// ******************************************
#define UART_FIFO_TX1_8 0x00000000
// Transmit interrupt at 1/8 Full
#define UART_FIFO_TX2_8 0x00000001
// Transmit interrupt at 1/4 Full
#define UART_FIFO_TX4_8 0x00000002
// Transmit interrupt at 1/2 Full
#define UART_FIFO_TX6_8 0x00000003
// Transmit interrupt at 3/4 Full
#define UART_FIFO_TX7_8 0x00000004
// Transmit interrupt at 7/8 Full

// Values that can be passed to UARTFIFOLevelSet as the ulRxLevel parameter and
// returned by UARTFIFOLevelGet in the pulRxLevel.

#define UART_FIFO_RX1_8 0x00000000
// Receive interrupt at 1/8 Full
#define UART_FIFO_RX2_8 0x00000008
// Receive interrupt at 1/4 Full
#define UART_FIFO_RX4_8 0x00000010
// Receive interrupt at 1/2 Full
#define UART_FIFO_RX6_8 0x00000018
// Receive interrupt at 3/4 Full
#define UART_FIFO_RX7_8 0x00000020
// Receive interrupt at 7/8 Full

// Values that can be passed to UARTDMAEnable() and UARTDMADisable().
// Stop DMA receive if UART error
#define UART_DMA_ERR_RXSTOP 0x00000004
// Enable DMA for transmit
#define UART_DMA_TX 0x00000002
// Enable DMA for receive
#define UART_DMA_RX 0x00000001

// Values returned from UARTRxErrorGet().

#define UART_RXERROR_OVERRUN 0x00000008
#define UART_RXERROR_BREAK 0x00000004
#define UART_RXERROR_PARITY 0x00000002
#define UART_RXERROR_FRAMING 0x00000001

// Values that can be passed to
// UARTModemControlSet() and
// UARTModemControlClear() or
// returned from UARTModemControlGet().

#define UART_OUTPUT_RTS 0x00000080
// Values that can be returned from UARTModemStatusGet().

//*******************************************************************************
//*******************************************************************************
#define UART_INPUT_CTS 0x00000001

//*******************************************************************************
//*******************************************************************************
// Values that can be passed to UARTFlowControl() or returned from UARTFlowControlGet().

//*******************************************************************************
//*******************************************************************************
#define UART_FLOWCONTROL_TX 0x00008000
#define UART_FLOWCONTROL_RX 0x00004000
#define UART_FLOWCONTROL_NONE 0x00000000

//*******************************************************************************
//*******************************************************************************
// Values that can be passed to UARTTxIntModeSet() or returned from UARTTxIntModeGet().

//*******************************************************************************
//*******************************************************************************
#define UART_TXINT_MODE_FIFO 0x00000000
#define UART_TXINT_MODE_EOT 0x00000010

//*******************************************************************************
//*******************************************************************************
// API Function prototypes
extern void UARTParityModeSet(unsigned long ulBase, unsigned long ulParity);

extern unsigned long UARTParityModeGet(unsigned long ulBase);

extern void UARTFIFOLevelSet(unsigned long ulBase, unsigned long ulTxLevel,
   unsigned long ulRxLevel);

extern void UARTFIFOLevelGet(unsigned long ulBase, unsigned long *pulTxLevel,
   unsigned long *pulRxLevel);

extern void UARTConfigSetExpClk(unsigned long ulBase, unsigned long ulUARTClk,
   unsigned long ulBaud, unsigned long ulConfig);

extern void UARTConfigGetExpClk(unsigned long ulBase, unsigned long ulUARTClk,
   unsigned long *pulBaud,
   unsigned long *pulConfig);

extern void UARTEnable(unsigned long ulBase);

extern void UARTDisable(unsigned long ulBase);

extern void UARTFIFOEnable(unsigned long ulBase);

extern void UARTFIFODisable(unsigned long ulBase);

extern tBoolean UARTCharsAvail(unsigned long ulBase);

extern tBoolean UARTSpaceAvail(unsigned long ulBase);
197| extern long UARTCharGetNonBlocking(unsigned long ulBase);
198| extern long UARTCharGet(unsigned long ulBase);
199| extern tBoolean UARTCharPutNonBlocking(unsigned long ulBase,
200|   unsigned char ucData);
201| extern void UARTCharPut(unsigned long ulBase, unsigned char ucData);
202| extern void UARTBreakCtl(unsigned long ulBase, tBoolean bBreakState);
203| extern tBoolean UARTBusy(unsigned long ulBase);
204| extern void UARTIntRegister(unsigned long ulBase, void(*pfnHandler)(void));
205| extern void UARTIntUnregister(unsigned long ulBase);
206| extern void UARTIntEnable(unsigned long ulBase, unsigned long ulIntFlags);
207| extern void UARTIntDisable(unsigned long ulBase, unsigned long ulIntFlags);
208| extern unsigned long UARTIntStatus(unsigned long ulBase, tBoolean bMasked);
209| extern void UARTIntClear(unsigned long ulBase, unsigned long ulIntFlags);
210| extern void UARTDMAEnable(unsigned long ulBase, unsigned long ulDMAFlags);
211| extern void UARTDMADisable(unsigned long ulBase, unsigned long ulDMAFlags);
212| extern unsigned long UARTRxErrorGet(unsigned long ulBase);
213| extern void UARTRxErrorClear(unsigned long ulBase);
214| extern void UARTModemControlSet(unsigned long ulBase,
215|   unsigned...
long ulControl);

extern void UARTModemControlClear(unsigned long ulBase,
        unsigned long ulControl);

extern unsigned long UARTModemControlGet(unsigned long ulBase);

extern unsigned long UARTModemStatusGet(unsigned long ulBase);

extern void UARTFlowControlSet(unsigned long ulBase, unsigned long ulMode);

extern unsigned long UARTFlowControlGet(unsigned long ulBase);

extern void UARTTxIntModeSet(unsigned long ulBase, unsigned long ulMode);

extern unsigned long UARTTxIntModeGet(unsigned long ulBase);

//*******************************************************************************
//                        Mark the end of the C bindings section
// for C++ compilers.
//*******************************************************************************

#ifdef __cplusplus
}
#endif
#endif // __UART_H__

// Prototypes and macros for the uDMA controller.

// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

// Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

// Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

//
Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
//******************************************
***********************************
//
#ifndef __UDMA_H__
#define __UDMA_H__
//
// If building with a C++ compiler, make all of the definitions in this header have a C binding.
//
//******************************************
***********************************
#endif
extern "C"
{
#endif
//
//******************************************
***********************************
//
// A structure that defines an entry in the channel control table. These fields are used by the uDMA controller and normally it is not necessary for software to directly read or write fields in the table.
typedef struct {
    volatile void *pvSrcEndAddr;
    volatile void *pvDstEndAddr;
    volatile unsigned long ulControl;
    volatile unsigned long ulSpare;
} tDMAControlTable;

#define uDMATaskStructEntry(ulTransferCount,
ulItemSize, ulSrcIncrement, pvSrcAddr, ulDstIncrement, pvDstAddr, ulArbSize, ulMode)
{

(((ulSrcIncrement) == UDMA_SRC_INC_NONE) ? (void *)(pvSrcAddr) :
 ((void *)((unsigned char *) pvSrcAddr)[((ulTransferCount) << 26) - 1])),

(((ulDstIncrement) == UDMA_DST_INC_NONE) ? (void *)(pvDstAddr) :
 (void *)((unsigned char *) pvDstAddr)[((ulTransferCount) << 30) - 1]]),

((ulSrcIncrement) >> 26) - 1])]])),  

((ulDstIncrement) >> 30) - 1]]),

(ulSrcIncrement) | (ulDstIncrement) | (ulItemSize) | (ulArbSize) | 

(((ulTransferCount) - 1) << 4) | 

(((ulMode) == UDMA_MODE_MEM_SCATTER_GATHER) || 

((ulMode) ==
UDMA_MODE_PER_SCATTER_GATHER)) ?
\  
180| (ulMode) | UDMA_MODE_ALT_SELECT : (ulMode)), 0
\  
181| }
182
183| .isSuccess() ||
184| // Close the Doxygen group.
185| //
186| //Flags that can be passed to
187| //uDMAChannelAttributeEnable(),
188| //uDMAChannelAttributeDisable(), and
189| // returned from uDMAChannelAttributeGet().
190| //
191| //DMA control modes that can be passed to
192| //uDMAModeSet() and returned
193| //uDMAModeGet().
194| //
195| #define UDMA_ATTR_USEBURST 0x00000001
196| #define UDMA_ATTR_ALTSELECT 0x00000002
197| #define UDMA_ATTR_HIGH_PRIORITY 0x00000004
198| #define UDMA_ATTR_REQMASK 0x00000008
199| #define UDMA_ATTR_ALL 0x0000000F
200| #define UDMA_ATTR_ALL 0x0000000F
201| //
202| //
// ******************************************
// 
// Flags to be OR'd with the channel ID to indicate if the primary or alternate control structure should be used.
// 
// ******************************************

#define UDMA_MODE_STOP 0x00000000
#define UDMA_MODE_BASIC 0x00000001
#define UDMA_MODE_AUTO 0x00000002
#define UDMA_MODE_PINGPONG 0x00000003
#define UDMA_MODE_MEM_SCATTER_GATHER 0x00000004
#define UDMA_MODE_PER_SCATTER_GATHER 0x00000006
#define UDMA_MODE_ALT_SELECT 0x00000001

// ******************************************
// 
// uDMA interrupt sources, to be passed to uDMAIntRegister() and uDMAIntUnregister().
// 
// ******************************************

#define UDMA_INT_SW INT_UDMA
```c
#define UDMA_INT_ERR        INT_UDMAERR

//**************************************************************************
//**************************************************************************

// Channel configuration values that can be passed to uDMAControlSet().

//**************************************************************************
//**************************************************************************
#define UDMA_DST_INC_8       0x00000000
#define UDMA_DST_INC_16      0x40000000
#define UDMA_DST_INC_32      0x80000000
#define UDMA_DST_INC_NONE    0xc0000000
#define UDMA_SRC_INC_8       0x00000000
#define UDMA_SRC_INC_16      0x04000000
#define UDMA_SRC_INC_32      0x08000000
#define UDMA_SRC_INC_NONE    0x0c000000
#define UDMA_SIZE_8          0x00000000
#define UDMA_SIZE_16         0x11000000
#define UDMA_SIZE_32         0x22000000
#define UDMA_SRC_INC_8       0x00000000
#define UDMA_SRC_INC_16      0x00004000
#define UDMA_SRC_INC_32      0x00008000
#define UDMA_SRC_INC_NONE    0x0000c000
#define UDMA_ARB_1           0x00000000
#define UDMA_ARB_2           0x00004000
#define UDMA_ARB_4           0x00008000
#define UDMA_ARB_8           0x0000c000
#define UDMA_ARB_16          0x00010000
#define UDMA_ARB_32          0x00014000
#define UDMA_ARB_64          0x00018000
#define UDMA_ARB_128         0x0001c000
#define UDMA_ARB_256         0x00020000
#define UDMA_ARB_512         0x00024000
#define UDMA_ARB_1024        0x00028000
#define UDMA_NEXT_USEBURST   0x00000008
```
// Values that can be passed to uDMAChannelAssign() to select peripheral mapping for each channel. The channels named RESERVED may be assigned to a peripheral in future parts.

// Channel 0
#define UDMA_CH0_TIMERA0_A 0x00000000
#define UDMA_CH0_SHAMD5_CIN 0x00010000
#define UDMA_CH0_SW 0x00030000

// Channel 1
#define UDMA_CH1_TIMERA0_B 0x00000001
#define UDMA_CH1_SHAMD5_DIN 0x00010001
#define UDMA_CH1_SW 0x00030001

// Channel 2
#define UDMA_CH2_TIMERA1_A 0x00000002
#define UDMA_CH2_SHAMD5_COUT 0x00010002
#define UDMA_CH2_SW 0x00030002

// Channel 3
#define UDMA_CH3_TIMERA1_B 0x00000003
#define UDMA_CH3_DES_CIN 0x00010003
```c
#define UDMA_CH3_SW 0x00030003
```

```
```

```
#define UDMA_CH4_TIMERA2_A 0x00000004
#define UDMA_CH4_DES_DIN 0x00010004
#define UDMA_CH4_I2S_RX 0x00020004
#define UDMA_CH4_SW 0x00030004
```

```
```

```
#define UDMA_CH5_TIMERA2_B 0x00000005
#define UDMA_CH5_DES_DOUT 0x00010005
#define UDMA_CH5_I2S_TX 0x00020005
#define UDMA_CH5_SW 0x00030005
```

```
```

```
#define UDMA_CH6_TIMERA3_A 0x00000006
#define UDMA_CH6_GSPI_RX 0x00010006
#define UDMA_CH6_GPIOA2 0x00020006
#define UDMA_CH6_SW 0x00030006
```

```
```

```
#define UDMA_CH7_TIMERA3_B 0x00000007
#define UDMA_CH7_GSPI_TX 0x00010007
#define UDMA_CH7_GPIOA3 0x00020007
#define UDMA_CH7_SW 0x00030007
```

```
```

```
#define UDMA_CH8_TIMERA4_A 0x00000008
#define UDMA_CH8_DES_DOUT 0x00010008
#define UDMA_CH8_I2S_TX 0x00020008
#define UDMA_CH8_SW 0x00030008
```

```
```

```
#define UDMA_CH9_TIMERA4_B 0x00000009
#define UDMA_CH9_GSPI_RX 0x00010009
#define UDMA_CH9_GPIOA4 0x00020009
#define UDMA_CH9_SW 0x00030009
```

```
```

```
#define UDMA_CH10_TIMERA5_A 0x0000000A
#define UDMA_CH10_GSPI_TX 0x0001000A
#define UDMA_CH10_GPIOA5 0x0002000A
#define UDMA_CH10_SW 0x0003000A
```

```
```

```
#define UDMA_CH11_TIMERA5_B 0x0000000B
#define UDMA_CH11_GSPI_RX 0x0001000B
#define UDMA_CH11_GPIOA6 0x0002000B
#define UDMA_CH11_SW 0x0003000B
```

```
```

```
#define UDMA_CH12_TIMERA6_A 0x0000000C
#define UDMA_CH12_GSPI_TX 0x0001000C
#define UDMA_CH12_GPIOA7 0x0002000C
#define UDMA_CH12_SW 0x0003000C
```

```
```

```
#define UDMA_CH13_TIMERA6_B 0x0000000D
#define UDMA_CH13_GSPI_RX 0x0001000D
#define UDMA_CH13_GPIOA8 0x0002000D
#define UDMA_CH13_SW 0x0003000D
```

```
```

```
#define UDMA_CH14_TIMERA7_A 0x0000000E
#define UDMA_CH14_GSPI_TX 0x0001000E
#define UDMA_CH14_GPIOA9 0x0002000E
#define UDMA_CH14_SW 0x0003000E
```

```
```

```
#define UDMA_CH15_TIMERA7_B 0x0000000F
#define UDMA_CH15_GSPI_RX 0x0001000F
#define UDMA_CH15_GPIOA10 0x0002000F
#define UDMA_CH15_SW 0x0003000F
```

```
```

```
#define UDMA_CH16_TIMERA8_A 0x00000010
#define UDMA_CH16_GSPI_TX 0x00010010
#define UDMA_CH16_GPIOA11 0x00020010
#define UDMA_CH16_SW 0x00030010
```

```
```

```
#define UDMA_CH17_TIMERA8_B 0x00000011
#define UDMA_CH17_GSPI_RX 0x00010011
#define UDMA_CH17_GPIOA12 0x00020011
#define UDMA_CH17_SW 0x00030011
```

```
```

```
#define UDMA_CH18_TIMERA9_A 0x00000012
#define UDMA_CH18_GSPI_TX 0x00010012
#define UDMA_CH18_GPIOA13 0x00020012
#define UDMA_CH18_SW 0x00030012
```

```
```

```
#define UDMA_CH19_TIMERA9_B 0x00000013
#define UDMA_CH19_GSPI_RX 0x00010013
#define UDMA_CH19_GPIOA14 0x00020013
#define UDMA_CH19_SW 0x00030013
```

```
```

```
#define UDMA_CH20_TIMERA10_A 0x00000014
#define UDMA_CH20_GSPI_TX 0x00010014
#define UDMA_CH20_GPIOA15 0x00020014
#define UDMA_CH20_SW 0x00030014
```

```
```

```
#define UDMA_CH21_TIMERA10_B 0x00000015
#define UDMA_CH21_GSPI_RX 0x00010015
#define UDMA_CH21_GPIOA16 0x00020015
#define UDMA_CH21_SW 0x00030015
```

```
```

```
#define UDMA_CH22_TIMERA11_A 0x00000016
#define UDMA_CH22_GSPI_TX 0x00010016
#define UDMA_CH22_GPIOA17 0x00020016
#define UDMA_CH22_SW 0x00030016
```

```
```

```
#define UDMA_CH23_TIMERA11_B 0x00000017
#define UDMA_CH23_GSPI_RX 0x00010017
#define UDMA_CH23_GPIOA18 0x00020017
#define UDMA_CH23_SW 0x00030017
```

```
```

```
#define UDMA_CH24_TIMERA12_A 0x00000018
#define UDMA_CH24_GSPI_TX 0x00010018
#define UDMA_CH24_GPIOA19 0x00020018
#define UDMA_CH24_SW 0x00030018
```

```
```

```
#define UDMA_CH25_TIMERA12_B 0x00000019
#define UDMA_CH25_GSPI_RX 0x00010019
#define UDMA_CH25_GPIOA20 0x00020019
#define UDMA_CH25_SW 0x00030019
```

```
```

```
#define UDMA_CH26_TIMERA13_A 0x0000001A
#define UDMA_CH26_GSPI_TX 0x0001001A
#define UDMA_CH26_GPIOA21 0x0002001A
#define UDMA_CH26_SW 0x0003001A
```

```
```

```
#define UDMA_CH27_TIMERA13_B 0x0000001B
#define UDMA_CH27_GSPI_RX 0x0001001B
#define UDMA_CH27_GPIOA22 0x0002001B
#define UDMA_CH27_SW 0x0003001B
```

```
```

```
#define UDMA_CH28_TIMERA14_A 0x0000001C
#define UDMA_CH28_GSPI_TX 0x0001001C
#define UDMA_CH28_GPIOA23 0x0002001C
#define UDMA_CH28_SW 0x0003001C
```

```
```

```
#define UDMA_CH29_TIMERA14_B 0x0000001D
#define UDMA_CH29_GSPI_RX 0x0001001D
#define UDMA_CH29_GPIOA24 0x0002001D
#define UDMA_CH29_SW 0x0003001D
```

```
```

```
#define UDMA_CH30_TIMERA15_A 0x0000001E
#define UDMA_CH30_GSPI_TX 0x0001001E
#define UDMA_CH30_GPIOA25 0x0002001E
#define UDMA_CH30_SW 0x0003001E
```

```
```

```
#define UDMA_CH31_TIMERA15_B 0x0000001F
#define UDMA_CH31_GSPI_RX 0x0001001F
#define UDMA_CH31_GPIOA26 0x0002001F
#define UDMA_CH31_SW 0x0003001F
```
```
#define UDMA_CH8_UARTA0_RX  0x00000008
#define UDMA_CH8_TIMERA0_A  0x00010008
#define UDMA_CH8_TIMERA2_A  0x00020008
#define UDMA_CH8_SW        0x00030008

//
// Channel 9
//
#define UDMA_CH9_UARTA0_TX  0x00000009
#define UDMA_CH9_TIMERA0_B  0x00010009
#define UDMA_CH9_TIMERA2_B  0x00020009
#define UDMA_CH9_SW        0x00030009

//
// Channel 10
//
#define UDMA_CH10_UARTA1_RX 0x0000000A
#define UDMA_CH10_TIMERA1_A 0x0001000A
#define UDMA_CH10_TIMERA3_A 0x0002000A
#define UDMA_CH10_SW        0x0003000A

//
// Channel 11
//
#define UDMA_CH11_UARTA1_TX 0x0000000B
#define UDMA_CH11_TIMERA1_B 0x0001000B
#define UDMA_CH11_TIMERA3_B 0x0002000B
#define UDMA_CH11_SW        0x0003000B

//
// Channel 12
//
#define UDMA_CH12_LSPI_RX   0x0000000C
```c
#define UDMA_CH12_SW 0x0003000C

// Channel 13
#define UDMA_CH13_LSPI_TX 0x0000000D
#define UDMA_CH13_SW 0x0003000D

// Channel 14
#define UDMA_CH14_ADC_CH0 0x0000000E
#define UDMA_CH14_SDHOST_RX 0x0002000E
#define UDMA_CH14_SW 0x0003000E

// Channel 15
#define UDMA_CH15_ADC_CH1 0x0000000F
#define UDMA_CH15_SDHOST_TX 0x0002000F
#define UDMA_CH15_SW 0x0003000F

// Channel 16
#define UDMA_CH16_ADC_CH2 0x00000010
#define UDMA_CH16_TIMERA2_A 0x00010010
#define UDMA_CH16_SW 0x00030010

// Channel 17
```

```c
#define UDMA_CH17_ADC_CH3 0x00000011
#define UDMA_CH17_TIMERA2_B 0x00010011
#define UDMA_CH17_SW 0x00030011

//
// Channel 18
//
#define UDMA_CH18_GPIOA0 0x00000012
#define UDMA_CH18_AES_CIN 0x00010012
#define UDMA_CH18_I2S_RX 0x00020012
#define UDMA_CH18_SW 0x00030012

//
// Channel 19
//
#define UDMA_CH19_GPIOA1 0x00000013
#define UDMA_CH19_AES_COUT 0x00010013
#define UDMA_CH19_I2S_TX 0x00020013
#define UDMA_CH19_SW 0x00030013

//
// Channel 20
//
#define UDMA_CH20_GPIOA2 0x00000014
#define UDMA_CH20_AES_DIN 0x00010014
#define UDMA_CH20_SW 0x00030014

//
// Channel 21
//
#define UDMA_CH21_GPIOA3 0x00000015
#define UDMA_CH21_AES_DOUT 0x00010015
#define UDMA_CH21_SW 0x00030015
```
// Channel 22
#define UDMA_CH22_CAMERA 0x00000016
#define UDMA_CH22_GPIOA4 0x00010016
#define UDMA_CH22_SW 0x00030016

// Channel 23
#define UDMA_CH23_SDHOST_RX 0x00000017
#define UDMA_CH23_TIMERA3_A 0x00010017
#define UDMA_CH23_TIMERA2_A 0x00020017
#define UDMA_CH23_SW 0x00030017

// Channel 24
#define UDMA_CH24_SDHOST_TX 0x00000018
#define UDMA_CH24_TIMERA3_B 0x00010018
#define UDMA_CH24_TIMERA2_B 0x00020018
#define UDMA_CH24_SW 0x00030018

// Channel 25
#define UDMA_CH25_SSPI_RX 0x00000019
#define UDMA_CH25_I2CA0_RX 0x00010019
#define UDMA_CH25_SW 0x00030019

// Channel 26
//
#define UDMA_CH26_SSPI_TX 0x0000001A
#define UDMA_CH26_I2CA0_TX 0x0001001A
#define UDMA_CH26_SW 0x0003001A

// Channel 27
#define UDMA_CH27_GPIOA0 0x0001001B
#define UDMA_CH27_SW 0x0003001B

// Channel 28
#define UDMA_CH28_GPIOA1 0x0001001C
#define UDMA_CH28_SW 0x0003001C

// Channel 29
#define UDMA_CH29_GPIOA4 0x0000001D
#define UDMA_CH29_SW 0x0003001D

// Channel 30
#define UDMA_CH30_GSPI_RX 0x0000001E
#define UDMA_CH30_SDHOST_RX 0x0001001E
#define UDMA_CH30_I2CA0_RX 0x0002001E
#define UDMA_CH30_SW 0x0003001E

//
// Channel 31
#define UDMA_CH31_GSPI_TX 0x0000001F
#define UDMA_CH31_SDHOST_TX 0x0001001F
#define UDMA_CH31_I2CA0_RX 0x0002001F
#define UDMA_CH31_SW 0x0003001F

//******************************************
// The following are defines for the Micro Direct Memory Access (uDMA) offsets.
//******************************************
#define UDMA_O_SRCENDP 0x00000000 // DMA Channel Source Address End Pointer
#define UDMA_O_DSTENDP 0x00000004 // DMA Channel Destination Address End Pointer
#define UDMA_O_CHCTL 0x00000008 // DMA Channel Control Word

//******************************************
// The following are defines for the bit fields in the UDMA_O_SRCENDP register.
//******************************************
#define UDMA_SRCENDP_ADDR_M 0xFFFFFFFF // Source Address End Pointer
#define UDMA_SRCENDP_ADDR_S 0
The following are defines for the bit fields in the UDMA_O_DSTENDP register.

```
#define UDMA_DSTENDP_ADDR_M 0xFFFFFFFF
// Destination Address End Pointer
#define UDMA_DSTENDP_ADDR_S 0
```

The following are defines for the bit fields in the UDMA_O_CHCTL register.

```
#define UDMA_CHCTL_DSTINC_M 0xC0000000
// Destination Address Increment
#define UDMA_CHCTL_DSTINC_8 0x00000000
// Byte
#define UDMA_CHCTL_DSTINC_16 0x40000000
// Half-word
#define UDMA_CHCTL_DSTINC_32 0x80000000
// Word
#define UDMA_CHCTL_DSTINC_NONE 0xC0000000
// No increment
#define UDMA_CHCTL_DSTSIZE_M 0x30000000
// Destination Data Size
#define UDMA_CHCTL_DSTSIZE_8 0x00000000
// Byte
#define UDMA_CHCTL_DSTSIZE_16 0x10000000
// Half-word
// Word
#define UDMA_CHCTL_DSTSIZE_32 0x20000000
// Source Address Increment
#define UDMA_CHCTL_SRCINC_M 0x0C000000
#define UDMA_CHCTL_SRCINC_8 0x00000000
#define UDMA_CHCTL_SRCINC_16 0x04000000
#define UDMA_CHCTL_SRCINC_32 0x08000000
#define UDMA_CHCTL_SRCINC_NONE 0x0C000000
// Source Data Size
#define UDMA_CHCTL_SRCSIZE_M 0x03000000
#define UDMA_CHCTL_SRCSIZE_8 0x00000000
#define UDMA_CHCTL_SRCSIZE_16 0x01000000
#define UDMA_CHCTL_SRCSIZE_32 0x02000000
#define UDMA_CHCTL_ARBSIZE_M 0x0003C000
#define UDMA_CHCTL_ARBSIZE_1 0x00000000
#define UDMA_CHCTL_ARBSIZE_2 0x00004000
#define UDMA_CHCTL_ARBSIZE_4 0x00008000
#define UDMA_CHCTL_ARBSIZE_8 0x0000C000
#define UDMA_CHCTL_ARBSIZE_16 0x00010000
#define UDMA_CHCTL_ARBSIZE_32 0x00014000
#define UDMA_CHCTL_ARBSIZE_64 0x00018000
#define UDMA_CHCTL_ARBSIZE_128 0x0001C000
// 128 Transfers
#define UDMA_CHCTL_ARBSIZE_256 0x00020000
// 256 Transfers
#define UDMA_CHCTL_ARBSIZE_512 0x00024000
// 512 Transfers
#define UDMA_CHCTL_ARBSIZE_1024 0x00028000
// 1024 Transfers
#define UDMA_CHCTL_XFERSIZE_M 0x00003FF0
// Transfer Size (minus 1)
#define UDMA_CHCTL_NXTUSEBURST 0x00000008
// Next Useburst
#define UDMA_CHCTL_XFERMODE_M 0x00000007
// uDMA Transfer Mode
#define UDMA_CHCTL_XFERMODE_STOP \ 0x00000000
// Stop
#define UDMA_CHCTL_XFERMODE_BASIC \ 0x00000001
// Basic
#define UDMA_CHCTL_XFERMODE_AUTO \ 0x00000002
// Auto-Request
#define UDMA_CHCTL_XFERMODE_PINGPONG \ 0x00000003
// Ping-Pong
#define UDMA_CHCTL_XFERMODE_MEM_SG \ 0x00000004
// Memory Scatter-Gather
#define UDMA_CHCTL_XFERMODE_MEM_SGA \ 0x00000005
// Alternate Memory Scatter-Gather
#define UDMA_CHCTL_XFERMODE_PER_SG \ 0x00000006
// Peripheral Scatter-Gather
#define UDMA_CHCTL_XFERMODE_PER_SGA \ 0x00000007
// Alternate Peripheral
// Scatter-Gather
#define UDMA_CHCTRL_XFERSIZE_S 4

//******************************************
// API Function prototypes
//******************************************
extern void uDMAEnable(void);
extern void uDMADisable(void);
extern unsigned long uDMAErrorStatusGet(void);
extern void uDMAErrorStatusClear(void);
extern void uDMAChannelEnable(unsigned long ulChannelNum);
extern void uDMAChannelDisable(unsigned long ulChannelNum);
extern tBoolean uDMAChannelIsEnabled(unsigned long ulChannelNum);
extern void uDMAControlBaseSet(void *pControlTable);
extern void *uDMAControlBaseGet(void);
extern void *uDMAControlAlternateBaseGet(void);
extern void uDMAChannelRequest(unsigned long ulChannelNum);
extern void uDMAChannelAttributeEnable(unsigned long ulChannelNum,
                                        unsigned long ulAttr);
extern void uDMAChannelAttributeDisable(unsigned long ulChannelNum,
unsigned long ulAttr);

extern unsigned long uDMAChannelAttributeGet(unsigned long ulChannelNum);

extern void uDMAChannelControlSet(unsigned long ulChannelStructIndex,
unsigned long ulControl);

extern void uDMAChannelTransferSet(unsigned long ulChannelStructIndex,
unsigned long ulMode, void *pvSrcAddr,
void *pvDstAddr,
unsigned long ulTransferSize);

extern void uDMAChannelScatterGatherSet(unsigned long ulChannelNum,
unsigned ulTaskCount, void *pvTaskList,
unsigned long ulIsPeriphSG);

extern unsigned long uDMAChannelSizeGet(unsigned long ulChannelStructIndex);

extern unsigned long uDMAChannelModeGet(unsigned long ulChannelStructIndex);

extern void uDMAIntRegister(unsigned long ulIntChannel,
void (*pfHandler)(void));
extern void uDMAIntUnregister(unsigned long ulIntChannel);

extern unsigned long uDMAIntStatus(void);

extern void uDMAIntClear(unsigned long ulChanMask);

extern void uDMAChannelAssign(unsigned long ulMapping);

/*--------------------------------*/
/*
// Mark the end of the C bindings section
    for C++ compilers.
//*/
/*--------------------------------*/

#ifdef __cplusplus
}
#endif
#endif // __UDMA_H__

```c
/* ******************************************
********* Prototypes and macros for utility APIs
*********

// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/

// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
// Redistributions of source code must
// retain the above copyright notice, this list of conditions and
// the following disclaimer.
// Redistributions in binary form must
```
reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#ifndef __UTILS_H__
#define __UTILS_H__

// If building with a C++ compiler, make all of the definitions in this header have a C binding.

#ifdef __cplusplus
extern "C"
{
#endif

// API Function prototypes

extern void UtilsDelay(unsigned long ulCount);

// Mark the end of the C bindings section for C++ compilers.
#ifdef __cplusplus
}
#endif
#endif //__UTILS_H__

//******************************************
***********************************
//
// version.h
//
// Contains Driverlib version details
//
// Copyright (C) 2014 Texas Instruments Incorporated - http://www.ti.com/
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions
// are met:
//
// Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//
// Redistributions in binary form must
reproduce the above copyright

notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the
distribution.

Neither the name of Texas Instruments Incorporated nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//******************************************
//******************************************
ifndef __DRIVERLIB_VERSION_H__
define __DRIVERLIB_VERSION_H__
//******************************************
//
// If building with a C++ compiler, make all of the definitions in this header have a C binding.
//******************************************
//
//ifndef __cplusplus
extern "C"
{
define DRIVERLIB_MAJOR_VERSION_NUM 1
define DRIVERLIB_MINOR_VERSION_NUM 2
define DRIVERLIB_SUBMINOR_VERSION_NUM 0
define DRIVERLIB_RELEASE_DAY 17
define DRIVERLIB_RELEASE_MONTH 2
define DRIVERLIB_RELEASE_YEAR 2016

//******************************************
// Mark the end of the C bindings section for C++ compilers.
//******************************************
#ifdef __cplusplus
}
#endif

#ifndef __DRIVERLIB_VERSION_H__
#endif // __DRIVERLIB_VERSION_H__

// ******************************************
// ***********************************
//
// wd...
notice, this list of conditions and
the following disclaimer in the
documentation and/or other materials
provided with the
distribution.

Neither the name of Texas Instruments
Incorporated nor the names of
its contributors may be used to
derose or promote products derived
from this software without specific
prior written permission.

THIS SOFTWARE IS PROVIDED BY THE
COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED
WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN
NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUE NTIAL
DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
#ifndef __WATCHDOG_H__
#define __WATCHDOG_H__

// If building with a C++ compiler, make all of the definitions in this header have a C binding.

#ifdef __cplusplus
extern "C"
{
#endif

// Prototypes for the APIs.

extern tBoolean WatchdogRunning(unsigned long ulBase);
extern void WatchdogEnable(unsigned long ulBase);
extern void WatchdogLock(unsigned long ulBase);
extern void WatchdogUnlock(unsigned long ulBase);
extern tBoolean WatchdogLockState(unsigned long ulBase);

#endif
extern void WatchdogReloadSet(unsigned long ulBase, unsigned long ulLoadVal);

extern unsigned long WatchdogReloadGet(unsigned long ulBase);

extern unsigned long WatchdogValueGet(unsigned long ulBase);

extern void WatchdogIntRegister(unsigned long ulBase, void(*pfnHandler)(void));

extern void WatchdogIntUnregister(unsigned long ulBase);

extern unsigned long WatchdogIntStatus(unsigned long ulBase, tBoolean bMasked);

extern void WatchdogIntClear(unsigned long ulBase);

extern void WatchdogStallEnable(unsigned long ulBase);

extern void WatchdogStallDisable(unsigned long ulBase);

//*******************************************************************************
//
// Mark the end of the C bindings section for C++ compilers.
//
//*******************************************************************************

#ifdef __cplusplus
}
#endif

#endif // __WATCHDOG_H__
<table>
<thead>
<tr>
<th>MainPage</th>
<th>Modules</th>
<th>Data Structures</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>File List</td>
<td>MainPage</td>
<td>Directory Reference</td>
<td></td>
</tr>
</tbody>
</table>

CC3200 Peripheral Driver Library User's Guide 1.2.0
mainpage.h