OSK5912 BSL

The Board Support Library (BSL) provides a C-language interface for configuring and controlling all on-board devices. The library consists of discrete modules that are built and archived into a library file. Each module represents an individual API and is referred to simply as an API module. The module granularity is structured so each device is implemented in an individual API module. The goal of the BSL is to provide a level of hardware abstraction and software standardization that improves development time and portability.

The BSL contains the following modules:

Module	**Function**
Board Setup	General board initialization
AIC23	Access the AIC23 codec
ENET	Ethernet config and I/O
LED	Manipulate LEDs
FLASH	Program and erase Flash
I2C	I2C config and I/O
POWER	Power management and config
UART	UART config and I/O

Libraries and Include Files

In its most basic form, using the BSL is just a matter of:

1) Linking the BSL in with your code.

2) Referencing the appropriate include (osk5912_###.h) files for each module (see the API descriptions for each function's requirements).

3) Calling the functions themselves, starting with OSK5912_init().

The libraries and include files reside in the "/lib" directory and "/include" directories of the OSK5912 directory. The full paths to these directories are:

c:\ti\boards\osk5912\lib\

c:\ti\boards\osk5912\include

Using the BSL

The Code Composer Projects that come with the OSK automatically includes the lib and include paths above so you don't need to manually include them in the search paths. To include the osk5912bsl.lib file for example, select **Project → Add Files to Project** for the file path you can type or search for c:\ti\boards\osk5912\lib\osk5912bsl.lib.

If you choose to specify your search paths, the library search path is specified on the same tab as the library file and the include search path is specified by selecting **Project → Build Options**, and setting the "Include Search Path" field in the preprocessor section of the compiler tab.

Customizing the BSL

The BSL source code is available in the lib directory along with the library itself. You may extend or modify the source code and build your own version of the library. You can also include the BSL source files directly into your application for greater control. One useful technique while debugging is to use Code Composer's ability to open multiple projects simultaneously to open both your application and the BSL at the same time. The debugger will treat the BSL almost as if the source code for both projects were actually part of the same project.

Project Checklist

The following are things to check when creating your own project:

GEL File	Make sure the gel (osk5912.gel) file is included in your project.
Linker Command File Included	Make sure the linker command (*.cmd) file is included in your project.
Endian Mode is Consistent	The endian mode settings must be consistent in each place it is set. It is set in the project options by selecting **Project → Build Options** and viewing the **Compiler** tab under the **Advanced** Section. Typically, little endian mode should be used on the OSK.
BSL Library Included	Make sure the correct BSL library is included. The BSL library should be specified by selecting **Project → Add Files to Project** for the file path you can type or search for c:\ti\boards\osk5912\lib\osk5912bsl.lib.
Include Paths are Set	If you can't see header (.h) files that your program uses, make sure you have the paths set correctly in **Project → Build Options** on the **Compiler** tab under the **Preprocessor** Section "**Include Search Path**" field. If you can't see header files in the same directory as your project, add a . to your search path. To include multiple paths, place a semicolon between paths, for example .;c:\ti\boards\osk5912\include includes both the project directory and the BSL include directory.
C Runtime Library Included	The TI C Runtime Library allows the use of printf( ) statements and other common C functions. The C Runtime library should be specified by selecting **Project → Add Files to Project** for the file path you can type or search for c:\ti\TMS470\cgtools\lib\rts32e.lib. Note: This is optional if you require the uses of printf( ) statements, or other common C functions.

OSK5912 Board Setup

The Board Setup API provides general functions used for board initialization. The OSK5912_init( ) call must be made before any other BSL functions as it is responsible for all BSL initialization.

The following list summarizes the Board Setup API in terms of its function calls. All programs that use the BSL should include the “osk5912.h” header file.

`OSK5912_init( )`	Initialize the BSL.
`OSK5912_wait( )`	Wait for a specified number of cycles.
`OSK5912_waitusec( )`	Wait for a specified number of microseconds.
`OSK5912_waitmsec( )`	Wait for a specified number of milliseconds.

The following list describes the variable types used throughout the BSL:

`Uint8`	Unsigned 8-bit value.
`Uint16`	Unsigned 16-bit value.
`Uint32`	Unsigned 32-bit value.
`Int8`	Signed 8-bit value.
`Int16`	Signed 16-bit value.
`Int32`	Signed 32-bit value.

OSK5912_init( )

Description	Initializes the BSL (Board Support Library). In addition, OSK5912_init( ) will initialize the EMIF(external memory interface), DSP, and peripherals. Must be called before using any other BSL functions.
Required Headers	osk5912.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_init( Uint16 freq )`
Parameters	freq - Desired frequency to run OSK5912. Only 96MHz &192MHz supported with the BSL.
Return Value	None

Example	`// Initialize the BSL( board support library ) with the 96MHz frequency. This will initialize the EMIF( external memory interface: SDRAM, FLASH ), DSP, and peripherals.`
	`OSK5912_init( 96 );`
	
	`// Initialize the BSL( board support library ) with the 192MHz frequency. This will initialize the EMIF( external memory interface: SDRAM, FLASH ), DSP, and peripherals.`
	`OSK5912_init( 192 );`

OSK5912_wait( )

Description	Wait for a specified number of cycles.
Required Headers	osk5912.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_wait( Uint32 cycles )`
Parameters	cycles - Number of cycles to wait.
Return Value	None

Example	`// Wait for 50 cycles. Once completed, this function will end.`
	`OSK5912_wait( 50 );`
	
	`// Wait for 1000 cycles. Once completed, this function will end.`
	`OSK5912_wait( 1000 );`

OSK5912_waitusec( )

Description	Wait for a specified number of microseconds. The actual wait time will be dependent on the current processor configuration. This includes Instruction Cache, Data Cache, Processor Frequency, and Memory Settings.
Required Headers	osk5912.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_waitusec( Uint32 usec )`
Parameters	usec - Number of microseconds to wait.
Return Value	None
Example	`// Wait for at about 50 microseconds.` `OSK5912_waitusec( 50 );`

OSK5912_waitmsec( )

Description	Wait for a specified number of milliseconds. The actual wait time will be dependent on the current processor configuration. This includes Instruction Cache, Data Cache, Processor Frequency, and Memory Settings.
Required Headers	osk5912.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_waitmsec( Uint32 msec )`
Parameters	msec - Number of milliseconds to wait.
Return Value	None
Example	`// Wait for at about 50 milliseconds.` `OSK5912_waitmsec( 50 );`

OSK5912_AIC23

The AIC23 module controls the audio input and output on the OSK5912.

The codec module include file osk5912_aic23.h contains constant definitions for the handles and AIC23 register offsets. Please view it to see what definitions are available.

The following list summarizes the AIC23 Codec API in terms of its function calls:

`OSK5912_AIC23_opecCodec( )`	Allocate an identifying handle for an instance of a codec.
`OSK5912_AIC23_closeCodec( )`	Release a codec handle.
`OSK5912_AIC23_config( )`	Set parameters on codec registers.
`OSK5912_AIC23_read( )`	Read 32 bits from the codec data stream.
`OSK5912_AIC23_write( )`	Write 32 bit value to the codec data stream.
`OSK5912_AIC23_rset( )`	Set the value of a codec control register.
`OSK5912_AIC23_rget( )`	Return the value of a codec register.
`OSK5912_AIC23_outGain( )`	Set the codec output gain.
`OSK5912_AIC23_loopback( )`	Enable/disable the codec loop-back mode.
`OSK5912_AIC23_mute( )`	Enable/disable the codec mute mode.
`OSK5912_AIC23_powerDown( )`	Enable/disable the codec power down modes.
`OSK5912_AIC23_setFreq( )`	Set the codec sample rate.

OSK5912_AIC23_openCodec( )

Description	Initialize the AIC23 codec module and perform codec setup. Must be called before any other AIC23 functions.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`OSK5912_AIC23_CodecHandle OSK5912_AIC23_openCodec( Uint32 id, OSK5912_AIC23_Config *config )`
Parameters	id - Specifies which codec to use, on the OSK5912, id = 0. config - Pointer to structure containing codec register values. Declared in osk5912_aic23.h. The function will initialize the codec registers with the values in the structure.
Return Value	OSK5912_AIC23_CodecHandle - is an active AIC23 Codec handle or, -1 for failure. The only failure condition is if either of the codec McBSPs are already in use.

Example	`/* Codec configuration settings */`
	`OSK5912_AIC23_Config config = { \`
	`  0x0017,  /* 0 OSK5912_AIC23_LEFTINVOL  Left line input channel volume */ \`
	`  0x0017,  /* 1 OSK5912_AIC23_RIGHTINVOL Right line input channel volume */\`
	`  0x01f9,  /* 2 OSK5912_AIC23_LEFTHPVOL  Left channel headphone volume */  \`
	`  0x01f9,  /* 3 OSK5912_AIC23_RIGHTHPVOL Right channel headphone volume */ \`
	`  0x0011,  /* 4 OSK5912_AIC23_ANAPATH  Analog audio path control */  \`
	`  0x0000,  /* 5 OSK5912_AIC23_DIGPATH  Digital audio path control */  \`
	`  0x0000,  /* 6 OSK5912_AIC23_POWERDOWN  Power down control */  \`
	`  0x0043,  /* 7 OSK5912_AIC23_DIGIF  Digital audio interface format */ \`
	`  0x0082,  /* 8 OSK5912_AIC23_SAMPLERATE Sample rate control */  \`
	`  0x0001  /* 9 OSK5912_AIC23_DIGACT  Digital interface activation */  \`
	`};`
	`// Open codec with these settings`
	`OSK5912_AIC23_CodecHandle hCodec;`
	`hCodec = OSK5912_AIC23_openCodec( 0, &config );`
	
	`// It is now possible to close the active codec handle`
	`OSK5912_ACI23_closeCodec( hCodec );`

OSK5912_AIC23_closeCodec( )

Description	Close the AIC23 codec.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_AIC23_closeCodec( OSK5912_AIC23_CodecHandle hCodec )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( ))
Return Value	None
Example	`// Close the AIC23 Handle.` `OSK5912_AIC23_closeCodec( hCodec )`

OSK5912_AIC23_config( )

Description	Change the codec configuration. Use to set volume, enable mute, etc.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_AIC23_config( OSK5912_AIC23_CodecHandle hCodec, OSK5912_AIC23_Config *Config )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) Config - Pointer to structure containing codec register values. The function will initialize the codec registers with the values in the structure.
Return Value	None

Example	`/* Codec configuration settings */`
	`OSK5912_AIC23_Config config = { \`
	`  0x0017,  /* 0 OSK5912_AIC23_LEFTINVOL  Left line input channel volume */ \`
	`  0x0017,  /* 1 OSK5912_AIC23_RIGHTINVOL Right line input channel volume */\`
	`  0x01f9,  /* 2 OSK5912_AIC23_LEFTHPVOL  Left channel headphone volume */  \`
	`  0x01f9,  /* 3 OSK5912_AIC23_RIGHTHPVOL Right channel headphone volume */ \`
	`  0x0011,  /* 4 OSK5912_AIC23_ANAPATH  Analog audio path control */  \`
	`  0x0000,  /* 5 OSK5912_AIC23_DIGPATH  Digital audio path control */  \`
	`  0x0000,  /* 6 OSK5912_AIC23_POWERDOWN  Power down control */  \`
	`  0x0043,  /* 7 OSK5912_AIC23_DIGIF  Digital audio interface format */ \`
	`  0x0082,  /* 8 OSK5912_AIC23_SAMPLERATE Sample rate control */  \`
	`  0x0001  /* 9 OSK5912_AIC23_DIGACT  Digital interface activation */  \`
	`};`
	`// Configure codec with defaults except mute bit set( bit 3 of reg 5 )`
	`OSK5912_AIC23_config( hCodec, &config );`

OSK5912_AIC23_read16( )

Description	Read a 16-bit signed value from the codec data port.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint16 OSK5912_AIC23_read16( OSK5912_AIC23_CodecHandle hCodec, Int16 *val )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) val – Address of 16-bit variable to receive codec data.
Return Value	0 - No new data read 1 - New data available. Stored as a 16-bit in 'val'.

Example	`Int16 data;`
	`// Read 16-bit codec data, loop to retry if data port is busy.`
	`while( ! OSK5912_AIC23_read16( hCodec, &data ) );`

OSK5912_AIC23_read32( )

Description	Read a 32-bit signed value from the codec data port.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint16 OSK5912_AIC23_read32( OSK5912_AIC23_CodecHandle hCodec, Int32 *val )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) val – Address of 32-bit variable to receive codec data. Left channel in lower 16-bits, right channel in upper 16-bits.
Return Value	0 - No new data read. 1 - New data available. Stored as a 32-bit in 'val'.

Example	`Int32 data;`
	`// Read 32-bit codec data, loop to retry if data port is busy`
	`while( ! OSK5912_AIC23_read32( hCodec, &data ) );`

OSK5912_AIC23_write16( )

Description	Write a 16-bit signed value to the codec data port.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`OSK5912_AIC23_write16( OSK5912_AIC23_CodecHandle hCodec, Int16 val )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) val - 16-bit value signed to write to codec
Return Value	0 - Data has not been written. Port busy. 1 - Data in 'val' has been written

Example	`Int16 data;`
	`data = 0x1234;`
	`// Write 0x1234 to the codec, loop to retry if data port is busy`
	`while( ! OSK5912_AIC23_write16( hCodec, data ) );`

OSK5912_AIC23_write32( )

Description	Write a 32-bit signed value to the codec data port.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint16 OSK5912_AIC23_write32( OSK5912_AIC23_CodecHandle hCodec, Int32 val )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) val - 32-bit value signed to write to codec
Return Value	0 - Data has not been written. Port busy. 1 - Data in 'val' has been written

Example	`Int32 data;`
	`data = 0x12345678;`
	`// Write 0x12345678 to the codec, loop to retry if data port is busy`
	`while( ! OSK5912_AIC23_write32( hCodec, data ) );`

OSK5912_AIC23_rset( )

Description	Set the value of a codec control register.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_AIC23_rset( OSK5912_AIC23_CodecHandle hCodec, Uint16 regnum, Uint16 regval )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) regnum - Index of register to get. Defined in osk5912_aic23.h as OSK5912_AIC23_XXX where XXX is the name of the register: OSK5912_AIC23_LEFTINVOL, OSK5912_AIC23_RIGHTINVOL OSK5912_AIC23_LEFTHPVOL, OSK5912_AIC23_RIGHTHPVOL OSK5912_AIC23_ANAPATH, OSK5912_AIC23_DIGPATH OSK5912_AIC23_POWERDOWN, OSK5912_AIC23_DIGIF OSK5912_AIC23_SAMPLERATE, OSK5912_AIC23_DIGACT OSK5912_AIC23_RESET regval - Value to write to the register
Return Value	None

Example	`// Set codec left and right gain to 0x79`
	`OSK5912_AIC23_rset( hCodec, OSK5912_AIC23_LEFTHPVOL, 0x79 );`
	`OSK5912_AIC23_rset( hCodec, OSK5912_AIC23_RIGHTHPVOL, 0x79 );`

OSK5912_AIC23_rget( )

Description	Return the value of a codec register.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint16 OSK5912_AIC23_rget( OSK5912_AIC23_CodecHandle hCodec, Uint16 regnum )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) regnum - Index of register to get. Defined in osk5912_aic23.h as OSK5912_AIC23_XXX where XXX is the name of the register. Register names are provide here: OSK5912_AIC23_LEFTINVOL, OSK5912_AIC23_RIGHTINVOL OSK5912_AIC23_LEFTHPVOL, OSK5912_AIC23_RIGHTHPVOL OSK5912_AIC23_ANAPATH, OSK5912_AIC23_DIGPATH OSK5912_AIC23_POWERDOWN, OSK5912_AIC23_DIGIF OSK5912_AIC23_SAMPLERATE, OSK5912_AIC23_DIGACT OSK5912_AIC23_RESET
Return Value	16-bit register value

Example	`Uint16 value;`
	`// Read codec register 1( OSK5912_AIC23_RIGHTINVOL )`
	`value = OSK5912_AIC23_rget( hCodec, OSK5912_AIC23_RIGHTINVOL );`

OSK5912_AIC23_outGain( )

Description	Set the codec output gain. The gain is a 7 bit value that corresponds directly to the 7-bit gain field in the codec registers 2 and 3.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_AIC23_outGain( OSK5912_AIC23_CodecHandle hCodec, Uint16 outGain )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) outGain - output gain, range[0-0x7F](0 = lowest, 0x7f = highest)
Return Value	None

Example	// Set output gain to 0x79
	OSK5912_AIC23_outGain(hCodec, 0x79);

OSK5912_AIC23_loopback( )

Description	Enable/disable the codec loop back mode. Loop back mode forces the codec to copy its input directly to its output. It is the opposite of a loop back cable that would copy the output back the input.
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_AIC23_loopback( OSK5912_AIC23_CodecHandle hCodec, Uint16 mode )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) mode - Loop back mode enable(1) or disable(0)
Return Value	None
Example	`// Enable codec loop back mode` `OSK5912_AIC23_loopback( hCodec, 1 );`

OSK5912_AIC23_mute( )

Description	Enable/disable codec mute mode
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_AIC23_mute( OSK5912_AIC23_CodecHandle hCodec, Uint16 mode )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) mode - Codec mute enable(1) or disable(0)
Return Value	None
Example	`// Disable codec mute mode.` `OSK5912_AIC23_mute( hCodec, FALSE );`

OSK5912_AIC23_powerDown( )

Description	Enable/disable codec power down modes for DAC, ADC
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_AIC23_powerDown( OSK5912_AIC23_CodecHandle hCodec, Uint16 sect )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) sect - Bitmap of which sections to enable power down mode. Matches bits in the AIC23 Power Down Control register. Power down modes for a particular codec section is enabled if its corresponding bit is 1. Writing 0 as sect turns disables all power down modes(all sections are on).
Return Value	None
Example	`// Enable the ADC power down mode.` `OSK5912_AIC23_powerDown( hCodec, 0x04 );`

OSK5912_AIC23_setFreq( )

Description	Set the codec sample rate frequency
Required Headers	osk5912.h osk5912_aic23.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_AIC23_setFreq( OSK5912_AIC23_CodecHandle hCodec, Uint32 freq )`
Parameters	hCodec - AIC23 Codec handle(see OSK5912_AIC23_openCodec( )) freq - Sample rate of the codec clock. Can be one of the following, default is 48KHz: OSK5912_AIC23_FREQ_8KHZ, OSK5912_AIC23_FREQ_16KHZ, OSK5912_AIC23_FREQ_24KHZ, OSK5912_AIC23_FREQ_32KHZ, OSK5912_AIC23_FREQ_44KHZ, OSK5912_AIC23_FREQ_48KHZ, OSK5912_AIC23_FREQ_96KHZ. If an invalid frequency is entered no change will be made
Return Value	None
Example	`// Set codec frequency to 48KHz` `OSK5912_AIC23_setFreq( hCodec, OSK5912_AIC23_FREQ_48KHZ );`

OSK5912_ENET

The ENET module controls the on-board Ethernet controller. The ENET module can get and set the registers on the external SMC91C96.

The following list summarizes the ENET API in terms of its function calls:

`OSK5912_ENET_rget( )`	Get the value of an ENET register.
`OSK5912_ENET_rset( )`	Set the value of an ENET register.
`OSK5912_ENET_readEEPROM( )`	Read from the ENET EEPROM.
`OSK5912_ENET_writeEEPROM( )`	Write to the ENET EEPROM.

OSK5912_ENET_rget( )

Description	Get the value of an ENET register.
Required Headers	osk5912.h osk5912_enet.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint16 OSK5912_ENET_rget( Uint16 regnum )`
Parameters	regnum - Index of register to get. Defined in osk5912_enet.h as SMC91C96_XXX where XXX is the name of the register.
Return Value	16-bit register value

Example	`Uint16 regval;`
	`// Get the value of register SMC91C96 MMU_COMMAND`
	`regval = OSK5912_ENET_rget( SMC91C96_MMU_COMMAND );`

OSK5912_ENET_rset( )

Description	Set the value of an ENET register.
Required Headers	osk5912.h osk5912_enet.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_ENET_rset( Uint16 regnum, Uint16 value )`
Parameters	regnum - Index of register to get. Defined in osk5912_enet.h as SMC91C96_XXX where XXX is the name of the register. value - 16-bit register value.
Return Value	None

Example	`Uint16 regval = 0x40;`
	`// Set the value of register SMC91C96 MMU_COMMAND to 0x40`
	`OSK5912_ENET_rget( SMC91C96_MMU_COMMAND, regval );`

OSK5912_ENET_readEEPROM( )

Description	Read from the on-board ENET EEPROM.
Required Headers	osk5912.h osk5912_enet.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint16 OSK5912_ENET_readEEPROM( Uint16 addr )`
Parameters	addr - Address to read from the Ethernet EEPROM.
Return Value	16-bit value

Example	`Uint16 data;`
	`// Read the ENET EEPROM contents at address 0`
	`data = OSK5912_ENET_readEEPROM( 0 );`

OSK5912_ENET_writeEEPROM( )

Description	Write to the on-board ENET EEPROM
Required Headers	osk5912.h osk5912_enet.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_ENET_writeEEPROM( Uint16 addr, Uint16 value )`
Parameters	addr - Address to read from the Ethernet EEPROM value - 16-bit to write to at address addr.
Return Value	None

Example	`Uint16 data;`
	`// Read the ENET EEPROM contents at address 0`
	`data = OSK5912_ENET_readEEPROM( 0 );`
	`// Write to the ENET EEPROM address 0, with the original contents`
	`OSK5912_ENET_writeEEPROM( 0, data );`

OSK5912_FLASH

The FLASH module API provides functions to erase as well as read and write from the on-board Flash memory. Programming errors can be detected through use of a checksum function. The total size of Flash memory can be determined by the number of pages on each ob-board Flash memory module through use of the getTotalPages function.

The Flash memory is divided into logical pages which must be erased before being re-programmed. After being erased, locations within the page can be programmed in any order. However, each location can only be programmed once per erasure.

To optimize very large reads and writes to Flash memory, the readBurst() and writeBurst() functions can be used instead of read() & write()..

The following list summarizes the FLASH API in terms of its function calls:

`OSK5912_FLASH_init( )`	Enable Flash memory.
`OSK5912_FLASH_checksum( )`	Calculate checksum for a memory range.
`OSK5912_FLASH_getTotalPages( )`	Calculate the total number of pages in Flash.
`OSK5912_FLASH_erase( )`	Erase a range of Flash memory.
`OSK5912_FLASH_read( )`	Read data from a range in Flash.
`OSK5912_FLASH_write( )`	Write data to a range in Flash.
`OSK5912_FLASH_readBurst( )`	Read data from a range in Flash using burst reads.
`OSK5912_FLASH_writeBurst( )`	Write data to a range in Flash using burst writes.

Programs that use the FLASH API should include both the osk5912.h and the osk5912_flash.h BSL header files. The header file osk5912_flash.h contains several constants that may be useful while programming:

Name	**Value**	**Description**
OSK5912_FLASH_BASE	0x00000000	Start address of Flash memory.
OSK5912_FLASH_PAGESIZE	0x20000	Size of a single page of Flash.

OSK5912_FLASH_init( )

Description	Initialize the Flash memory, set the mode to read, and disable the Flash write protect. Must be called before using any other Flash function.
Required Headers	osk5912.h osk5912_flash.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_FLASH_init( )`
Parameters	None
Return Value	None

Example	`// Initialize Flash memory. The current Flash state is in read mode with the Flash write protection disabled. With the write protection disabled, the Flash can now be erased and programmed.`
	
	`OSK5912_FLASH_init( );`

OSK5912_FLASH_checksum( )

Description	A checksum is most commonly used check for programming errors in Flash. The checksum is the unsigned sum of all 8-bit bytes in a given memory range. If the sum is exceeds 0xFFFFFFFF it wraps around at 32-bits. The sum is returned as a single 32-bit value
Required Headers	osk5912.h osk5912_flash.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint32 OSK5912_FLASH_checksum( Uint32 start, Uint32 length )`
Parameters	start - Starting address of Flash memory. length - Length in bytes.
Return Value	32-bit checksum that is generated by adding all the bytes in the Flash range.

Example	`Uint32 checksum;`
	`// Calculate the checksum for the first sector in Flash`
	`checksum = OSK5912_FLASH_checksum( OSK5912_FLASH_BASE, OSK5912_FLASH_PAGESIZE );`

OSK5912_FLASH_erase( )

Description	Erase a given range of memory in Flash. The erase function can only erase full pages of Flash memory, not an arbitrary length in Flash. Therefore the pages that encapsulate the given range are erased. Once the erase is complete, the values at Flash memory are all 'ones'.
Required Headers	osk5912.h osk5912_flash.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_FLASH_erase( Uint32 start, Uint32 length )`
Parameters	start - Starting address of Flash to beginning erase. length - Minimum length of erase in bytes. Flash memory can only erase full pages.
Return Value	None

Example	`// Erase the first page of Flash.`
	`OSK5912_FLASH_erase( OSK5912_FLASH_BASE, OSK5912_FLASH_PAGESIZE );`
	
	`// This function has a similar effect as the function call above. Only full pages can be erased, not single entries.`
	`OSK5912_FLASH_erase( OSK5912_FLASH_BASE, 1 );`
	
	`// Erase the entire contents of Flash in the first device.`
	`Uint32 pages = OSK5912_FLASH_getTotalPages( 1 );`
	`OSK5912_FLASH_erase( OSK5912_FLASH_BASE, pages * OSK5912_FLASH_PAGESIZE );`

OSK5912_FLASH_getTotalPages( )

Description	Each Flash device has a Device ID & Manufacture ID. The total number of pages in a Flash device can be determined by reading these 2 IDs from each Flash device.
Required Headers	osk5912.h osk5912_flash.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint32 OSK5912_FLASH_getTotalPages( Uint16 num_flash_device )`
Parameters	num_flash_device - Number of physical Flash devices on the OSK5912(typically 2 devices)
Return Value	Number of total pages in Flash memory.

Example	`Uint32 pages;`
	
	`// Get the total number of pages from the 2 Flash devices on the OSK5912. The supported Flash Devices are Intel & Micron based Flash.`
	
	`pages = OSK5912_FLASH_getTotalPages( 2 );`

OSK5912_FLASH_read( )

Description	Read data from a range in Flash.
Required Headers	osk5912.h osk5912_flash.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_FLASH_read( Uint32 src, Uint32 dst, Uint32 length )`
Parameters	src - Address of Flash memory to read from. dst - Address of non-Flash memory to write to. length - Length in bytes.
Return Value	None

Example	`Uint8 buffer[256];`
	`// Copy the first 256 8-bit bytes from the beginning of Flash Memory to buffer. Buffer must be type-cast to Uint32( unsigned 32-bit integer ).`
	`OSK5912_FLASH_read( OSK5912_FLASH_BASE,( Uint32 )buffer, 256 );`

OSK5912_FLASH_write( )

Description	Write data to a range in Flash. The range in Flash must be erased prior to writing.
Required Headers	osk5912.h osk5912_flash.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_FLASH_write( Uint32 src, Uint32 dst, Uint32 length )`
Parameters	src - Address of non-Flash memory to read from. dst - Address of Flash memory to write to. length - Length in bytes.
Return Value	None

Example	`Uint8 buffer[256];`
	`Int16 i;`
	`for( i = 0 ; i < 256 ; i++ )`
	`buffer[i] = i;`
	
	`// Erase the contents in Flash before writing. This assures that the data will be valid.`
	`OSK5912_FLASH_erase( OSK5912_FLASH_BASE, 256 );`
	
	`// Write the memory pattern in 'buffer' to the first 256 bytes of Flash.`
	`OSK5912_FLASH_write( OSK5912_FLASH_BASE,( Uint32 )buffer, 256 );`

OSK5912_FLASH_readBurst( )

Description	Read data from a range in Flash. This function will use a faster approach to reading Flash memory.
Required Headers	osk5912.h osk5912_flash.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_FLASH_readBurst( Uint32 src, Uint32 dst, Uint32 length )`
Parameters	src - Address of Flash memory to read from. dst - Address of non-Flash memory to write to. length - Length in bytes.
Return Value	None

Example	`Uint8 buffer[256];`
	`// Copy the first 256 8-bit bytes from the beginning of Flash Memory to buffer. Buffer must be type-cast to Uint32( unsigned 32-bit integer ).`
	`// Functionally this is identical to OSK5912_FLASH_read(), however it is more efficient at reading large bursts of data but not single entries.`
	`OSK5912_FLASH_readBurst( OSK5912_FLASH_BASE,( Uint32 )buffer, 256 );`

OSK5912_FLASH_writeBurst( )

Description	Write data to a range in Flash. The range in Flash must be erased prior to writing. This function uses a buffer on the Flash device to speed up writes.
Required Headers	osk5912.h osk5912_flash.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_FLASH_writeBurst( Uint32 src, Uint32 dst, Uint32 length )`
Parameters	src - Address of non-Flash memory to read from. dst - Address of Flash memory to write to. length - Length in bytes.
Return Value	None

Example	`Uint8 buffer[256];`
	`Int16 i;`
	`for( i = 0 ; i < 256 ; i++ )`
	`buffer[i] = i;`
	
	`// Erase the contents in Flash before writing. This assures that the data will be valid.`
	`OSK5912_FLASH_erase( OSK5912_FLASH_BASE, 256 );`
	`// Write the memory pattern in 'buffer' to the first 256 bytes of Flash.`
	`// Functionally this is identical to OSK5912_FLASH_write(), however it is more efficient at writing large bursts of data but not single entries.`
	`OSK5912_FLASH_writeBurst( OSK5912_FLASH_BASE,( Uint32 )buffer, 256 );`

OSK5912_I2C

The I2C module API allows provides functions to read and write to various I2C slave devices. The I2C module will configure the on-chip I2C controller to function at 400 KHz.

The following list summarizes the I2C API in terms of its function calls:

`OSK5912_I2C_init( )`	Initialize the I2C controller.
`OSK5912_I2C_read( )`	Read from an I2C Slave device.
`OSK5912_I2C_write( )`	Write to an I2C Slave device.

OSK5912_I2C_init( )

Description	Initialize the I2C controller. The I2C module only supports the I2C controller in master transmit/receive modes. Must be called before any other I2C function.
Required Headers	osk5912.h osk5912_i2c.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_I2C_init( )`
Parameters	None
Return Value	None
Example	`// Initialize the I2C Controller. The I2C frequency is set to 400KHz.` `OSK5912_I2C_init( );`

OSK5912_I2C_read( )

Description	Read data from an I2C slave device.
Required Headers	osk5912.h osk5912_i2c.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint16 OSK5912_I2C_read( Uint16 slaveaddr, Uint8* data, Uint16 length )`
Parameters	slaveaddr - 7-bit slave address data - Pointer to the buffer to store data. length - Length in bytes
Return Value	0 - Pass: Data read. 1 - Fail: Data was not read.

Example	`Uint8 buffer[2];`
	`Uint8 addr = 0x1b;`
	`// Initialize the I2C Controller before using I2C module.`
	`OSK5912_I2C_init( );`
	
	`// Grab 1 byte of data from I2C Slave device with address = addr, and store data in buffer.`
	`OSK5912_I2C_read( addr, buffer, 1 );`
	
	`// Grab 2 bytes of data from I2C Slave device with address = addr, and store data in buffer.`
	`OSK5912_I2C_read( addr, buffer, 2 );`

OSK5912_I2C_write( )

Description	Write data to an I2C slave device.
Required Headers	osk5912.h osk5912_i2c.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint16 OSK5912_I2C_write( Uint16 slaveaddr, Uint8* data, Uint16 length )`
Parameters	slaveaddr - 7-bit slave address data - Pointer to the data to be written. length - Length in bytes
Return Value	0 - Pass: Data written. 1 - Fail: Data was not written.

Example	`Uint8 buffer[2] = { 1, 2 };`
	`Uint8 addr = 0x1b;`
	`// Initialize the I2C Controller before using I2C module.`
	`OSK5912_I2C_init( );`
	
	`// Send 1 byte of data to I2C Slave device with address = addr.`
	`OSK5912_I2C_write( addr, buffer, 1 );`
	
	`// Send 2 bytes of data to I2C Slave device with address = addr.`
	`OSK5912_I2C_write( addr, buffer, 2 );`

OSK5912_LED

The LED module API provides functions to control the on-board OSK5912 LEDs. The LEDs can be turned on and off, or have the state toggle between on and off.

The following list summarizes the LED API in terms of its function calls:

`OSK5912_LED_init( )`	Initialize the LEDs. Default state is LED D2 & D3 off.
`OSK5912_LED_off( )`	Turn specified LED off
`OSK5912_LED_on( )`	Turn specified LED on
`OSK5912_LED_toggle( )`	Toggle specified LED

OSK5912_LED_init( )

Description	Initialize the LED module. LED state after initialization is D2 & D3 OFF. Must be called before any other LED functions.
Required Headers	osk5912.h osk5912_led.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_LED_init( )`
Parameters	None
Return Value	None
Example	`// Initialize the LED module. LED D2 & D3 will be turned off.` `OSK5912_LED_init( );`

OSK5912_LED_off( )

Description	Turn one of the LEDs off.
Required Headers	osk5912.h osk5912_led.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_LED_off( Uint16 lednum )`
Parameters	lednum - Indicates which led to turn off. Range [0-1].
Return Value	None

Example	`// Turn off LED 0.`
	`OSK5912_LED_off( 0 );`
	
	`// Turn off LED 0. The second call will not affect LED 0, since it is already off.`
	`OSK5912_LED_off( 0 );`

OSK5912_LED_on( )

Description	Turn one of the LEDs on.
Required Headers	osk5912.h osk5912_led.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_LED_on( Uint16 lednum )`
Parameters	lednum - Indicates which led to turn on. Range [0-1].
Return Value	None

Example	`// Turn on LED 0`
	`OSK5912_LED_on( 0 );`
	
	`// Turn on LED 0. The second call will not affect LED 0, since it is already on.`
	`OSK5912_LED_on( 0 );`

OSK5912_LED_toggle( )

Description	Toggle one of the LED between on & off.
Required Headers	osk5912.h osk5912_led.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_LED_toggle( Uint16 lednum )`
Parameters	lednum - Indicates which led to toggle. Ranges [0-1].
Return Value	None

Example	`// Toggle LED 0`
	`OSK5912_LED_toggle( 0 );`
	
	`// Toggle LED 0`
	`OSK5912_LED_toggle( 0 );`
	`// At this point LED 0 will have toggled to its original state.`

OSK5912_POWER

The POWER module API controls the on-board TI Power Controller (TIS65010).

The following list summarizes the POWER API in terms of its function calls:

`OSK5912_POWER_rget( )`	Get the value of a power controller register.
`OSK5912_POWER_rset( )`	Set the value of a power controller register.

OSK5912_POWER_rget( )

Description	Read from a Power Controller register.
Required Headers	osk5912.h osk5912_power.h
Required Libraries	osk5912bsl.lib
Prototype	`Int16 OSK5912_POWER_rget( Uint8 regnum, Uint8 *regval )`
Parameters	regnum - register address on the Power Controller. regnum range [0x00 - 0x10] regval - pointer to store the register value.
Return Value	0 - Registers read passed. 1 - Registers read failed.

Example	`Uint8 regnum = 0x10;`
	`Uint8 regval;`
	`// Read from register regnum and store in regval.`
	`OSK5912_POWER_rget( regnum, &regval );`

OSK5912_POWER_rset( )

Description	Write to a Power Controller register.
Required Headers	osk5912.h osk5912_power.h
Required Libraries	osk5912bsl.lib
Prototype	`Int16 OSK5912_POWER_rset( Uint8 regnum, Uint8 regval )`
Parameters	regnum - register address on the Power Controller. regnum range [0x00 - 0x10] regval - value to write to the register.
Return Value	0 - Registers write passed. 1 - Registers write failed.

Example	`Uint8 regnum = 0x10;`
	`Uint8 regval = 0x55;`
	`// Write to register regnum the value regval.`
	`OSK5912_POWER_rset( regnum, regval );`

OSK5912_UART

The UART module communicates through the on-chip UART peripheral to the RS232 Serial Connector. The UART API provides functions to send and receive 8-bit data serially to an external device. To use the module, you must use the OSK5912_UART_open( ) call to get a handle of type OSK5912_UART_Handle which is then passed as a parameter to the other functions.

The following list summarizes the UART API in terms of its function calls:

`OSK5912_UART_open( )`	Initialize the UART module and return an active UART handle.
`OSK5912_UART_rget( )`	Read from a UART register.
`OSK5912_UART_rset( )`	Write to a UART register.
`OSK5912_UART_rcvReady( )`	Check if UART is ready to receive data.
`OSK5912_UART_getChar( )`	Get an 8-bit byte through UART.
`OSK5912_UART_xmtReady( )`	Check if UART is ready to send data.
`OSK5912_UART_putChar( )`	Send an 8-bit byte through UART.

OSK5912_UART_open( )

Description	Open a UART device at the specific baud rate and return a UART handle.
Required Headers	osk5912.h osk5912_uart.h
Required Libraries	osk5912bsl.lib
Prototype	`OSK5912_UART_Handle OSK5912_UART_open( Int16 devid, Int16 baudrate, OSK5912_UART_Config *Config )`
Parameters	devid - Device id, specifies the UART, Range [0-2]. baudrate - Baud rate, uses constants defined in osk5912_uart.h to set the UART to the matching baud rate. Possible values are: UART_BAUD_RATE_300, UART_BAUD_RATE_600, UART_BAUD_RATE_1200, UART_BAUD_RATE_2400, UART_BAUD_RATE_4800, UART_BAUD_RATE_9600, UART_BAUD_RATE_14400, UART_BAUD_RATE_19200, UART_BAUD_RATE_28800, UART_BAUD_RATE_38400, UART_BAUD_RATE_57600, UART_BAUD_RATE_115200, UART_BAUD_RATE_230400, UART_BAUD_RATE_460800, UART_BAUD_RATE_921600, UART_BAUD_RATE_1834200, UART_BAUD_RATE_3686400. Config - Pointer to a UART registers config structure, declared in osk5912.uart.h.
Return Value	Active UART handle.

Example	`// Open UART 0 at 19200 baud in normal( non-FIFO mode )`
	`OSK5912_UART_Handle hUart;`
	`OSK5912_UART_Config uartcfg = {`
	`  0x00, // IER`
	`  0x00, // FCR`
	`  0x03, // LCR`
	`  0x00 // MCR`
	`};`
	`hUart = OSK5912_UART_open( 0, OSK5912_UART_BAUD_19200, &uartcfg );`

OSK5912_UART_rget( )

Description	Get the value of a UART register.
Required Headers	osk5912.h osk5912_uart.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint8 OSK5912_UART_rget( OSK5912_UART_Handle hUart, Int16 regnum )`
Parameters	hUart - UART handle(see OSK5912_UART_open( )) regnum - Index of register to get. Defined in osk5912_uart.h as OSK5912_UART_XXX where XXX is the name of the register
Return Value	An 8-bit register value.

Example	`// Get the value of the LCR register`
	`Uint8 data;`
	`data = OSK5912_UART_rget( hUart, UART_LCR );`

OSK5912_UART_rset( )

Description	Set the value of a UART register.
Required Headers	osk5912.h osk5912_uart.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_UART_rset( OSK5912_UART_Handle hUart, Int16 regnum, Uint8 regval )`
Parameters	hUart - UART handle(see OSK5912_UART_open( )) regnum- Index of register to get. Defined in osk5912_uart.h as OSK5912_UART_XXX where XXX is the name of the register regval - Value to set the register to, only lower 8-bits are assigned
Return Value	None
Example	`// Set the value of the LCR register to 0x03` `OSK5912_UART_rset( hUart, OSK5912_UART_LCR, 0x03 );`

OSK5912_UART_rcvReady( )

Description	Check if the UART is ready to receive data.
Required Headers	osk5912.h osk5912_uart.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint8 OSK5912_UART_rcvReady( OSK5912_UART_Handle hUart )`
Parameters	hUart - UART handle(see OSK5912_UART_open( ))
Return Value	0 - No data waiting to be read. 1 - Data waiting to be read.

Example	`Int16 data;`
	`// Wait for data to be read.`
	`while( ! OSK5912_UART_rcvReady( hUart ) );`
	`// Read a character from the UART`
	`data = OSK5912_UART_getChar( hUart );`

OSK5912_UART_getChar( )

Description	Get a character from the UART.
Required Headers	osk5912.h osk5912_uart.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint8 OSK5912_UART_getChar( OSK5912_UART_Handle hUart )`
Parameters	hUart - UART handle(see OSK5912_UART_open( ))
Return Value	8-bit character value.

Example	`// Read a character from the UART`
	`Int16 data;`
	`data = OSK5912_UART_getChar( hUart );`

OSK5912_UART_xmtReady( )

Description	Check if UART is ready to send data.
Required Headers	osk5912.h osk5912_uart.h
Required Libraries	osk5912bsl.lib
Prototype	`Uint8 OSK5912_UART_xmtReady( OSK5912_UART_Handle hUart )`
Parameters	hUart - UART handle(see OSK5912_UART_open( ))
Return Value	0 - UART no ready to send data. 1 - UART ready to send data.

Example	`// Wait for UART to be ready to send.`
	`while( ! OSK5912_UART_xmtReady( hUart ) );`
	`// Send the character 'A' to the UART`
	`OSK5912_UART_putChar( hUart, 'A' );`

OSK5912_UART_putChar( )

Description	Send a character through the UART.
Required Headers	osk5912.h osk5912_uart.h
Required Libraries	osk5912bsl.lib
Prototype	`void OSK5912_UART_putChar( OSK5912_UART_Handle hUart, Uint8 data )`
Parameters	hUart - UART handle(see OSK5912_UART_open( )) data - character to be sent
Return Value	None
Example	`// Send the character 'A' to the UART` `OSK5912_UART_putChar( hUart, 'A' );`