LED Example

Description:

The LED example is a simple program to demonstrate the most basic usage of the BSL. While running, it will continuously blink LED #0 about 2.5 times per second.

Perform the following steps to run the LED example:

1) Load project [**led.pjt**]

   [To Load Project] Select the tab **Project → Open**, then select **led.pjt** in the directory `c:\ti\boards\osk5912\examples\led`.]

   [NOTE: The default install directory for Code Composer is `c:\ti`. All of the OSK5912 specific documentation refers to file paths as if Code Composer is installed at the default location. If Code Composer is in a different location, please remember to mentally remap the files to the other location while reading this document. For example, if Code Composer is installed at in `c:\tiosk5912` the led example would be located at `c:\tiosk5912\boards\osk5912\examples\led`.]

2) Load program [**led.out**]

   [To Load Program] Select tab **File → Load Program**. It will open a file browser dialog. Select the **led.out** file in the Debug directory in the file browser and hit "**Open**" to load.

   [NOTE: Recompile the program every time changes are made.]

3) Run program [**led.out**]

   [To Run Program] Select tab **Debug → Run**. LED #0 will be blinking slowly.

4) Halt program [**led.out**]

   [To Halt Program] Select tab **Debug → Halt.** Halt the program when you are satisfied with the LED demo.

LED Example Description

LED program starts at the beginning of main( ). The first call is to OSK5912_init( ) which will initialize the Board Support Library (BSL). The BSL is a library designed specifically to make it easier to use the components on the OSK5912 board. OSK5912_init( ) should be called before any other BSL functions. You can recognize BSL calls easily because they all start with the prefix OSK5912 the BSL programming interface is described fully in the BSL section of this help file. The BSL functions are included as a library called osk5912bsl.lib.

The LED example demonstrates use of the LED module. You must call the OSK5912_LED_init( ) prior to calling the other functions in the module. The OSK5912_LED_toggle( ) call toggles the state of LED #0. A software delay loop:

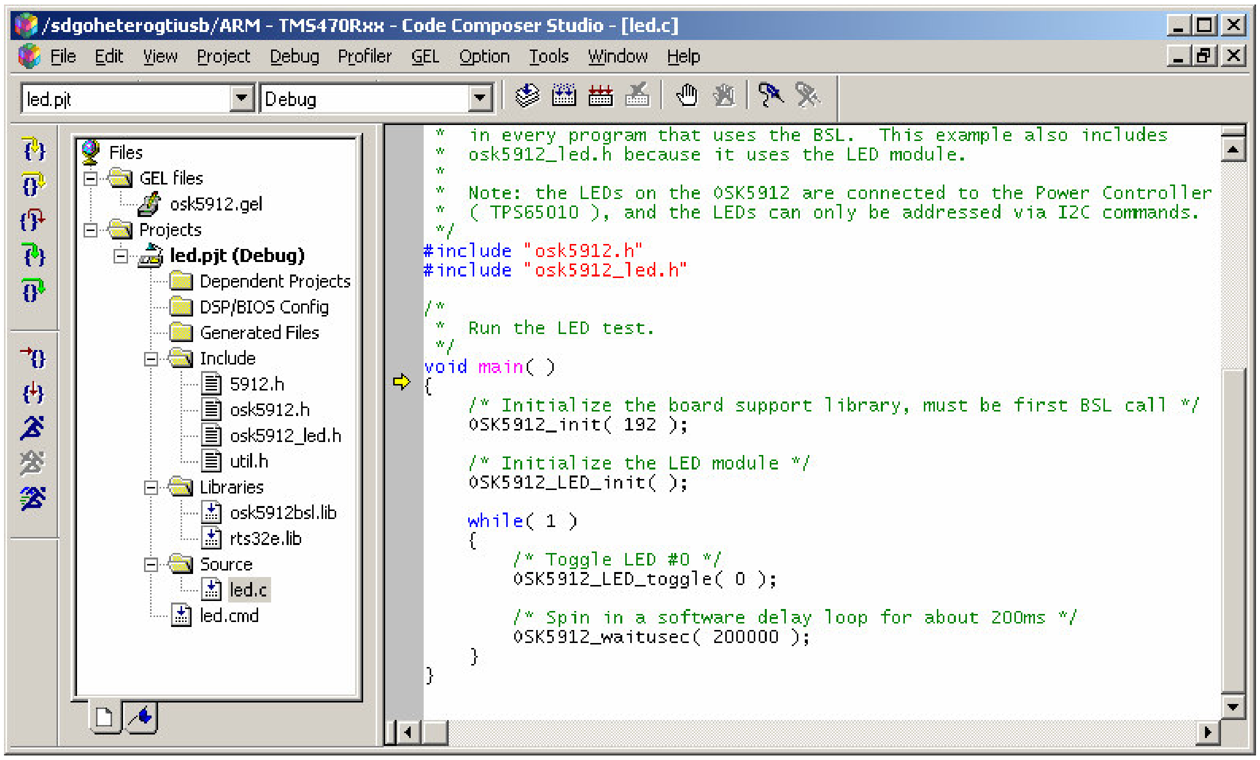
```
/* Spin in a software delay loop for about 200ms */

OSK5912_waitusec( 200000 );
```

introduces roughly 200 milliseconds delay before toggling the LED again. This delay loop is responsible for controlling the speed of the blinking LED.

Since all of this code is in a while loop with no termination condition, the program will run forever until you halt the program.

To examine the program code, expand the **Projects** tree at the left of the workspace, then expand the **led.pjt** and **Source** sub items. Double click on **led.c** to view the code.



```
 *  in every program that uses the BSL.  This example also includes
 *  osk5912_led.h because it uses the LED module.
 *
 *  Note: the LEDs on the OSK5912 are connected to the Power Controller
 *  ( TPS65010 ), and the LEDs can only be addressed via I2C commands.
 */
#include "osk5912.h"
#include "osk5912_led.h"

/*
 *  Run the LED test.
 */
void main( )
{
    /* Initialize the board support library, must be first BSL call */
    OSK5912_init( 192 );

    /* Initialize the LED module */
    OSK5912_LED_init( );

    while( 1 )
    {
        /* Toggle LED #0 */
        OSK5912_LED_toggle( 0 );

        /* Spin in a software delay loop for about 200ms */
        OSK5912_waitusec( 200000 );
    }
}
```

Zoom In if needed

Making Simple Changes

In order to become familiar with Code Composer, this example will take a quick walk through the steps involved in making simple changes to the example. One of the easiest changes to make is to make the LED blink at a slower/faster rate.

View the source for the main( ) function call. The `delay` loop contains a value that represents the number of microseconds to wait inside the while( ) loop between LED transitions. Change the statement:

```
/* Spin in a software delay loop for about 200ms */
OSK5912_waitusec( 200000 );
```

to:

```
/* Spin in a software delay loop for about 100ms */
OSK5912_waitusec( 100000 );
```

You should change the delay loop count from 200000 to 100000 to cut the delay in half, increasing the LED blink rate by a factor of 2.

To try out your new changes you must first:

1) Save [**led.c**]

   [To Save] Select tab **File → Save**

2) Re-compile [**led.pjt**]

   [To Compile] Select **Project → Build**. A status of the build will pop up. When no errors are found the project successfully compiled.

3) Re-load [**led.out**]

   [To Load Program] Select tab **File → Load Program**. Select **led.out** file in the Debug directory.

4) Run program [**led.out**]

   [To Run Program] Select tab **Debug → Run**. LED #0 will start blinking faster than before.

5) Halt program [**led.out**]

   [To Halt Program] Select tab **Debug → Halt.** Halt the program when you are satisfied with the LED demo.

Other changes to be made involved the other LED on the OSK5912. Simple refer to LED 1 by switching to the instance of 0 in the toggle() function with a 1.

The other LED functions in the BSL are available and can be added to the project.

Tone Example

The tone example (found in **c:\ti\boards\osk5912\examples\tone**) is a slightly more complicated program that directs the AIC23 & McBSP to generate a 1 KHz sine wave through the headphone jack.

Perform the following steps to run the Tone example:

1) Close previous projects

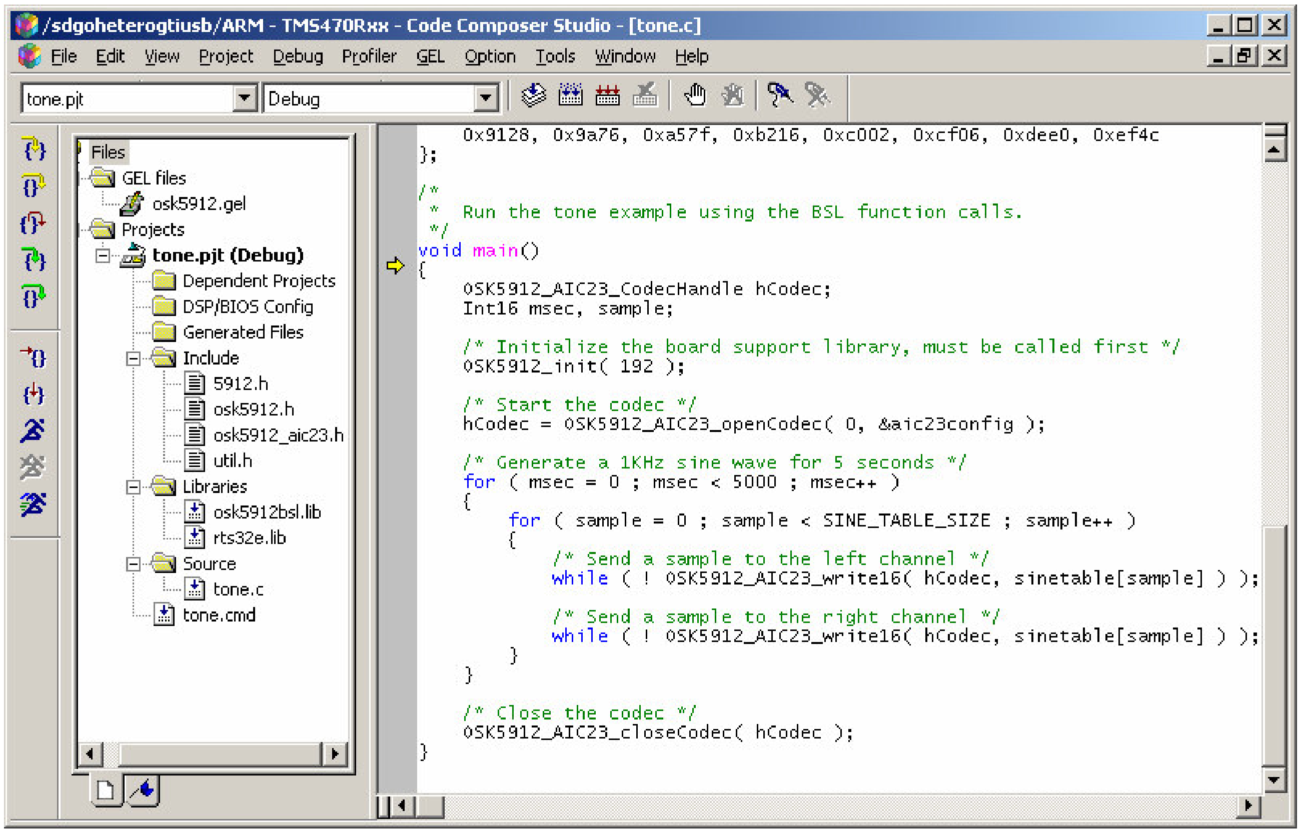
   [To Close Project] Select tab **Project → Close**. The current project, indicated by the **bold** letters under Projects tree, will be closed.

   [NOTE: Close any open files associated with the previous project to avoid confusion.]

2) Load Project [**tone.pjt**]

   [To Load Project] Select the tab **Project → Open**, then select tone.pjt in the directory **c:\ti\boards\osk5912\examples\tone**.]

   [NOTE: Below is a depiction of tone.pjt]



```
    0x9128, 0x9a76, 0xa57f, 0xb216, 0xc002, 0xcf06, 0xdee0, 0xef4c
};

/*
 *  Run the tone example using the BSL function calls.
 */
void main()
{
    OSK5912_AIC23_CodecHandle hCodec;
    Int16 msec, sample;

    /* Initialize the board support library, must be called first */
    OSK5912_init( 192 );

    /* Start the codec */
    hCodec = OSK5912_AIC23_openCodec( 0, &aic23config );

    /* Generate a 1KHz sine wave for 5 seconds */
    for ( msec = 0 ; msec < 5000 ; msec++ )
    {
        for ( sample = 0 ; sample < SINE_TABLE_SIZE ; sample++ )
        {
            /* Send a sample to the left channel */
            while ( ! OSK5912_AIC23_write16( hCodec, sinetable[sample] ) );

            /* Send a sample to the right channel */
            while ( ! OSK5912_AIC23_write16( hCodec, sinetable[sample] ) );
        }
    }

    /* Close the codec */
    OSK5912_AIC23_closeCodec( hCodec );
}
```

Zoom In if needed

3) Plug a headphone or speaker into the headphone audio jack.

4) Load program [**tone.out**]

[To Load Program] Select tab **File → Load Program**. It will open a file browser dialog. Select the **tone.out** file in the Debug directory in the file browser and hit "**Open**" to load.

[NOTE: You must reload the compiled program every time you make changes to the program.]

5) Run program [**tone.out**]

[To Run Program] Select tab **Debug → Run**. Listen for a 1 KHz 5 seconds long tone.

Tone Example Description

The array `sinetable` [ ] contains a pre-generated sine wave using signed 16-bit data that matches the AIC23. The data covers exactly one period and the amplitude matches the full range of the codec.

```
#define SINE_TABLE_SIZE  48

/* Pre-generated sine wave data, 16-bit signed samples */

Unt16 sinetable[SINE_TABLE_SIZE] = {

  0x0000, 0x10b4, 0x2120, 0x30fb, 0x3fff, 0x4dea, 0x5a81, 0x658b,

  0x6ed8, 0x763f, 0x7ba1, 0x7ee5, 0x7ffd, 0x7ee5, 0x7ba1, 0x76ef,

  0x6ed8, 0x658b, 0x5a81, 0x4dea, 0x3fff, 0x30fb, 0x2120, 0x10b4,

  0x0000, 0xef4c, 0xdee0, 0xcf06, 0xc002, 0xb216, 0xa57f, 0x9a75,

  0x9128, 0x89c1, 0x845f, 0x811b, 0x8002, 0x811b, 0x845f, 0x89c1,

  0x9128, 0x9a76, 0xa57f, 0xb216, 0xc002, 0xcf06, 0xdee0, 0xef4c

};
```

The main loop of the code writes each data point in the sine wave table out to the codec using the AIC23 codec package of the BSL. Each write function sends a single 16 bit sample to the codec. In this case the same data is sent out twice, once to the left channel and once to the right channel. The codec is configured to accept data at a rate of 48,000 stereo samples per second. Since the sine table is 48 entries long, the resulting output wave will be a 1 KHz sine wave with the same output on both the left and right channels.

```
// Generate a 1 KHz sine wave for 5 seconds

for ( msec = 0; msec < 5000; msec++ )

{

  for ( sample = 0; sample < SINE_TABLE_SIZE; sample++ )

  {

    /* Send a pair of stereo samples, left high, right low */

    while ( !OSK5912_AIC23_write( hCodec, ( sinetable[sample] << 16 ) |

      sinetable[sample] ) );

  }

}
```

The McBSP is used to transmit data to the codec at a much slower rate than the DSP can process data. It accepts data 32 bits at a time and shifts them out slowly one at a time. The write function returns a 1 if the write is completed successfully or a 0 if the serial channel is busy. The while( ) loop around the writes waits while the serial port is busy so program can be synchronized to the data rate of the codec.

The 32 bit data consists of two 16-bit samples, each corresponding to one of the two audio channels. The left data sits in the top half of the 32-bit word while the right data sits in the bottom half . Each sample is a signed 16-bit value.

The following two commands are used to initialize and shut down the codec and are found at the beginning and end of all programs that use BSL codec module. The OSK5912_openCodec( ) command returns a handle that is passed each of the other codec functions.

```
// Start the codec

hCodec = OSK5912_AIC23_openCodec( 0, &config, OSK5912_AIC23_OUTPUT );

// Close the codec

OSK5912_AIC23_closeCodec( hCodec );
```

One of the most important pieces of the codec code is the setup structure defined at the top of the code:

```
/* AIC23 codec settings - each entry below represents an AIC23 register */

OSK5912_AIC23_Config aic23config = {

    0x0017,  /* 0 OSK5912_AIC23_LEFTINVOL  Left line input channel volume   */ \

    0x0017,  /* 1 OSK5912_AIC23_RIGHTINVOL Right line input channel volume  */ \

    0x00d8,  /* 2 OSK5912_AIC23_LEFTHPVOL  Left channel headphone volume    */ \

    0x00d8,  /* 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   */ \

    0x0081,  /* 8 OSK5912_AIC23_SAMPLERATE Sample rate control              */ \

    0x0001   /* 9 OSK5912_AIC23_DIGACT     Digital interface activation     */ \

};
```

The ten values correspond to the ten internal configuration registers on the AIC23. Changing the values allows the codec to be set up in different modes or enable and disable certain functions. Parameters can be set statically with the OSK5912_openCodec( ) function or dynamically with the OSK5912_config( ) function. Please see the help section on the AIC23 for full definitions of the registers.

Compact Flash Test Overview

The Compact Flash test requires a compact flash card inserted below the OSK5912. The primary purpose of the test is to ability to read and write to the Compact Flash Card.

NOTE: this test will write to the first page of the compact flash card, this page is usually reserved for file system data and can corrupt the file system. You can avoid this by commenting the CF_test() call or by using a compact flash card with no file system.

Perform the following steps to run Compact Flash Test:

1) Load project [**cftest.pjt**]

   [To Load Project] Select the tab **Project → Open**, then select **cftest.pjt** in the directory **`c:\ti\boards\osk5912\examples\cftest`**.]

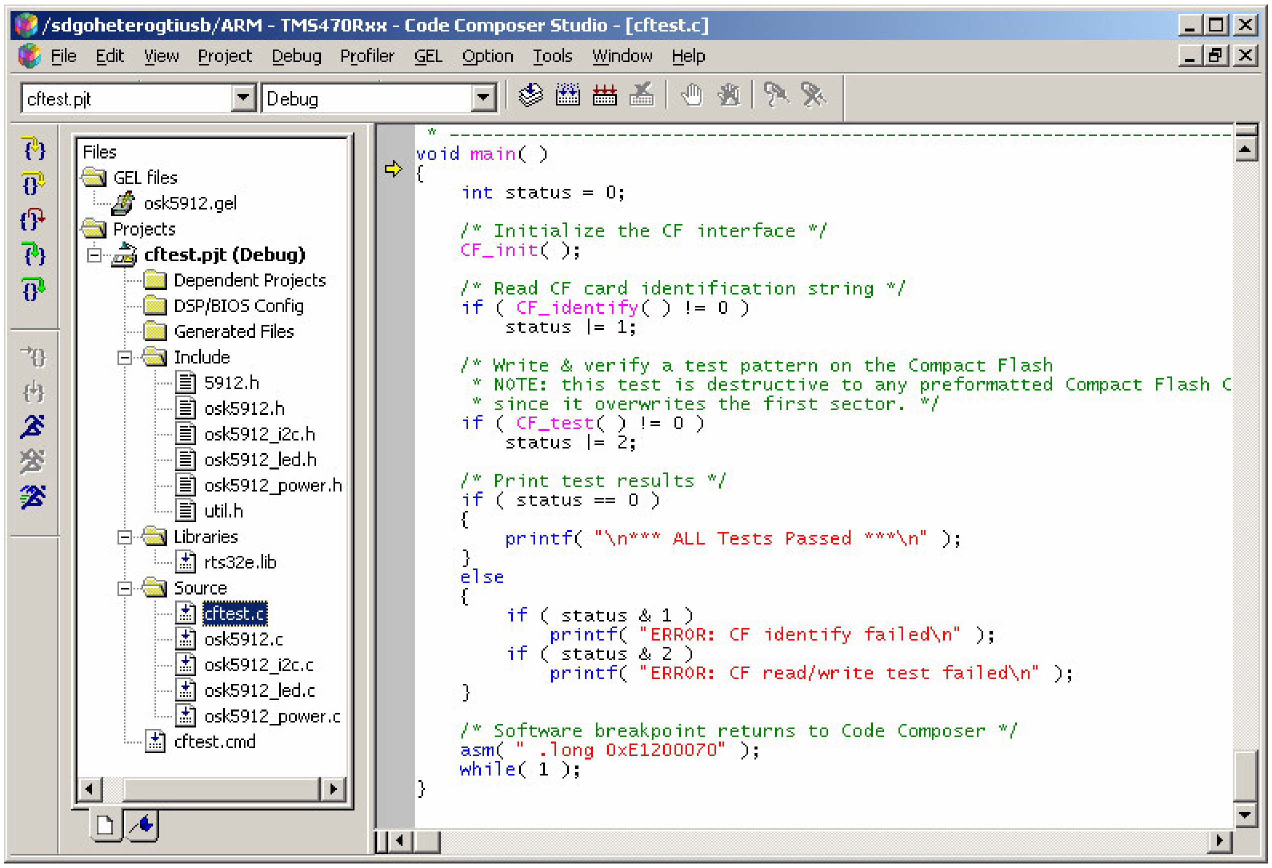
2) Load program [**cftest.out**]

   [To Load Program] Select tab **File → Load Program**. It will open a file browser dialog. Select the **cftest.out** file in the Debug directory in the file browser and hit "**Open**" to load.

3) Run program [**cftest.out**]

   [To Run Program] Select tab **Debug → Run**. The results will be shown in the 'Output Window'.

Below is a representation of cftest.pjt once loaded with Code Composer.



```
 * --------------------------------------------------------------------------
void main( )
{
    int status = 0;

    /* Initialize the CF interface */
    CF_init( );

    /* Read CF card identification string */
    if ( CF_identify( ) != 0 )
        status |= 1;

    /* Write & verify a test pattern on the Compact Flash
     * NOTE: this test is destructive to any preformatted Compact Flash C
     * since it overwrites the first sector. */
    if ( CF_test( ) != 0 )
        status |= 2;

    /* Print test results */
    if ( status == 0 )
    {
        printf( "\n*** ALL Tests Passed ***\n" );
    }
    else
    {
        if ( status & 1 )
            printf( "ERROR: CF identify failed\n" );
        if ( status & 2 )
            printf( "ERROR: CF read/write test failed\n" );
    }

    /* Software breakpoint returns to Code Composer */
    asm( " .long 0xE1200070" );
    while( 1 );
}
```

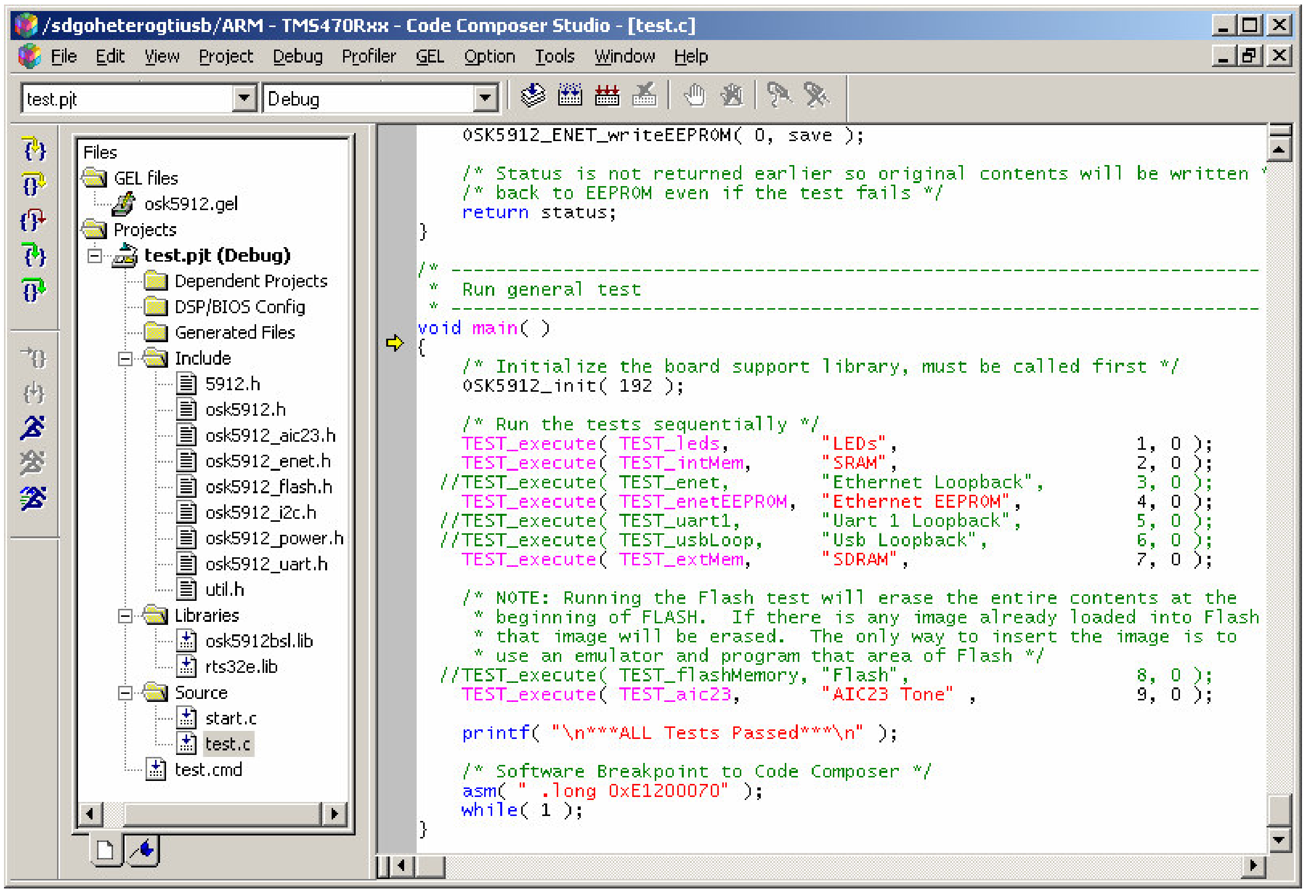
Zoom In if needed

General Test Overview

A variant of the board's production test code is provided with the OSK5912 as a confidence test (every board passes a superset of these tests before it ships) and as an example of how to manipulate the various board components. The test project is located at:

`c:\ti\boards\osk5912\examples\test`

The general test will run a series of tests as shown here:



```
    OSK5912_ENET_writeEEPROM( 0, save );

    /* Status is not returned earlier so original contents will be written *
    /* back to EEPROM even if the test fails */
    return status;
}

/* ------------------------------------------------------------------------
 *  Run general test
 * ------------------------------------------------------------------------
void main( )
{
    /* Initialize the board support library, must be called first */
    OSK5912_init( 192 );

    /* Run the tests sequentially */
    TEST_execute( TEST_leds,        "LEDs",                     1, 0 );
    TEST_execute( TEST_intMem,      "SRAM",                     2, 0 );
  //TEST_execute( TEST_enet,        "Ethernet Loopback",        3, 0 );
    TEST_execute( TEST_enetEEPROM,  "Ethernet EEPROM",          4, 0 );
  //TEST_execute( TEST_uart1,       "Uart 1 Loopback",          5, 0 );
  //TEST_execute( TEST_usbLoop,     "Usb Loopback",             6, 0 );
    TEST_execute( TEST_extMem,      "SDRAM",                    7, 0 );

    /* NOTE: Running the Flash test will erase the entire contents at the
     * beginning of FLASH.  If there is any image already loaded into Flash
     * that image will be erased.  The only way to insert the image is to
     * use an emulator and program that area of Flash */
  //TEST_execute( TEST_flashMemory, "Flash",                    8, 0 );
    TEST_execute( TEST_aic23,       "AIC23 Tone" ,              9, 0 );

    printf( "\n***ALL Tests Passed***\n" );

    /* Software Breakpoint to Code Composer */
    asm( " .long 0xE1200070" );
    while( 1 );
}
```

Zoom In if needed

Perform the following steps to run the General Test:

1) Load project [**test.pjt**]

   [To Load Project] Select the tab **Project → Open**, then select test.pjt in the directory **`c:\ti\boards\osk5912\examples\test`**.]

2) Load program [**test.out**]

   [To Load Program] Select tab **File → Load Program**. It will open a file browser dialog. Select the test.out file in the Debug directory in the file browser and hit "**Open**" to load.

3) Run program [**test.out**]

   [To Run Program] Select tab **Debug → Run**. The test will output which tests have run correctly.

   [NOTE: Upon completion, the general test will hit the breakpoint at the bottom of the file.]

Test Descriptions

Index	Test	Description
1	LEDs	Alternates the LEDs for a few seconds between ON and OFF.
2	SRAM	Runs pattern, address and walking 1s tests on the SRAM.
3	Ethernet Loopback	Simple loop back test for ETHERNET, requires external loop back connector.
4	Ethernet EEPROM	Runs a pattern to verify the Ethernet EEPROM is working.
5	UART 1 Loopback	Simple loop back on UART, requires external loop back connector, connecting UART.TX & UART.RX signals.
6	USB Loopback	Simple loop back on USB, requires external loop back connector connecting USB.DP & USB.DM signals.
7	SDRAM	Runs pattern, address and walking 1s tests on the SDRAM.
8	FLASH	Erases and programs the Flash with a pattern.
9	AIC23 Tone	Generates a 1KHz sine wave and outputs it on the HEADPHONE jack.

Configuring the Tests

Only tests that can complete without external connections are, by default, enabled. The Flash test is also disabled because it will erase the any data in Flash Memory. To enable/disable a test simply comment or uncomment the corresponding TEST_execute( ) call in main( ).

External Connectors

Ethernet Loopback test: Requires an external Ethernet loop back connector attached to act as a loop back path. This connector has TD+ connected to TD- (pin 1 connected to pin 3) and RD+ connected to RD- (pin 2 connected to pin 6). This test is primarily a check for a working Ethernet connector and to its ability to send and receive data.

UART 1 Loopback test: Requires an external RS232 female loop back connector attached to act as a loop back path. The connector has TX connected to RX (pin 2 connected to pin 3). This test is primarily a check for a working UART connector and to its ability to send and receive data.

USB Loopback test: Requires an external USB loop back connector attached to act as a loop back path. This connector has DP connected to DM (pin 2 connected to pin 3). This test is primarily a check for a working USB connector.

AIC23 Tone test: Requires a headphone or speaker to hear to 1 KHz tone.

Rebuild warning:

Upon rebuilding the general test, the compiler will respond with a warning such as:

```
>> warning: entry point other than _c_int00 specified
```

It means that the normal entry point to the project (_c_int00) has been replaced with a new entry point. In the case with the general test, the entry point is (_start) located in (start.c). Start.c is a special assembly file written in inline assembly code, with the purpose of providing the necessary setup for OSK5912 that cannot be written in normal C code. These operations include configuring & enabling the stacks, disabling the MMU, and disabling the Interrupts.

Setmac Overview

Setmac is included with the BSL examples to set the Ethernet MAC address. The MAC address on each board is shipped with what is written on the front of the OSK5912 labeled MAC ADDRESS 00-0E-99-##-##-##.

Perform the following steps to run Setmac:

1) Load project [**setmac.pjt**]

   [To Load Project] Select the tab **Project → Open**, then select **setmac.pjt** in the directory `c:\ti\boards\osk5912\examples\setmac`.]

2) Modify MAC address and Re-compile project

   [To Modify MAC address] Open **setmac.c** and go to the main( ) function. In main( ), go to where MAC_newaddr[ ] is being set. Verify that 6 entries in the array correspond to the 6 numbers on the front of the OSK5912, if not change the code to match.

   [To Compile] Select **Project → Build**. A status of the build will pop up. When no errors are found the project has successfully compiled.

   [NOTE: To avoid conflicting with another board's MAC address, the MAC_newaddr[ ] array must be the same value on the label.]

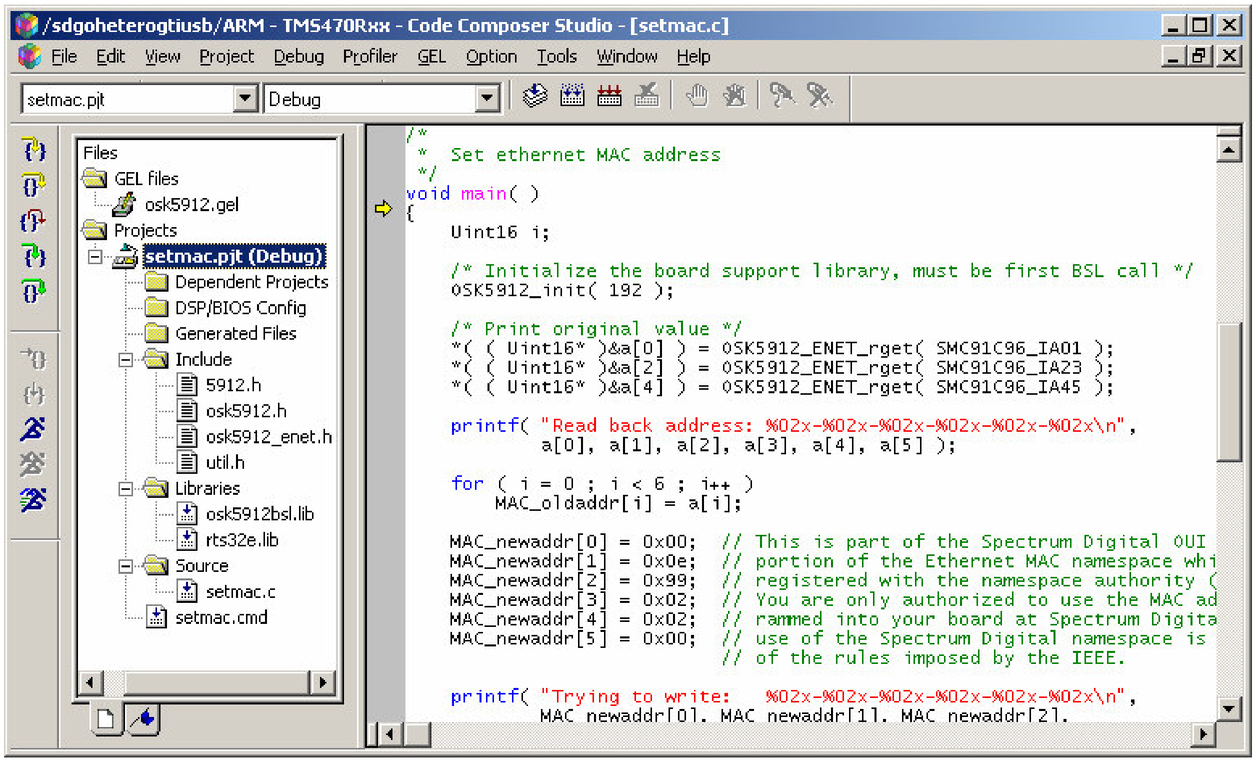
3) Load program [**setmac.out**]

   [To Load Program] Select tab **File → Load Program**. It will open a file browser dialog. Select the **setmac.out** file in the Debug directory in the file browser and hit "**Open**" to load.

4) Run program [**setmac.out**]

   [To Run Program] Select tab **Debug → Run**. The results will be shown in the 'Output Window'.

Below is a representation of **setmac.pjt** once loaded with Code Composer.



```
/*
 *  Set ethernet MAC address
 */
void main( )
{
    Uint16 i;

    /* Initialize the board support library, must be first BSL call */
    OSK5912_init( 192 );

    /* Print original value */
    *( ( Uint16* )&a[0] ) = OSK5912_ENET_rget( SMC91C96_IA01 );
    *( ( Uint16* )&a[2] ) = OSK5912_ENET_rget( SMC91C96_IA23 );
    *( ( Uint16* )&a[4] ) = OSK5912_ENET_rget( SMC91C96_IA45 );

    printf( "Read back address: %02x-%02x-%02x-%02x-%02x-%02x\n",
            a[0], a[1], a[2], a[3], a[4], a[5] );

    for ( i = 0 ; i < 6 ; i++ )
        MAC_oldaddr[i] = a[i];

    MAC_newaddr[0] = 0x00;  // This is part of the Spectrum Digital OUI
    MAC_newaddr[1] = 0x0e;  // portion of the Ethernet MAC namespace whi
    MAC_newaddr[2] = 0x99;  // registered with the namespace authority (
    MAC_newaddr[3] = 0x02;  // You are only authorized to use the MAC ad
    MAC_newaddr[4] = 0x02;  // rammed into your board at Spectrum Digita
    MAC_newaddr[5] = 0x00;  // use of the Spectrum Digital namespace is
                            // of the rules imposed by the IEEE.

    printf( "Trying to write:   %02x-%02x-%02x-%02x-%02x-%02x\n",
            MAC_newaddr[0], MAC_newaddr[1], MAC_newaddr[2],
```

Zoom In if needed