SignalMaster Manual

Version 20190410

SignalMaster

Hardware Version 2.00-2019 PN: M072005
Released Mar. 2019
Blue Tooth Enabled

Intelligent Hearing Systems Corp.

6860 S.W. 81st Street – Miami, FL 33143 - USA

Introduction:

SignalMaster was developed under a United States Department of Health, National Institutes of Health (NIH), National Institute on Deafness and Other Communication Disorders (NIDCD), Small Business Innovation Research (SBIR) grant to Intelligent Hearing Systems, Corp. The purpose of the grant was to develop an open architecture hardware and software system that will allow researchers to conduct a wide range of studies involving multi-channel acoustic input and output for psychophysical research and hearing aid development. The current system provides users with the ability to develop their own processing algorithms by providing source code examples and the ability to upload programs to the device and communicate with a user developed control program on an external computer. The hardware provides two input microphone and two output speakers, analog channels with up to 32 bit digital conversion resolution and is able to operate independently from battery power. The system provides sufficient processing capabilities to perform real-time signal processing with minimal delay (or latency) between input and output with various point-by-point or batch processing capabilities.

SignalMaster is based on a Texas Instruments (TI) Digital Signal Processor (DSP), TMS320C6746. For full specifications, see the SignalMaster Specifications section below.

Web Site: http://ohsspds.ihsys.info/

Signal processing modules are available online for download by users from the following web address:

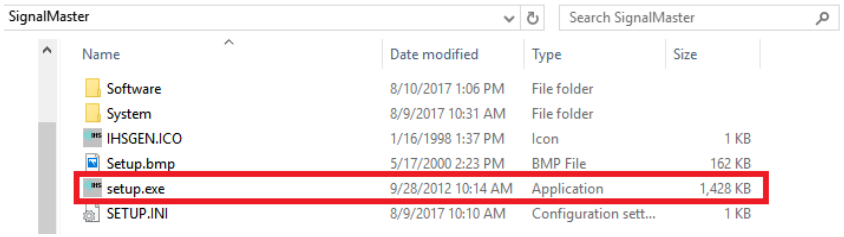
http://www.ihsys.com/ohsspds/index.asp

This site contains a list of documents that are available. The DSP source code documents (zip files) start with the name “SampleDSPCode_” follow by a descriptive name of the type of processing. These zip files contain all files required by the TI code composer to modify the DSP program. IHS provided executables can be used to upload any DSP code to the hardware.

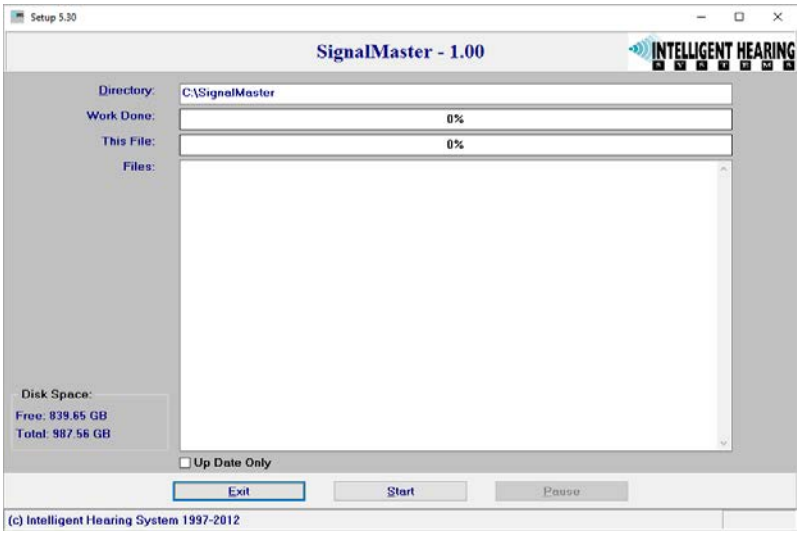
Getting Started:

Your SignalMaster system doesn't require any special drivers. Simply install the software from your SignalMaster thumb drive. The thumb drive contains example programs and libraries required for developing new applications.

Simply run the Setup.EXE application on the thumb drive:



After running the Setup program, press the start button to install the software to the C:\SignalMaster directory:



It is highly recommended to install the software in the default C:\SignalMaster directory as all the CCS example projects are pre-set to look for libraries in that directory. If you wish to install SignalMaster in a different directory, you will need to modify the default compiler and linking directories in CCS.

The installation utility will default to C:\SignalMaster and contains the following subdirectories:

C:\SignalMaster\...
DSPProjects – containing sample DSP applications
PCProjects – containing sample PC application with their corresponding DSP OUTFiles (For the source code, look under the DSPProjects directory of the corresponding project)
IHSC6748HW – containing software and hardware specific files

All PC programs developed require at a minimum the following files that are provided or can be developed by the user:

1) **MyProgram.EXE** - Your application program.
2) **MyDSPProgram.OUT** – Compiled TI DSP program generated using TI Code Composer.
3) **SIGMASDLL.DLL** – IHS DLL that provides DSP communication routines and ability to upload programs into the DSP.
4) **Out2rprc.EXE** – TI provided program (available from your SignalMaster distribution thumbdrive) used by the DLL to convert DSP compiled OUT files to BIN format for upload.
5) **IHSWIN.SYS** – IHS License file that contain information about your hardware. The DLL uses this file to determine if your hardware is available or not.

Using the provided DLL, you may develop PC based applications in any programming language you like, as long as that language supports DLLs. The DLL will allow you to transfer data to and from the DSP to your application program on your PC. It will also allow you to send user defined command instruction to the DSP.

Before you can continue to develop any programs for SignalMaster, you will need to install TI Code Composer Studio (CCS). The following section will walk you through the installation process.

Installing Code Composer Studio (CCS):

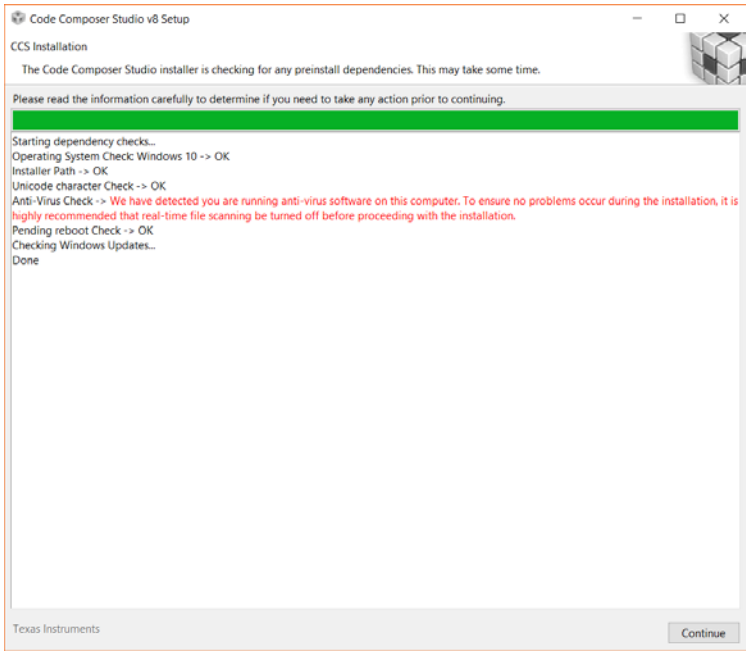
Sections:

- **Installing Code Composer Studio**
- **Importing Projects**
- **Setting Up The User Interface**
- **Installing the Compiler**
- **Updating Project Compiler Settings**
- **Compiling a Project**
- **Loading and Running a Project**
- **Starting a New Project**
- **Selecting a New Work Space**
- **Code Composer Studio Compiler Settings (Advanced Options):**

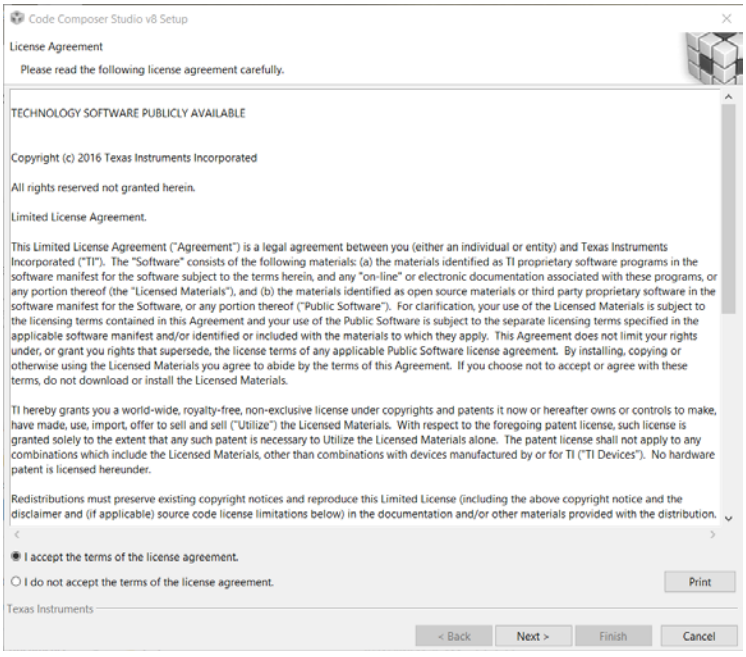
Installing Code Composer Studio:

1. Download the offline version of the Code Composer Studio (CCS) installer from http://processors.wiki.ti.com/index.php/Download_CCS

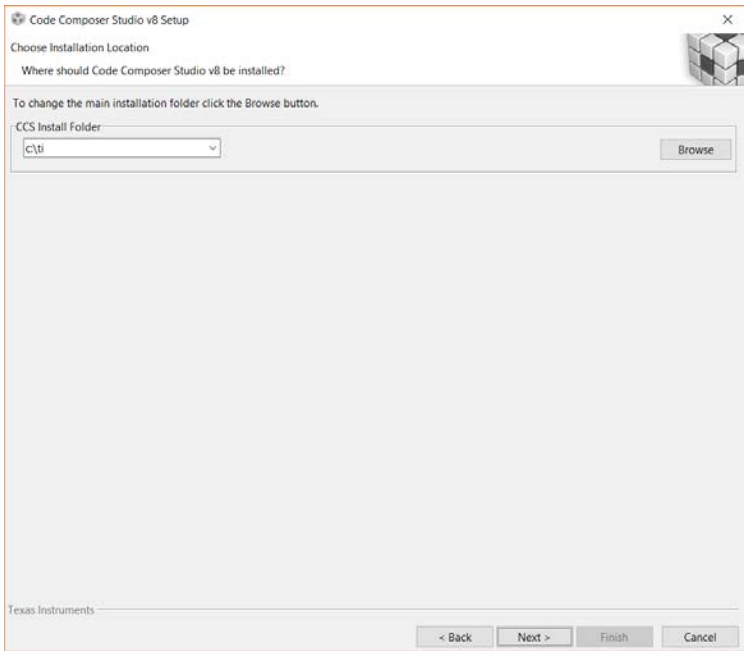
2. Begin installing the application. If you are greeted with a window like the one below chances are you'll be fine to just click Continue: *Note: If you do run into an error, as shown below, then turn off your anti-virus and try again.*



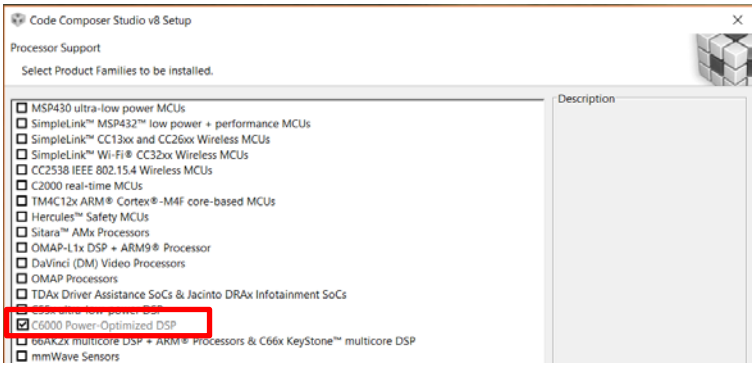
3. Now click Accept and Next:



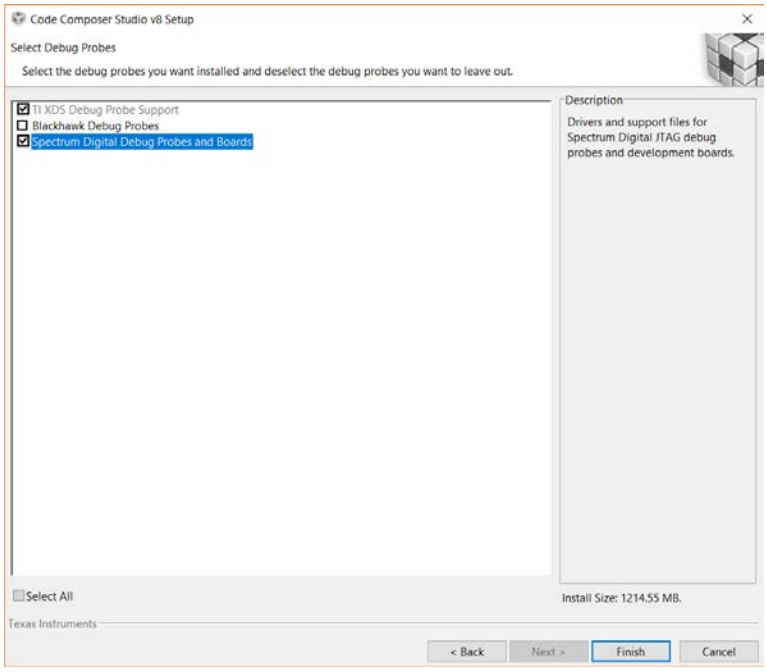
4. Click Next again:



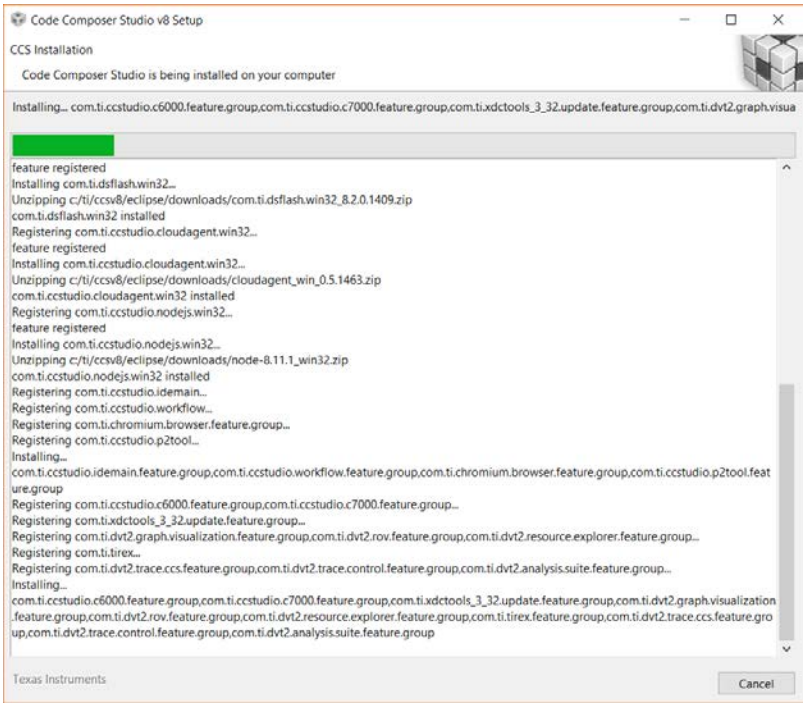
5. Select C6000 Power-Optimized DSP when prompted, and click Next:



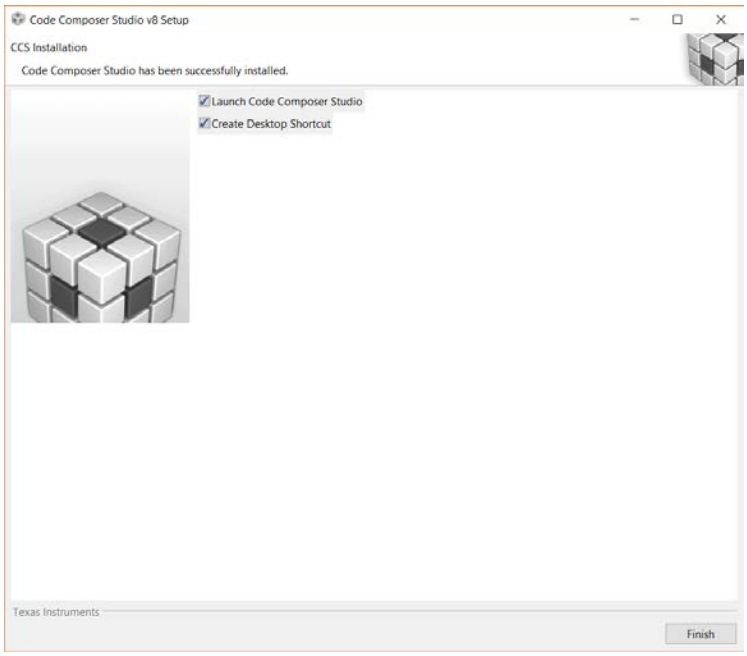
6. Enable TI XDS Debug Probe Support and Spectrum Digital Debug Probes and Boards, and click Finish:
   Note: Allow internet access if prompted.



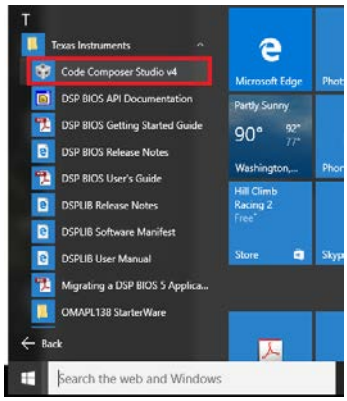
7. The application will now take some time to perform the installation… Wait until it finishes…



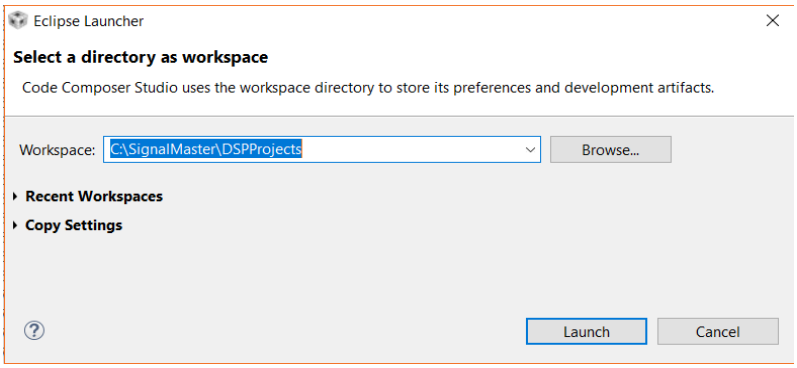
8. Click Finish:



You have now completed the installation process for the interphase portion of CCS. There are still a few more items to take care of, but now you can run CCS. To Run CCS, simple select the CCS icon on your desktop or select from available applications:



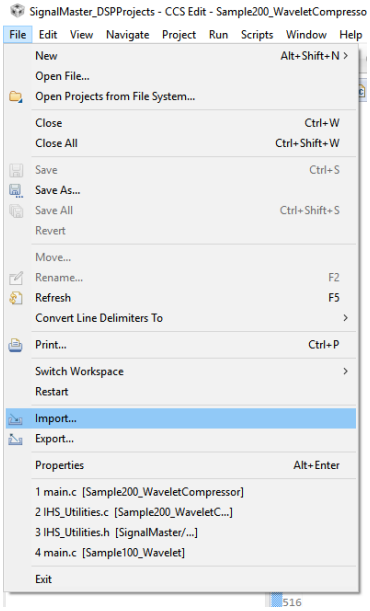
When running CCS for the first time, enter C:\SignalMaster\DSPProjects as the default workspace directory.



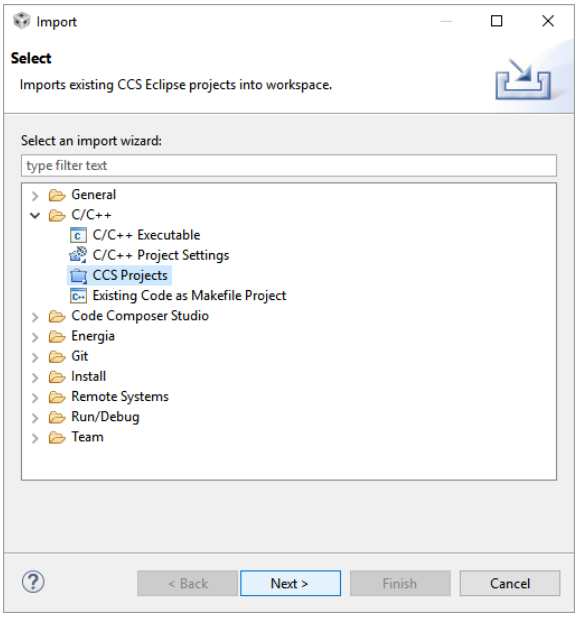
Importing Projects:

From within CCS:

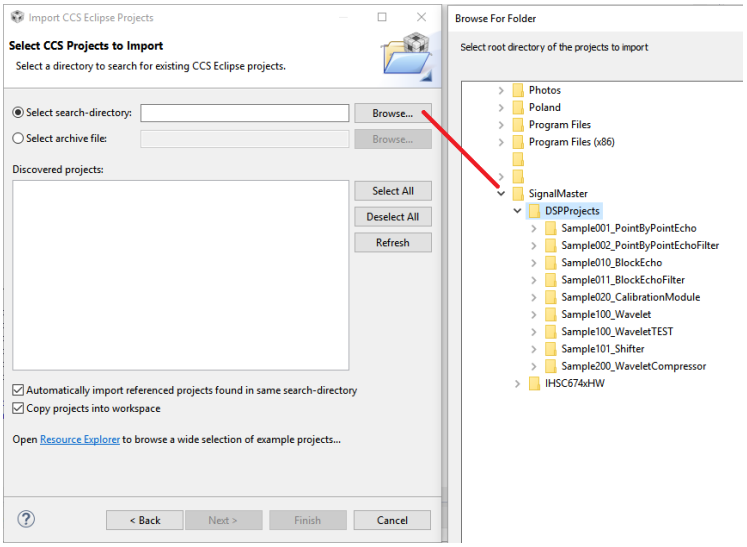
1. Click File > Import…



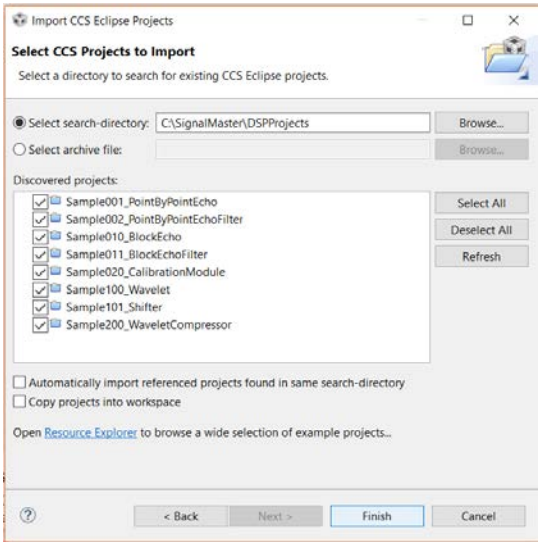
2. Click to C/C++ > CCS Projects
3. Click Next



4. For “Select search-Directory:” Browse to C:\SignalMaster\DSPProjects. This will show you all of the projects contained in the DSPProjects folder.

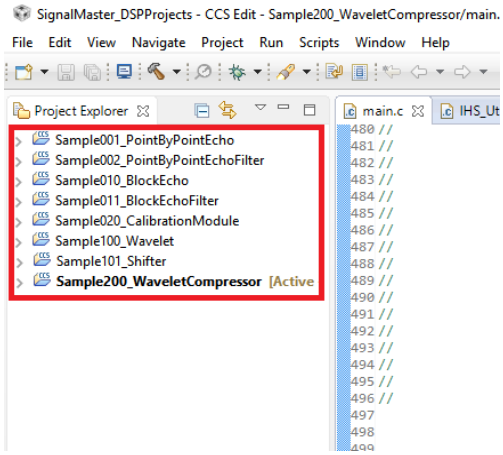


5. Click Select All: The dialog box should look similar to how it does in the picture below.



(Note: Current 2019 example program names begin with name Samp3xxx)

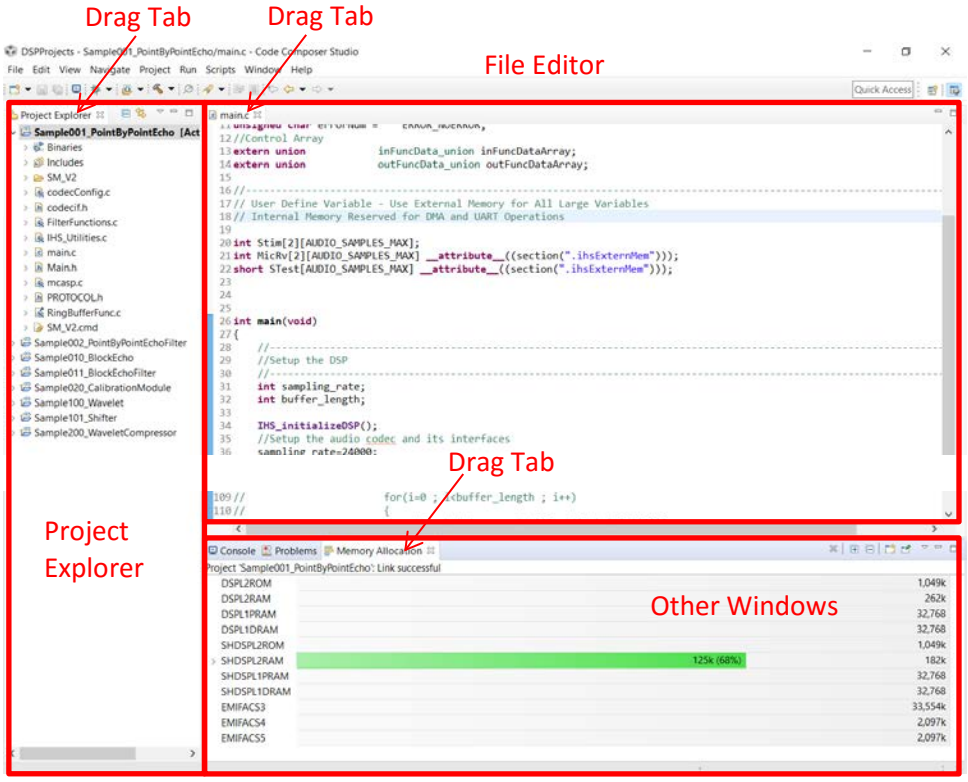
6. Click Finish. The demonstration projects we supplied should now show in the project explorer window.



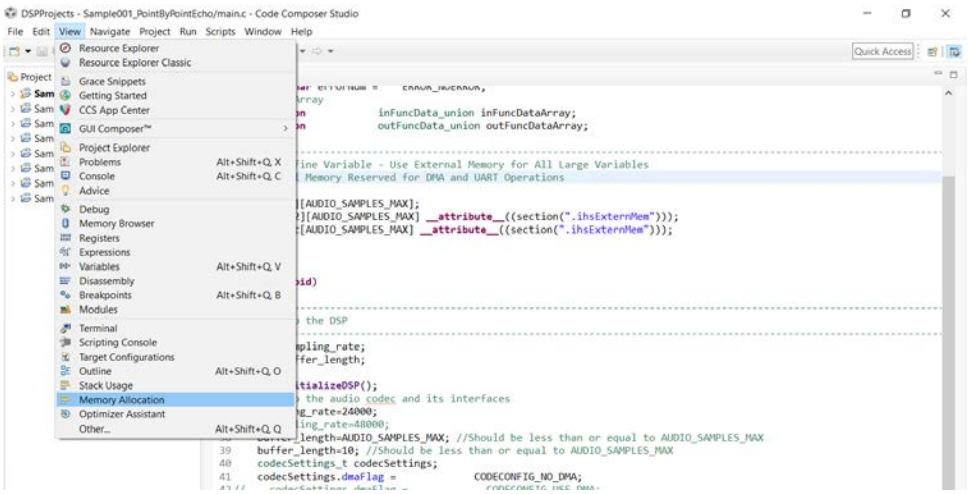
(Note: Current 2019 example program names begin with name Samp3xxx)

Setting Up The User Interface:

We find that CCS by default has its windows arranged oddly. We therefore recommend the following rearrangement for better ease of use. Drag and drop the respective tabs by clicking on the drag marker until they are arranged as shown or personally desired.



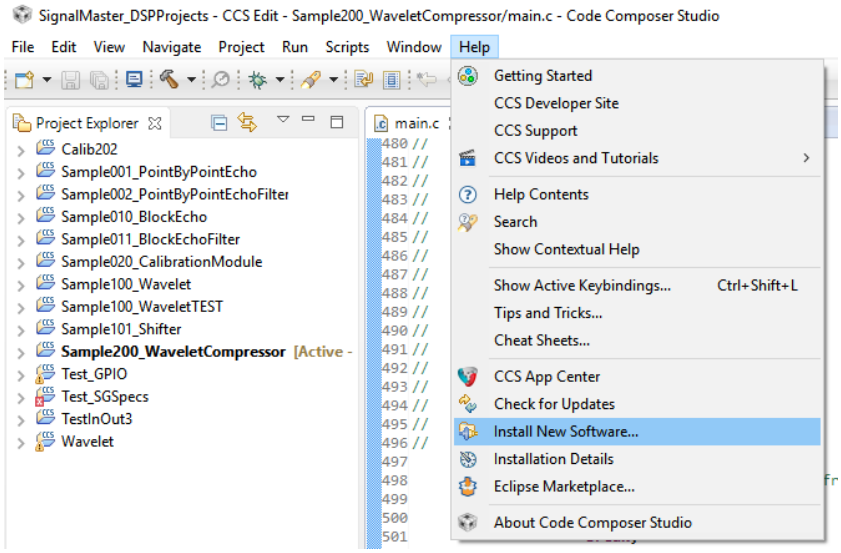
If a window is not visible, go to View menu and select any window you may want:



Installing the Compiler:

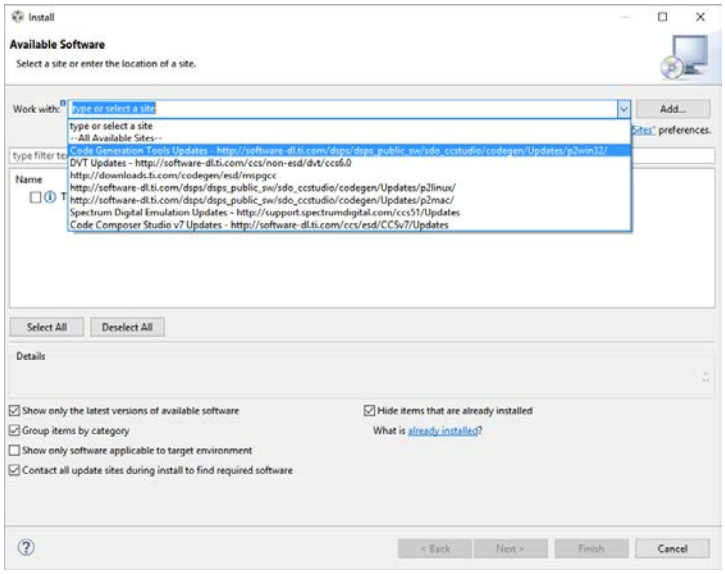
After you have installed CCS, you will also need to install the **C6000 Compiler Module**. This module contains the actual c language compiler for the TI family of DSP chips used by SignalMaster. CCS will not be able to compile your code without that module.

1. Click Help > Install New Software:

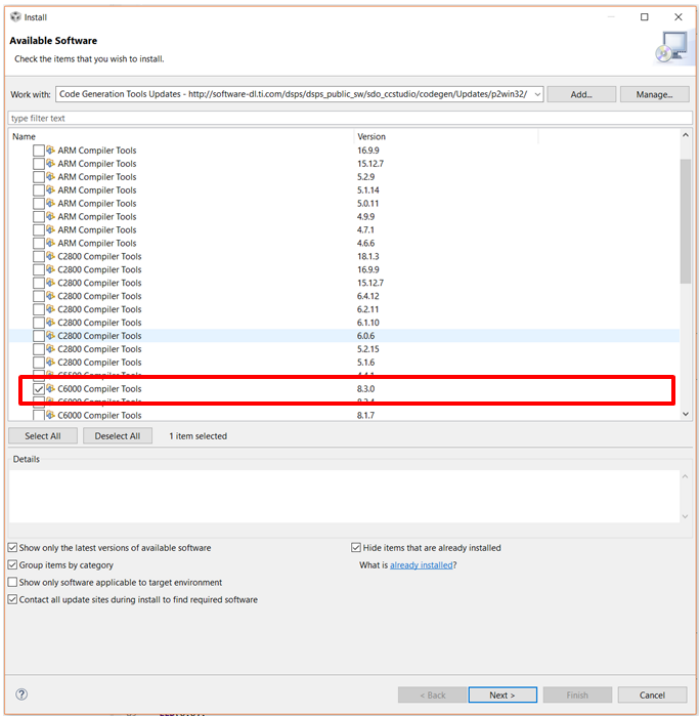


(Note: Current 2019 example program names begin with name Samp3xxx)

2. For “Work with”, Select Code Generation Tools Updates:



3. Expand TI Compiler Updates, and then click the latest version of C6000 Compiler Tools. The dialog window should look like the picture shown below.



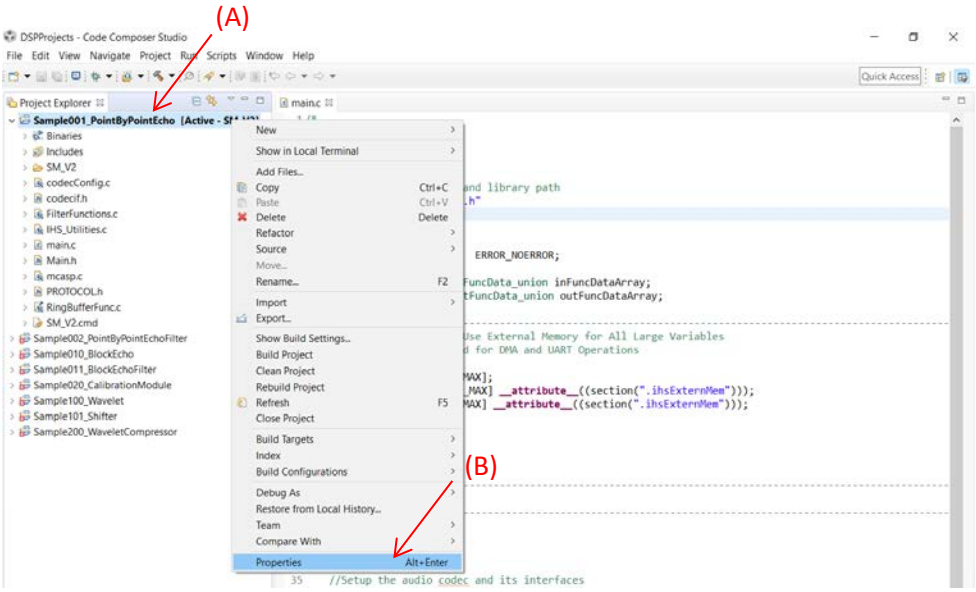
Note: Demonstration projects may have been compiled and supplied with a previous compiler version. This should not be problem. Continue to select the latest compiler. The next section will cover how to handle compiler discrepancies.

4. Click Next, then next again, then accept, then finish.

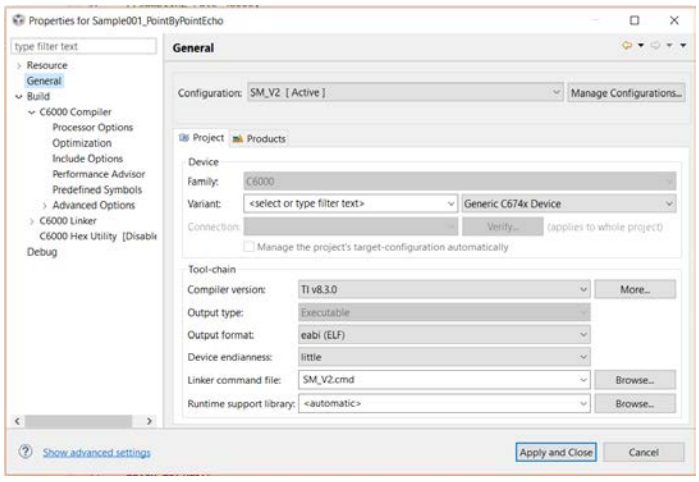
Updating Project Compiler Settings:

Chances are the latest compiler version has changed since we supplied the demonstration projects. In this case the projects will by default throw an error “This project was created using a version of compiler that is not currently installed…” . If this is the case, then we must update the project’s compiler settings.

1. Right click the project in the Project Explorer window (A) and select Properties (B).



2. Under General > Project > Tool-Chain > Compiler Version select the compiler version you just installed or wish to use. The dialog option should look similar to the picture below.



3. Click Apply & Close

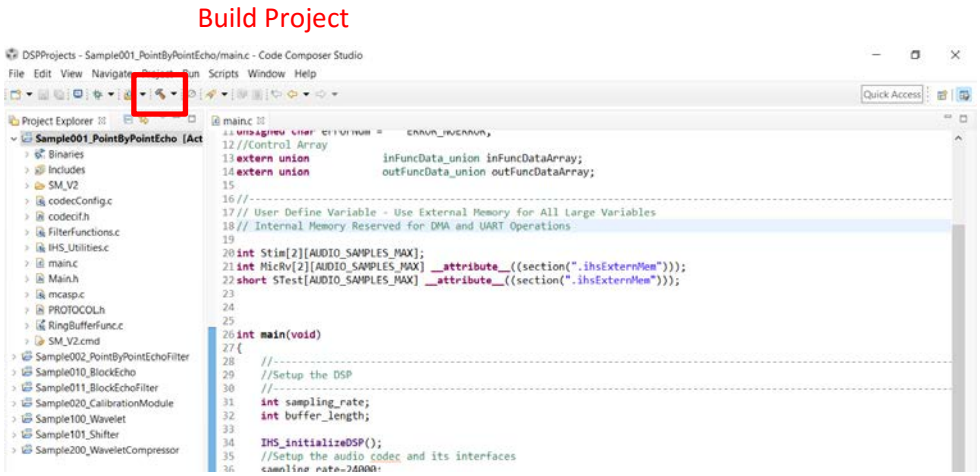
Compiling A Project:

This process will generate an .OUT file which will then be used by the PC applications to program the SignalMaster device with the DSP code you wrote. If no errors occurred while compiling, then the .OUT file is automatically stored in the project folder by Code Composer.

Example .OUT file location:

C:\SignalMaster\DSPProjects\Sample001_PointByPointEcho\SM_V2\Sample001_PointByPointEcho.out

1. Click the Build / Compile button.

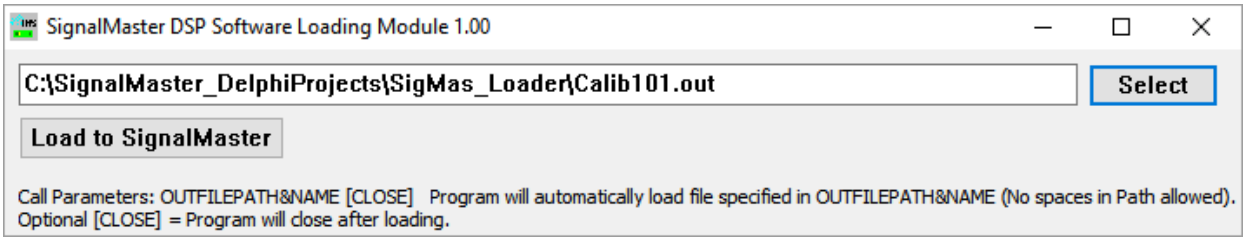


Loading and Running a Project:

After you have compiled your application, you are ready to load and run the file on the SignalMaster hardware. You can load your programs using the SigMasLoader.EXE program. This program is found in the SignalMaster installation directory.

Make sure that the SignalMaster hardware is connected to one of your computer's USB ports and is turned on. The top left LED on the box will start to blink when it is turned on and ready to accept a program. If you do not see the LED blinking or if you need to reset the hardware at any time, you can toggle the on/off switch until you see the LED blinking.

Use the select button to browse to the location of your compiled .OUT file. Then select the Load to SignalMaster button to upload the program. The program will automatically start running.

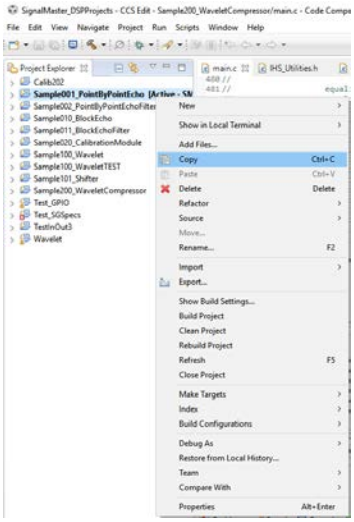


Note: Although CCS provides an option to load and run programs directly from within CCS, you cannot perform this option with the provided USB cable. Direct CCS hardware control and debugging options requires the use of a special Joint Test Action Group (JTAG) cable. We do not recommend using this option as it requires having your SignalMaster system hardware open. If you would like to use a JTAG cable with your SignalMaster hardware, please contact IHS for further information.

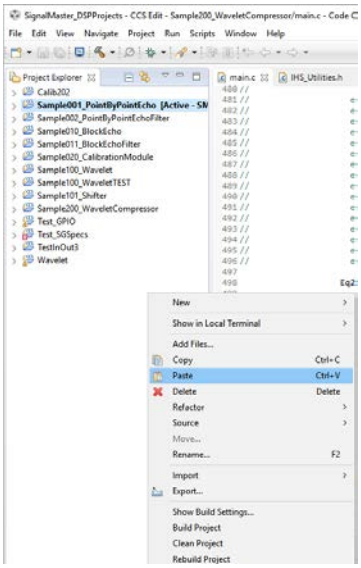
Starting a New Project:

When generating a new SignalMaster DSP program, we recommend that you start out from one of the provided examples that is most similar to what you want to accomplish and copy that project.

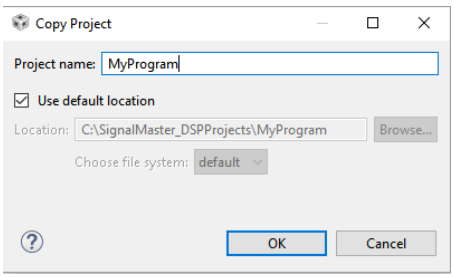
1. Simply right mouse click over the project you wish to copy and select the copy option on the popup menu:



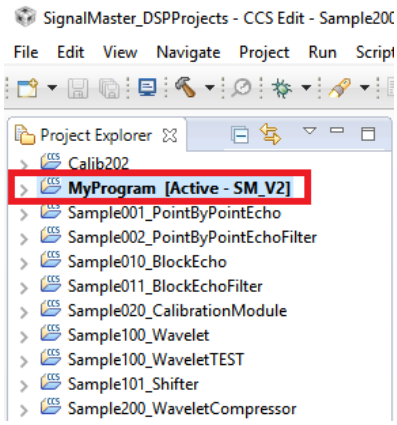
2. Now right mouse click over an empty region of the window showing the list of projects and select paste from the popup menu:



3. Enter the name of your new project:



4. Your new program will now appear in the list of projects:



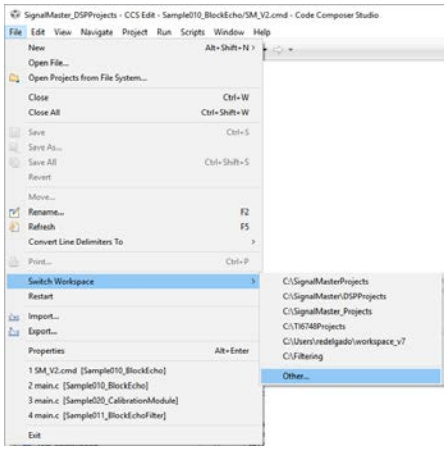
(Note: Current 2019 example program names begin with name Samp3xxx)

Selecting a New Workspace:

During installation, CCS will ask you to select a Workspace Directory. We recommend that you use the following Workspace Directory as the default:

C:\ SignalMaster\DSPProjects

If for any reasone, you need to have multiple work spaces, you can do this from the File menu, Switch Workspace option.

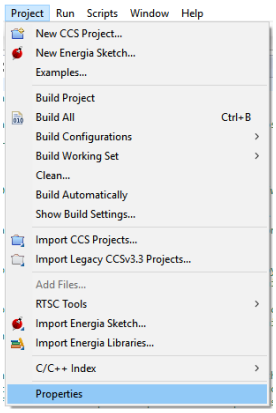


Code Composer Studio Compiler Settings (Advanced Options):

When you load one of the SignalMaster example programs into CCS, all required settings should automatically be preset in the project files. This section is only provided for reference in case you need to make any changes and so that settings can be verified.

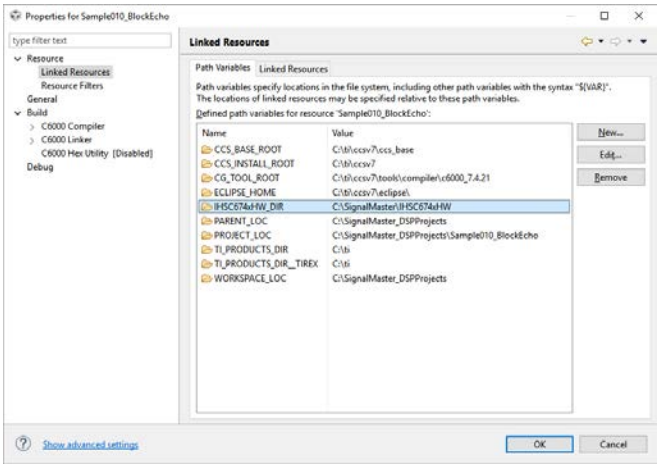
Setting up the complier and linker options:

The following items are under the Project Menu, Properties Item:

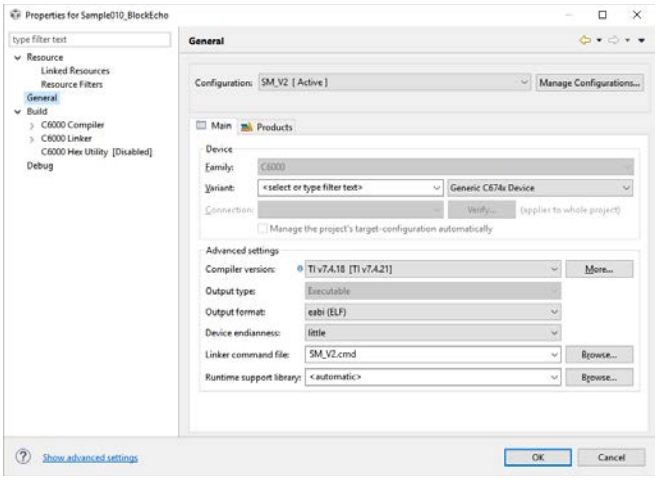


After selecting this option, a Properties dialog box will appear showing options for resources and building your applications.

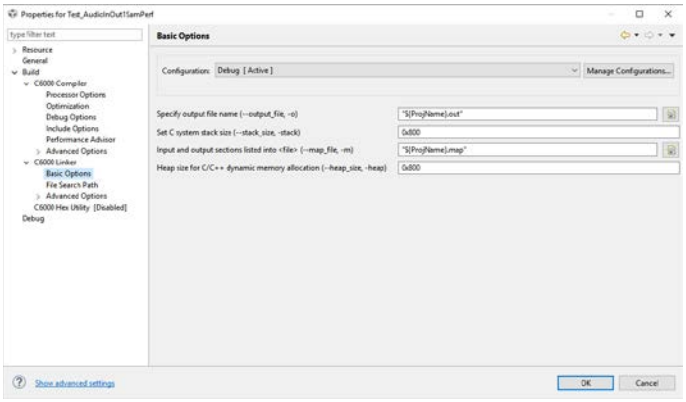
Make sure that the variable **IHSC674xHW_DIR** is declared and pointing to the SignalMaster installation directory: C:\SignalMaster\IHSC674xHW where the hardware support libraries are located. This variable is used in the projects in order to easily point to the directory where SignalMaster has been installed containing all the required files.



Make sure that the rest of the dialog windows are set as shown in the images below:

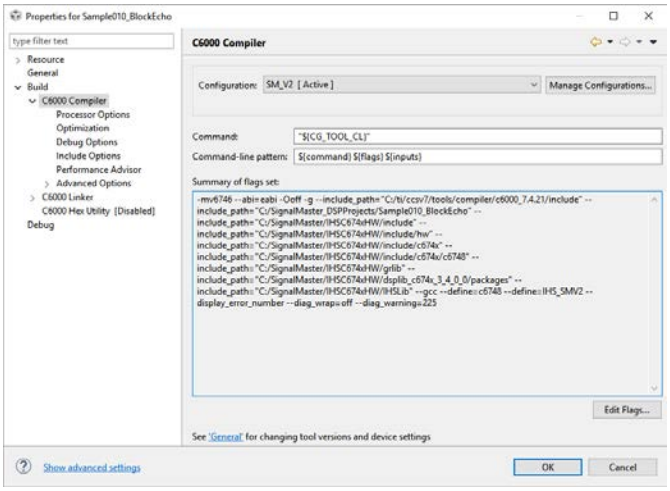


The Linker command file should be **SM_V2.CMD** – this file contains the memory map for the hardware and other important parameters needed to compile your application.



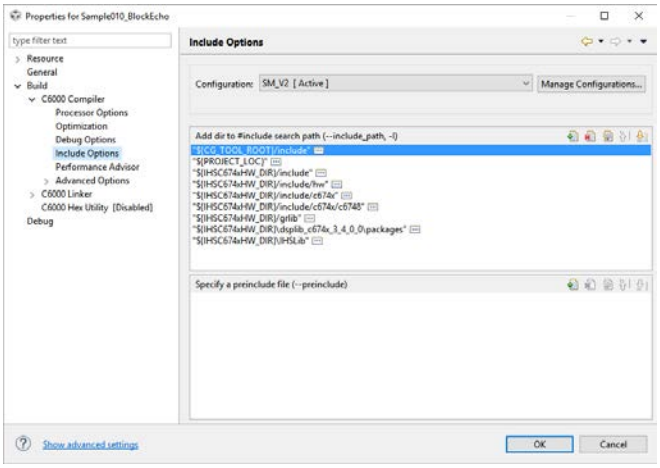
<u>Compiler Options</u>:

Note that in the compiler and linker options, the path variable is shown in the include search path.

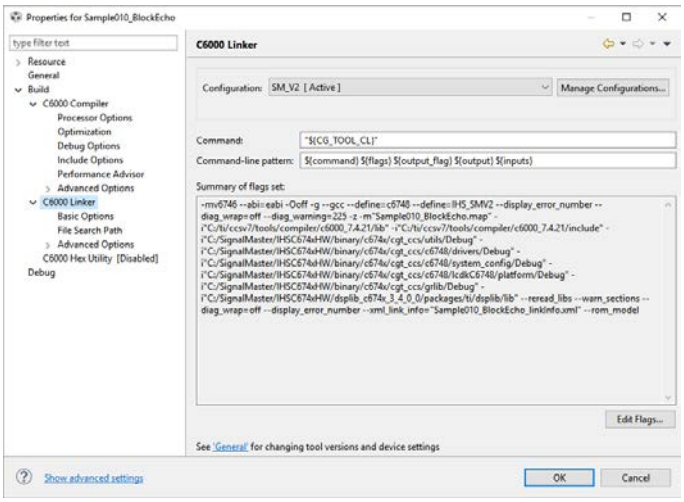


Make sure that the following are specified under the compiler options:

-mv6746 --abi=eabi -Ooff -g --include_path="C:/ti/ccsv7/tools/compiler/c6000_7.4.21/include" --include_path="C:/SignalMaster_DSPProjects/Sample010_BlockEcho" --include_path="C:/SignalMaster/IHSC674xHW/include" --include_path="C:/SignalMaster/IHSC674xHW/include/hw" --include_path="C:/SignalMaster/IHSC674xHW/include/c674x" --include_path="C:/SignalMaster/IHSC674xHW/include/c674x/c6748" –include_path="C:/SignalMaster/IHSC674xHW/grlib" --include_path="C:/SignalMaster/IHSC674xHW/dsplib_c674x_3_4_0_0/packages" --include_path="C:/SignalMaster/IHSC674xHW/IHSLib" --gcc --define=c6748 --define=IHS_SMV2 –display_error_number --diag_wrap=off --diag_warning=225



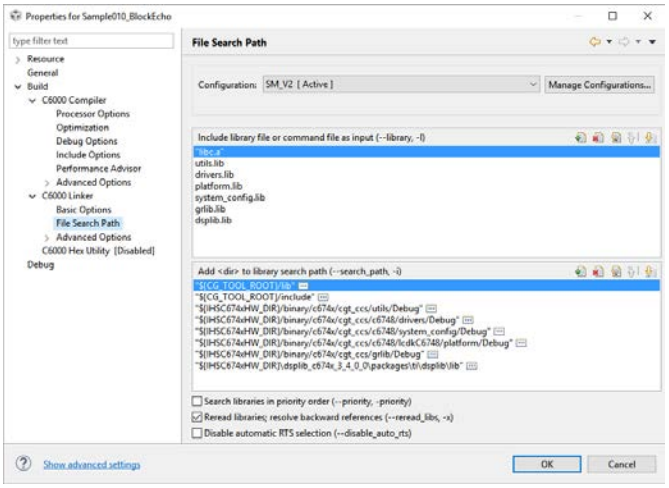
Linker Options:



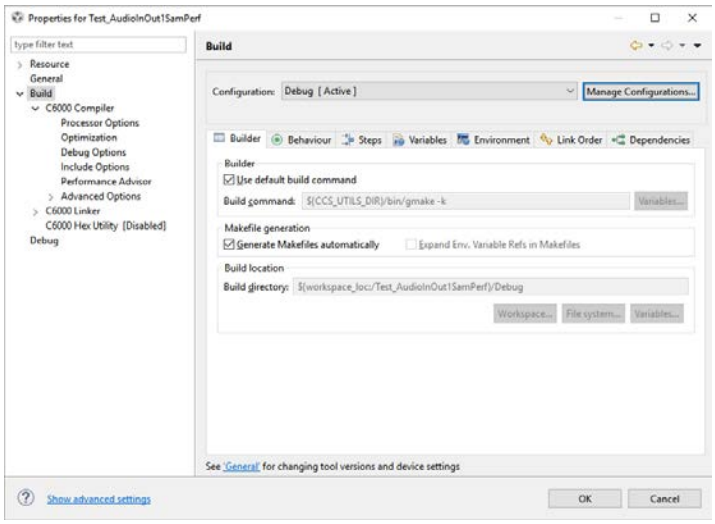
Make sure that the following hardware specific paths are specified:

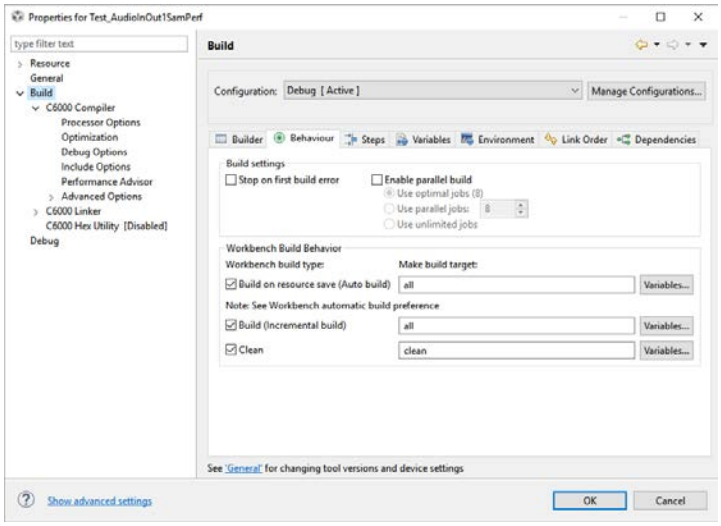
-i"${IHSSignalMaster}/IHSC6748HW/binary/c674x/cgt_ccs/utils/Debug"
-i"${IHSSignalMaster}/IHSC6748HW/binary/c674x/cgt_ccs/c6748/drivers/Debug"
-i"${IHSSignalMaster}/IHSC6748HW/binary/c674x/cgt_ccs/c6748/system_config/Debug"
-i"${IHSSignalMaster}/IHSC6748HW/binary/c674x/cgt_ccs/c6748/lcdkC6748/platform/Debug"
-i"${IHSSignalMaster}/IHSC6748HW/binary/c674x/cgt_ccs/grlib/Debug"

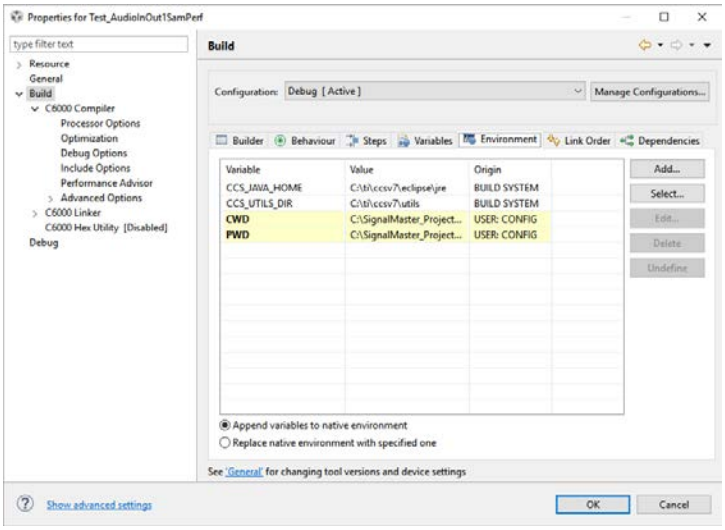
Linker Include Libraries:



Make sure that the following libraries are shown:
"libc.a"
utils.lib
drivers.lib
platform.lib
system_config.lib
grlib.lib







Note that in the above image, CCS_UTILS_DIR shows, C:\ti\ccsv7\utils – the ccsv7 is a reference to the version (version 7) of CCS that was installed at the time the image was generated. If you have installed a later or earlier version of CCS, that string may read ccsv8 (for version 8). This will not affect the compilation of your program.

Other tabs: Steps, Variables. Link Order and Dependencies should be blank.

Project Menu:

Properties…

Include Options – make sure that the correct directory is shown in the path

Developing PC Applications:

PC applications can be developed in any programming language that allows the use of a DLL. These include all high level languages such as C++, Pascal and Visual Basic. Other programming environments such as Matlab and LabView also allow the use of DLLs. The DLL provides routines to load programs to the DSP and to communicate with the DSP during execution. For a full list of DLL commands and data types, see the **IHS SignalMaster DLL Commands** section below.

Every PC application will need to use DSP program. The DSP program receives commands from the PC and either uploads data from the PC or downloads data to the PC at the PC's request. Applications can be written to present sounds at different frequencies and intensities and record from the DSP microphones. The DSP can also work independently and automatically process the microphone signals and output them back through the earphones. The PC can also simply provide processing parameters to adjust band pass filters to change the processing and characteristics of signals being presented to a subject.

After setting up the DLL declarations in your program as shown in the **IHS SignalMaster DLL Commands** section below, you will be able to load DSP programs and start sending commands and data to the DSP.

First load the DSP code by calling the SM_Initialize routine specifying the location of the .OUT file that is generated by the TI Code Composer Suite (CCS) (See **TI Code Composer Studio Setup** section below). Make sure that both the **our2sprc.exe** program and **SIGMAS.DLL** are in the same directory as your application.

```
SM_Initialize('C:\MyPath\MyDSPProgram.OUT');
```

You may also program a DSP application with all processing parameters coded within the DSP code and not use a PC program. In this case, you can simply load a DSP application by running the **SigMasLoader.EXE** program (See the **Loading Programs to the DSP** section below for additional information).

Sending Function Calls and Data to the DSP:

Data may be sent to DSP using one of several arrays types defined in the SignalMaster DLL.
To transfer bytes, simply use a byte array (TxArray in this example). You can load the values into the array:

```
TxArray[1]:= value1;
TxArray[2]:= value2;
 :
TxArray[n]:= valuen;
```

Then call the SM_PutByteArray command specifying the array and number of bytes.

```
SM_PutByteArray(TxArray,n);
```

You may also use a Short Integer (16 bit) and Integer (32 bit) type arrays with their corresponding calls.
After you perform the transfer call, the data is sent to the DSP, but you need to let the DSP know that it has data waiting to be process, so you will need to implement a function call:

```
SM_DSPFuncTx(MyFunctionNumber, n);
```

Where MyFunctionNumber is a function that you have defined in your DSP code and n is the number of bytes that the function will read. Make sure that the number of bytes (n) matches the actual number sent.

Requesting Data from the DSP:

To request data from the DSP, you will need to call another function you have defined for that purpose:

SM_DSPFuncTx(MyFunctionNumber, n);

In this case, n can be equal to 0 if you are not sending any parameters to that function. In some cases, you might want to read data from one microphone or another, you may specify the microphone number by first call filling in the corresponding data array and then transferring the parameters using the **SM_PutByteArray(TxArray,n)** function before calling the DSP function as done in the previous example.

After the function call, read the data by first calling:

n:=SM_DSPFuncRx(MyFunctionNumber);

The function will return the number of bytes (n) transferred by the DSP. Note that if you call multiple data request function calls without calling the SM_DSPFuncRx receive function, the previous data requests will be discarded.

After the SM_DSPFuncRx, you will need to transfer the actual data from the DSP to the PC application. Simply call the corresponding SM_Get function corresponding to the data type you wish to read:

SM_GetIntArray(RxArrayInt,(n/4));

In the above example, the Integer array RxArrayInt will be filled with n integers of data. Important note: The SM_DSPFuncRx function returns the number of bytes transferred, however, the SM_Get functions will read the number of data points for the corresponding data type. In this example, the data type is integer (32 bits = 4 bytes), therefore n is divided by 4 when calling the SM_GetIntArray function.

With these basic command, you will be able to develop complex applications that call on the SignalMaster DSP to perform any number of functions you define and transfer data to and from the DSP. Several PC application source code examples are provided in the PCProjects subdirectory under the various programming languages. Although currently all the examples are provided in Delphi Pascal, additional examples in other languages will also be provided soon.

IHS SignalMaster DLL Commands:

Type Declarations:

```
ArrayType       = Array[1..40000] of Byte;       //Byte Array
ArraySIntType   = Array[1..20000] of SmallInt;   //Small Integer Array (16 Bits)
ArrayIntType    = Array[1..10000] of Integer;    //Integer Array (32 Bits)
```

HeaderType = Array[1..8] of Byte;

Functions:

Name:	Returns:	Description:
SM_DLLVer	PANSIChar	Returns a pointer to an array of characters with the version of the dll in use.
SM_Initialize(DSPProgram:ShortString)	Integer	Initializes the SignalMaster system and loads the DSPProgram OUT file.
SM_Initialize(n:Integer)	Integer	Opens communication using COM port specified by n without loading a DSP program. This is used when working with a JTAG debugging connection and the program has already been uploaded directly from the complier and you simply wish to establish communication with your PC application.
SM_Close	Integer	Closes SignalMaster system and resets Bootloader to wait for next program upload. If you terminate an application without calling this function, you will need to turn your SignalMaster system off and on to reset the bootloader. Otherwise, you will not be able to load another application.
SM_DSPFuncTx(FuncNum,Count:Integer)	Integer	Calls a user defined or IHS predefined DSP function number FuncNum to send data to the DSP. Counter specifies the number of bytes passed as parameters for use by the function.
SM_DSPFuncRx(FuncNum:Integer)	Integer	Calls a user defined or IHS predefined DSP function number FuncNum to recieve data to the DSP. The returned integer specifies the number of bytes passed to the PC from the DSP.
SM_GetByteArray(var A:ArrayType; Count:Integer)	Integer	Download Count bytes from byte array.
SM_GetSIntArray(var A:ArraySIntType; Count:Integer)	Integer	Download Count bytes from Small Integer (16 bit) array.
SM_GetIntArray(var A:ArrayIntType; Count:Integer)	Integer	Download Count bytes from Integer (32 bit) array.
SM_PutByteArray(var A:ArrayType; Count:Integer)	Integer	Upload Count bytes from byte array.
SM_PutSIntArray(var A:ArraySIntType; Count:Integer)	Integer	Upload Count bytes from Small Integer (16 bit) array.
SM_PutIntArray(var A:ArrayIntType; Count:Integer)	Integer	Upload Count bytes from Integer (32 bit) array.

Implementation:

```
var
  hdll:THandle;
Const
```

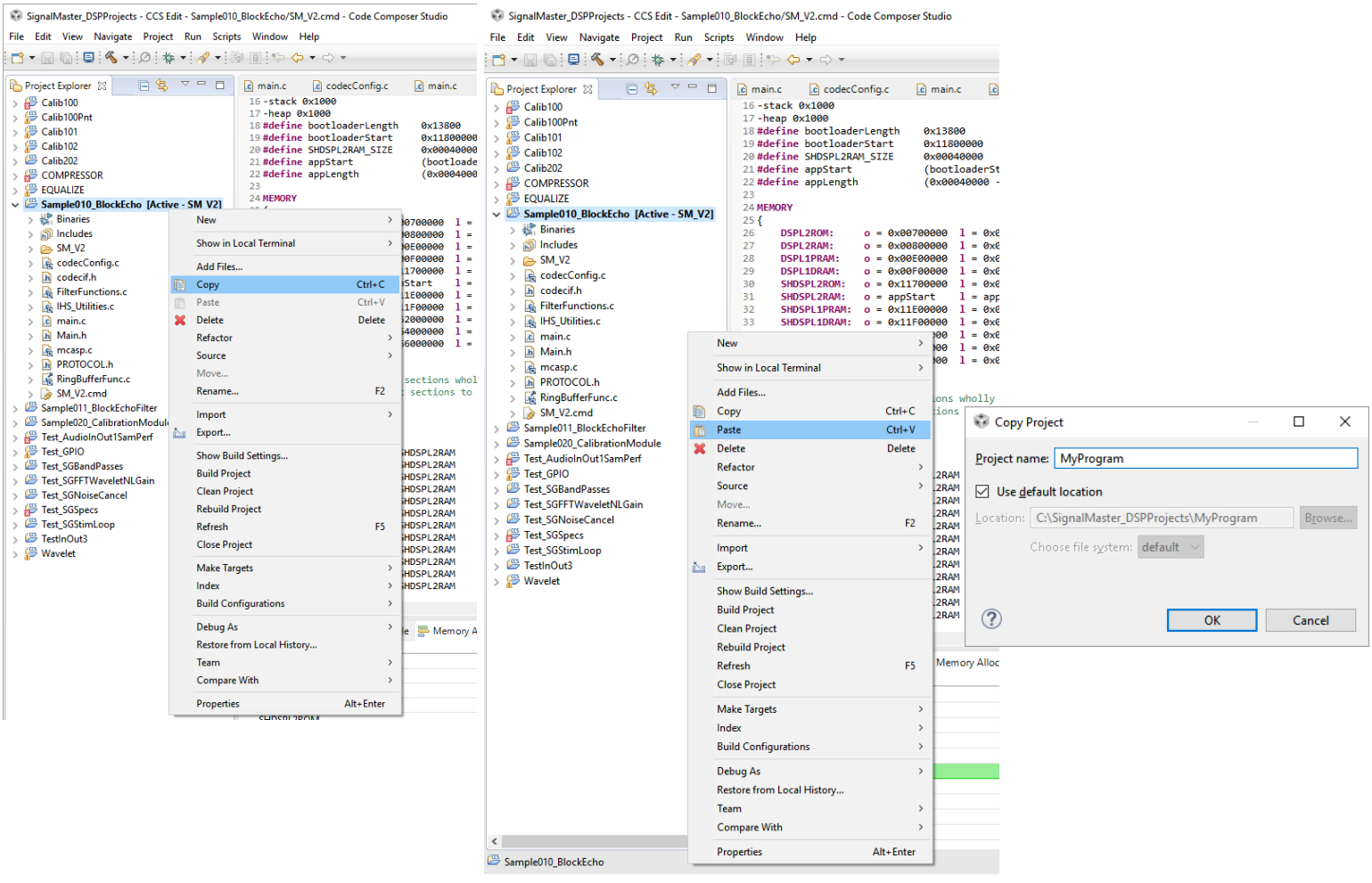
```
  SIGMASDLL = 'SIGMASDLL.dll'#0;


Type

    ArrayType        = Array[1..40000] of byte;
    ArraySIntType    = Array[1..20000] of SmallInt;
    ArrayIntType     = Array[1..10000] of Integer;
    HeaderType       = Array[1..8] of byte;

Function SM_DLLVer:PANSIChar;   cdecl;   EXTERNAL SIGMASDLL;
Function SM_Initialize(DSPProgram:ShortString):Integer; cdecl;    EXTERNAL SIGMASDLL;
Function SM_Close:Integer;   cdecl;    EXTERNAL SIGMASDLL;
Function SM_DSPFuncTx(FuncNum,Count:Integer):Integer;   cdecl;    EXTERNAL SIGMASDLL;
Function SM_GetByteArray(var A:ArrayType; Count:Integer):Integer;     cdecl;    EXTERNAL SIGMASDLL;
Function SM_GetSIntArray(var A:ArraySIntType; Count:Integer):Integer;  cdecl;    EXTERNAL SIGMASDLL;
Function SM_GetIntArray(var A:ArrayIntType; Count:Integer):Integer;    cdecl;    EXTERNAL SIGMASDLL;
Function SM_PutByteArray(var A:ArrayType; Count:Integer):Integer;      cdecl;    EXTERNAL SIGMASDLL;
Function SM_PutSIntArray(var A:ArraySIntType; Count:Integer):Integer;  cdecl;    EXTERNAL SIGMASDLL;
Function SM_PutIntArray(var A:ArrayIntType; Count:Integer):Integer;    cdecl;    EXTERNAL SIGMASDLL;
Function SM_DSPFuncRx(FuncNum:Integer):Integer; cdecl;      EXTERNAL SIGMASDLL;

var
    TxArray        : ArrayType;
    TxArraySInt    : ArraySIntType;
    TxArrayInt     : ArrayIntType;
    RxHeader       : HeaderType;
    RxArray        : ArraySIntType;
    RxArraySInt    : ArraySIntType;
    RxArrayInt     : ArrayIntType;

//You must first initialize the DLL on your application…
procedure TAudioANC.FormCreate(Sender: TObject);
begin
   Hdll:=0; //Initialize...
   Hdll:=LoadLibrary(PChar('SIGMASDLL.dll'#0));
   If Hdll=0 then ShowMessage('Error Loading IHSInterface DLL Library');
end;
```

Developing DSP Applications:

To modify the DSP code provided or develop your own applications, you will need to use the TI Code Composer Studio (CCS). Instructions are provided in the **TI Code Composer Studio (CCS) Setup** section on how to set up CCS on your computer. Please make sure you follow the CCS setup instructions very carefully. Additional information is also provided from TI documentation. IHS code examples are provided in the distribution thumb drive, in the DSPProjects subdirectory and from the SignalMaster webpage.

At this point you should have already installed CCS and selected a Work Space directory. We recommend using C:\ SignalMasterProjects. You should have also Imported the samples provided by IHS into your work space directory.

The easiest way to start a new DSP application is to look for one of the examples provided by IHS that is most similar to the application you want to develop. In CCS, simply right mouse click on the application you want to copy, and press copy from the drop down menu:



Starting a new DSP project: 1) Right mouse click on desired project and select copy, 2) Right mouse click on open area of Project Explorer (Left Panel) and select Paste, 3) Enter name of new project.

You code should be primarily limited to the main.c file. Other files and libraries contain important files that if modified could affect the function of the DSP and other project using the same file.

In the Main.C file, the DSP application has two main areas of code:

1) Initialization
2) While Loop

1) The initialization code is used to set up all the important parameters, registers and variables needed for the DSP to operate. Most of the important hardware components are initialized in the procedure: `IHS_initializeDSP`(); , but others require your attention as shown in the code example below (**Sample3010_BlockEcho**). 2) The main application While loop is where all the work is done while your application is running. The main application While is subdivided in to 2a) a region used to monitor commands from an external program and 2b) a region used for data acquisition, output and signal processing.

Therefore, your DSP application source code should focus primarily in the main While loop in the Main.C file. This loop deals with receiving function calls from the PC program and sending and receiving data based on your define function calls. After the DSP code is loaded into the system, the DSP waits in an infinite loop (see the sample source code below) for instructions from the host PC and for hardware interrupt notifications that data is ready to be processed. All DSP programs should contain code as shown in the example below:

Sample3010_BlockEcho:

```
#include "main.h"

/*****************************************************************************
PRIVATE VARIABLE DEFINITIONS
*****************************************************************************/
/* Application---------------------------------------------------------------*/
int stim                              [NUM_I2S_CHANNELS][AUDIO_SAMPLES_MAX];
int audio_rx                          [NUM_I2S_CHANNELS][AUDIO_SAMPLES_MAX];
struct FilterStorage_t filter_handle  [FILTER_NUM_MAX];

int main(void)
{
    /* Hardware Initialization-------------------------------------------------*/
    /* Setup the DSP */
    struct CommandHandle_t *command_handle_p = IHS_InitializeDSP();
    int32_t *data_array_int_ptr = (int32_t *)&(command_handle_p->data_array[0]);

    /* Setup the audio codec and its interfaces */
    double sampling_rate = 24000;
    codecSettings_t codecSettings;
    codecSettings.dmaFlag =             CODECONFIG_USE_DMA;
    codecSettings.samplingRate =        sampling_rate;  /* 24kHz, 44.1, 48, 96, 192 */
    codecSettings.wordSize =            WORD_SIZE;
    codecSettings.enableHPL =           1;      /* Enable headphone output for left  ch */
    codecSettings.enableHPR =           1;      /* Enable headphone output for right ch */
    codecSettings.enableIN1L =          1;      /* Enable mic input for left  ch */
    codecSettings.enableIN1R =          1;      /* Enable mic input for right ch */
    codecSettings.enableLOR =           1;      /* Enable aux line output for left ch */
    codecSettings.enableIN2R =          0;      /* Enable aux line input for right ch */
    codecSettings.gainMicLeft =         0;      /*  0 to 47.5 dB */
    codecSettings.gainMicRight =        0;      /*  0 to 47.5 dB */
    codecSettings.gainHeadphoneLeft =   0;      /* -6 to 29 dB */
    codecSettings.gainHeadphoneRight =  0;      /* -6 to 29 dB */
```

```
codecSettings.gainLineoutRight =    0;      /* -6 to 29 dB */
codecSettings.loopback_codec =      0;      /* The codec will route ADC data to its DAC */

/* Application Initialization----------------------------------------------*/
uint32_t i, ii;
uint32_t sample_size = 600; //Should be less than or equal to AUDIO_SAMPLES_MAX
if(sample_size > AUDIO_SAMPLES_MAX)
    IHS_ErrorSend(COMMAND_ID_ERROR_APP_SAMPLESIZE, __LINE__, __FILE__);



/* Update the transmit arrays */
IHS_Audio_WriteTxBuffers(&stim[0][0], &stim[1][0], sample_size);
/* Begin the audio loop----------------------------------------------------*/
int32_t result = IHS_InitializeCodec(&codecSettings, sample_size);
I2SDataTxRxActivate(&codecSettings);
while(1)
{
    /* Check if the PC has sent data via Bluetooth or USB-UART */
    if(IHS_CommandRxCheck() == COMMAND_RXCHECK_AVAILABLE)
    {
        switch(command_handle_p->id)
        {
            /* Send left mic to PC */
            case 201:
            i = 0;
            for(ii=0 ; ii<sample_size ; ii++)
            {
                data_array_int_ptr[ii] = (int)audio_rx[i][ii];
            }
            IHS_CommandSend(201, sample_size*sizeof(int32_t));
            break;

            /* Send right mic to PC */
            case 202:
            i = 1;
            for(ii=0 ; ii<sample_size ; ii++)
            {
                data_array_int_ptr[ii] = (int)audio_rx[i][ii];
            }
            IHS_CommandSend(202, sample_size*sizeof(int32_t));
            break;

            /* Undefined function call from PC */
            default:
            IHS_ErrorSend(COMMAND_ID_ERROR_PROCO_UNDEFINED_FUNCTION_RX, __LINE__, __FILE__);
            break;
        }
    }



    /* Check if we finished receiving a new array */
    if(IHS_Audio_RxCheck())
    {
        IHS_LED_Set(1, 1);
        IHS_Audio_UpdateBuffIndex();

        /* Read Last Buffer */
        IHS_Audio_Receive(&audio_rx[0][0], &audio_rx[1][0], sample_size);
```

```
            /* Send Next Audio Buffers */
            IHS_Audio_Send(&audio_rx[0][0], &audio_rx[1][0], sample_size);
            IHS_LED_Set(1, 0);
        }
    }
}
```

Sample DSP Programs Provided:

The following example DSP programs are provided:

Sample Name:	Description:
Sample3001_PointByPointEcho	Acquires data on a point by point basis based on the selected clock rate and outputs same point to speakers.
Sample3002_PointByPointEchoFilter	Acquires data on a point by point basis based on the selected clock rate, applies filters and outputs same point to speakers.
Sample3010_BlockEcho	Reads a block of data from the microphones and outputs the blocks to the speakers.
Sample3011_BlockEchoFilter	Reads a block of data from the microphones and outputs the blocks to the speakers after applying IIR filters. Weight constants can be added to the filter banks to provide different gain factors for each bank. This would be similar to a linear analog hearing aid.
Sample3020_CalibrationModule	Generates calibration tone and outputs through speakers. Also reads data through microphone for analysis. This application can be used with
Sample3100_Wavelet	Reads a block of data and performs an FFT. The FFT can be weighted to adjust the outputs of each bin. An iFFT is preformed and the result outputted. An overlapping window scheme is used compensate for window-to-window amplitude difference and edge effects. This program can be used with PC application – SigMasEqual.EXE in folder \PCProjects\Delphi_Pascal\SigMas_Equalizer
Sample3101_Shifter	Acquires a block of data, performs FFT and does a frequency shift of components, then performs iFFT to generate time domain output. Resulting sound output has a remapping of frequency components. This program can be used with PC application – SigMasEqual.EXE in folder \PCProjects\Delphi_Pascal\SigMas_Equalizer. You will need to modify the source code to load this OUT file instead of the default Sample100_Wavelet.out
Sample3200_WaveletCompressor	Performs compressor algorithm for HA applications with two transfer functions. Performs a 128 point FFT, selects the appropriate transfer function based on the sound level, attack and release timing of each frequency band, and perform an iFFT in less than 5.33 msec for both ears. This program can be used with PC application – SigMasComp.EXE in folder \PCProjects\Delphi_Pascal \SigMas_Compressor

Debugging DSP Programs:

The easiest method to debug your DSP applications is to develop your function calls one at a time and make sure they return specific values that you can check to make sure they are functioning as expected. You can also turn the LEDs on your device on and off in order to monitor the progress of your application as the execution move from one routine to another.

To turn ON an LED call:

```
IHS_LED_Set(n,1);  // Where n=LED number (1-4)
```

To turn ON ALL LEDs call:

```
IHS_LED_Set (0,1);  // Turns on ALL LEDs (1-4)
```

To turn OFF an LED call:

```
IHS_LED_Set (n,0); // Where n=LED number (1-4)
```

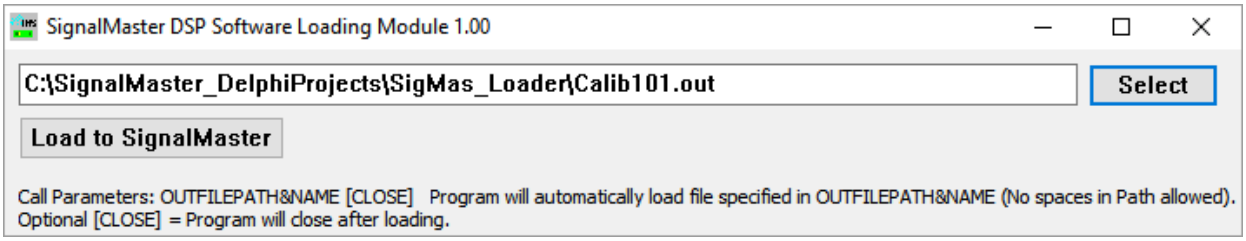
To turn OFF ALL LEDs call:

```
IHS_LED_Set (0,0);  // Turns OFF ALL LEDs (1-4)
```

Note: Although CCS provides the ability to do live debugging of code during the development process from the compiler, this feature is not directly available in the enclosed SignalMaster product ver 2.00.

Loading Programs to the DSP:

After you have developed your own DSP applications, you can load your programs on to the DSP using the SigMasLoader.EXE program. The source code for this program is provided in the PCProjects\Delphi_Pascal\SigMas_Loader subdirectory. The program uses the DLL to call the SM_Initialize routine to load the corresponding DSP compiled program (OUT File). When you run this program, it will ask you to select the DSP program you want to run.



You can also call this program from another application using call parameters. The first parameter should be the path and name of the DSP compiled program you wish to load. For example, if you wish to run the program, MyDSPProgram.OUT, you can call: “SigMasLoader.EXE C:\MyDirectory\MyDSPProgram.OUT”. This will automatically load that program. If you also want to close SigMasLoader after loading the program, simply add [CLOSE] as a second parameter: “SigMasLoader.EXE C:\MyDirectory\MyDSPProgram.OUT [CLOSE]”. When passing parameters to another application, remember not to use spaces in the file path or file name. Spaces indicate a new parameter and will result in the loading program not being able to find you DSP program.

SignalMaster Ver 2.00 Specifications:

Processing Family: Texas Instrument TMS320C6746 32/64 bit floating point processor**.**

For complete datasheet, visit:
http://www.ihsys.com/ohsspds/signalmaster/tms320c6746.pdf

Processing Speeds: The TMS320C6746 is capable of performing 2100 million floating point operations per second (MFLOPS) with a 2.8 ns cycle time

Codec: TLV320AIC3254 Ultra Low Power Stereo Audio Codec with imbedded miniDSP. For additional information, visit:
http://www.ihsys.com/ohsspds/Documents/slaa408a.pdf

Memory: 256KB (Kilo Bytes) of internal memory and addressing lines providing access to 4MB (Mega Bytes) of external asynchronous memory.

Data Sampling Rate: Programmable with rates from 8 to 96kHz.

PC Communication: USB & Ethernet

Analog Input: Microphone inputs with stereo drivers & A/Ds (programmable 16/20/24/32 bits)

Analog Output: Stereo headphones outputs with drivers & D/As (Programmable 16/20/24/32 24 bits)

Universal Asynchronous Receiver/Transmitter (UART) communication modules: The system provides a USB/Serial communication and EtherNet UARTs for data exchange with a PC for programming, parameter selection and data exchange.

LEDs: Battery power indicators and programmable logic indicators.

Power Supply: Options to run off battery or A/C power adaptor.

Battery: ICR18650 Li-Ion 2 X 3.7 V 2600 mAh (10 hours) (1.67 oz each)

Size: 190 mm X 90 mm X 30 mm

Weight: <8 oz

Photo of SignalMaster device showing dual microphones and stereo headphones connected to the system. The system can use any standard 8-12 ohm headphones.

SignalMaster™ Board

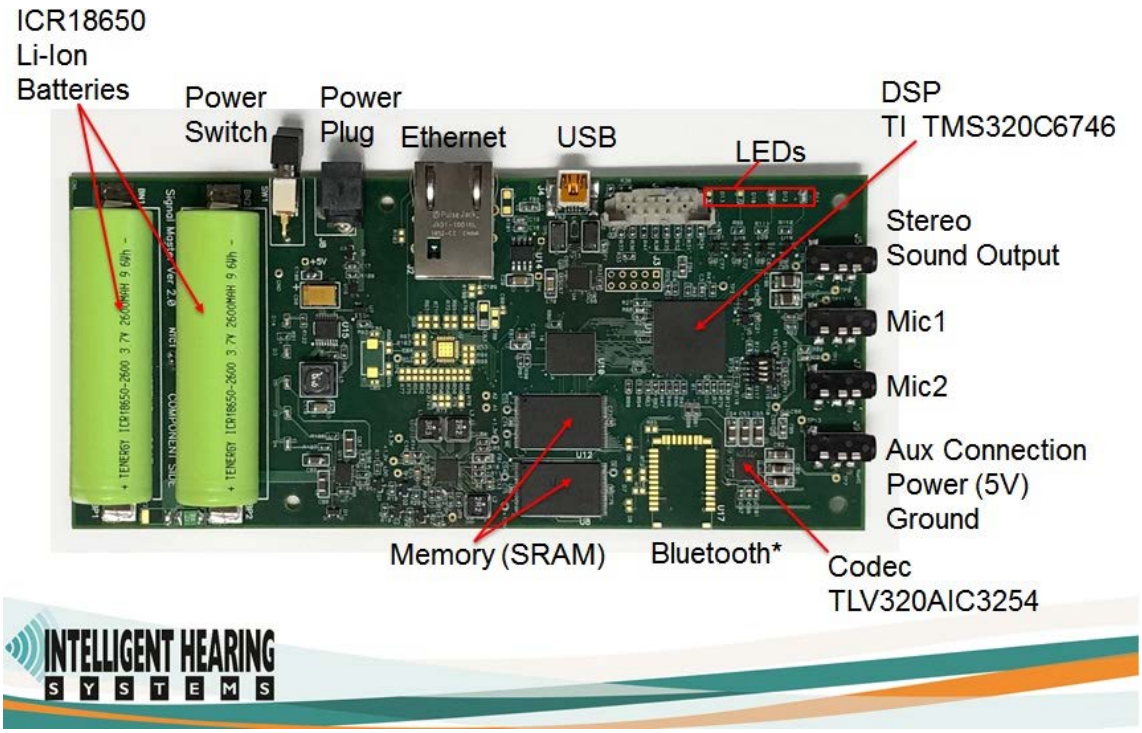


Photo of inside of SignalMaster enclosure showing DSP circuit board, battery charging circuit board, power switch, and power, microphone, headphone connectors.

USE EXTREME CAUTION WHEN OPENING THE SignalMaster ENCLOSURE
Sensitive electronic components can be damaged by electrostatic discharge.
Do not mishandle or damage Lithium-Ion batteries.
Do not ship system if batteries are damaged.
Follow all Lithium-Ion battery shipment regulations.

Fire Hazard if batteries shorted, mishandled or damaged!

Technical Support:

For any questions or technical support with your SignalMaster system, email: support@ihsys.com
Please make sure to enter “SignalMaster Support” in the subject line and the serial number of your system.