Ver. November 11, 2022

OMAP-L138 Experimenter Kit and CCS6: First example

1 Objective

CCS6 gives the possibility to download code compiled for the DSP C6748 onto the OMAP-L138 Experimenter Kit. In this guide we will compile and download a simple example onto the board, as the objective of the laboratory is to practice and verify the basics of the Code Composer Studio Integrated Design Environment. This laboratory has the following goals:

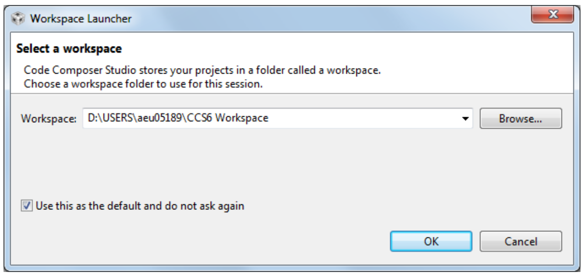
- Create and build a new project
- Compile a program for the C6748 DSP
- Load the program onto the target
- Examine variables, memory and code
- Run, halt, step, multi-step, use breakpoints
- Graph results in memory

2 Sine generation algorithm

There are many ways to create sine values. Here we have chosen a simple model based upon a monostable IIR filter. It is implemented in the code:

```
/* A Generates a value for each output sample */
float y[3] = {0, 0. 0654031, 0};
float A = 1. 9957178;
short sineGen() {
y[0] = y[1] * A - y[2];
y[2] = y[1];
y[1] = y[0];
return((short)(32000*y[0]);
}
```

The block sine wave generator function generates the next sine value in the sequence each time it is called.


Workspace Launcher

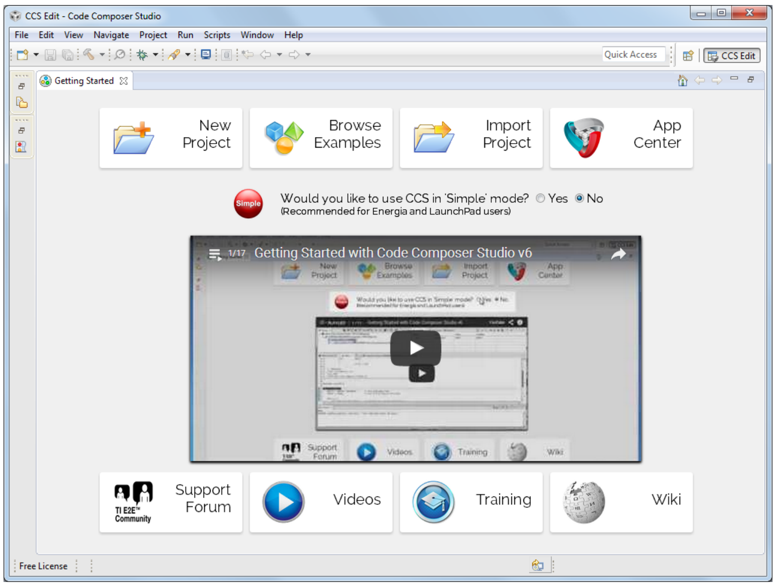
Select a workspace

Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\USERS\aeu05189\CCS6 Workspace

[✓] Use this as the default and do not ask again

Figure 1: Selecting the workspace.


CCS Edit - Code Composer Studio

Would you like to use CCS in 'Simple' mode? Yes [✓ No]
(Recommended for Energia and LaunchPad users)

Figure 2: CCS4 opening window.

3 Procedure

The following procedure will help you build your first project for the TMS320C6748 DSP under CCS6.

3.1 Open Files, Create Project File

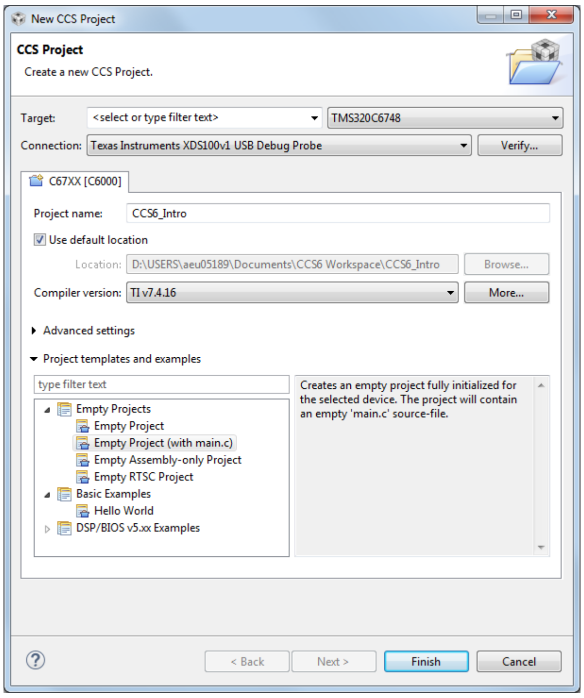
Launch CCS6 and select a workspace as in Figure 1. The window in Figure 2 will pop up (every time a new workspace is created). Close the "Getting Started" tab.

Create a new project "CCS6_Intro", by selecting "File" > "New" > "CCS Project". The window in Figure 3 will pop-up. In this window you will need to set the following settings:

- Target: TMS320C6748
- Connection: Texas Instruments XDS100v2 USB Debug Probe
- Compiler version: TI v7.4.16
- Project templates and examples: Empty Project (with main.c)

Under "Advanced settings", set the following settings:

- Output type: Executable
- Output format: legacy COFF


New CCS Project

CCS Project

Create a new CCS Project.

Target: <select or type filter text> TMS320C6748

Connection: Texas Instruments XDS100v1 USB Debug Probe

C67XX [C6000]

Project name: CCS6_Intro

☑ Use default location

Location: D:\USERS\aeu05189\Documents\CCS6 Workspace\CCS6_Intro

Compiler version: TI v7.4.16

Advanced settings

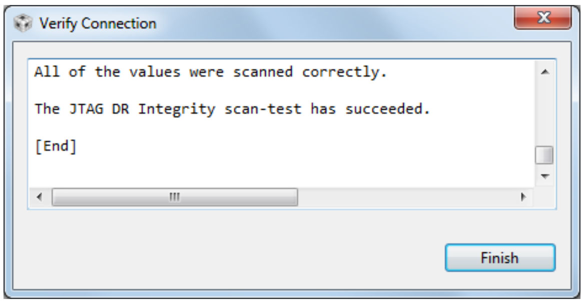
Project templates and examples

type filter text

- Empty Projects
  - Empty Project
  - Empty Project (with main.c)
  - Empty Assembly-only Project
  - Empty RTSC Project
- Basic Examples
  - Hello World
- DSP/BIOS v5.xx Examples

Creates an empty project fully initialized for the selected device. The project will contain an empty 'main.c' source-file.

Figure 3: Creating a new project.


Verify Connection

```
All of the values were scanned correctly.

The JTAG DR Integrity scan-test has succeeded.

[End]
```

Figure 4: Verify connection.

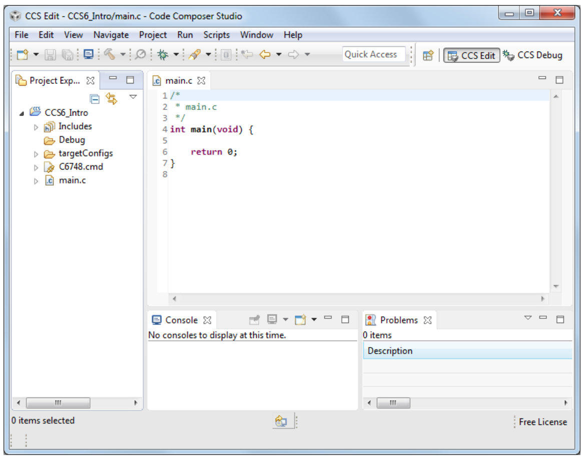
- Device endianness: little
- Linker command file: <none>
- Runtime support library: <automatic>

In this window you can also verify the connection with the board. To do this, once the OMAP-L138 board is connected and powered, click on the "Verify..." button. A small pop-up window should appear; and if the connection is successful, the window should show a message as illustrated in Figure 4. If the test is not successful, try change the Connection type to XDS100v2.

Finally, give a name to the project ("CCS6_Intro") and click "Finish". The new project will be created, as illustrated in Figure 5.

3.2 Create C files

The current project already contains a C file called "main.c". Right-click on the file in the Project Explorer, and select "Delete". In the "Delete Resources" pop-up window that appears, click "OK".



```
/*
 * main.c
 */
int main(void) {

    return 0;
}
```

No consoles to display at this time.

0 items

0 items selected

Figure 5: New project created.

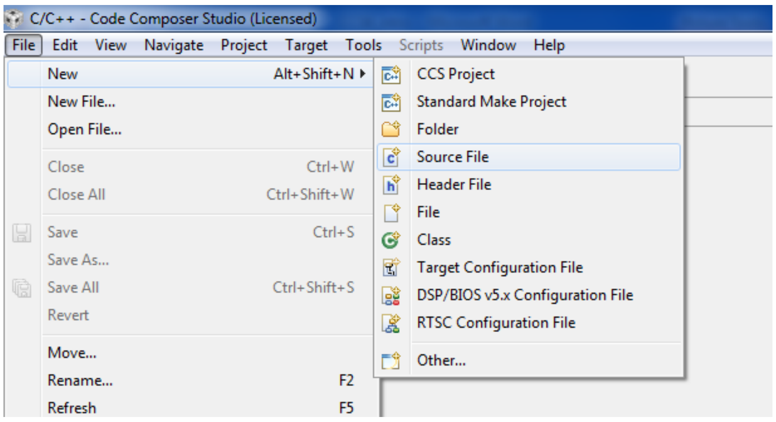
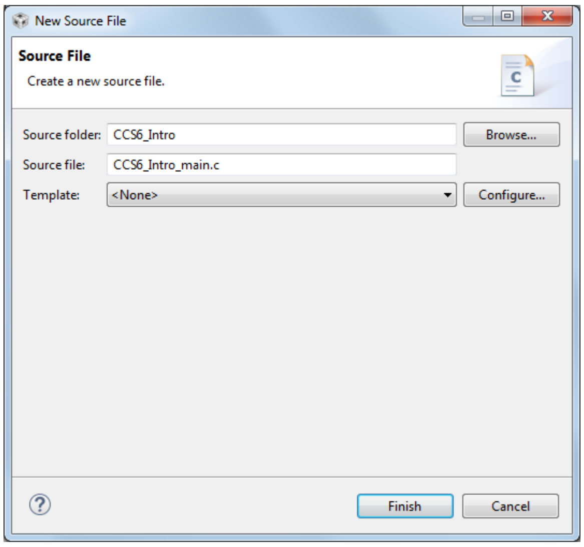


Figure 6: Creating a new source file (1).



Source File

Create a new source file.

Source folder: CCS6_Intro

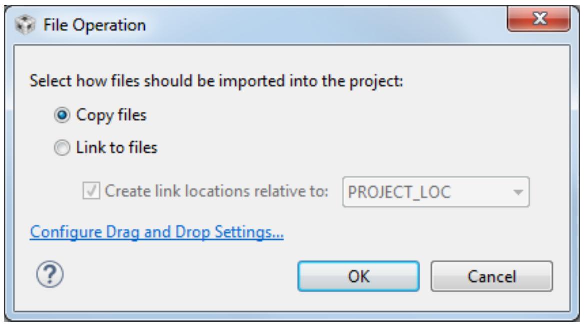
Source file: CCS6_Intro_main.c

Template: <None>

Figure 7: Creating a new source file (2).

Create a new source code file by clicking “File” > “New” > “Source File”, as in Figure 6. A new window will open (Figure 7). Browse and select “CCS6_Intro” as “Source Folder”, enter the file name “CCS6_Intro_main.c”, select “<None>” as “Template”, an click “Finish”. The new file will appear in the project window.

In the file “CCS6_Intro_main.c” enter the code shown below:


File Operation

Select how files should be imported into the project:

[✓ Copy files]

Link to files

Create link locations relative to: PROJECT_LOC

Configure Drag and Drop Settings...

Figure 8: Copy source code files.

```
// Declarations
#define BUFFSIZE 128
void blockSine(short *buf, int len);

// Global Variables
static short gBuffer[BUFFSIZE];

// ======== main ========
// Simple function which calls blockSine
void main()
{
    blockSine(gBuffer, BUFFSIZE); // Fill buffer with sine data
    return;
}
```

Select “File” > “Save” to save.
To add existing files to the project, right-click on the project name and select “Add Files...”. In the new pop-up window, browse and select the provided source code file “block_sine.c”. In the next “File Operation” window, select “Copy files” and click “OK” (Figure 8).

The code of the source file “block_sine.c” is shown below:

```
// ======== block_sine.c =================================
// The coefficient A and the three initial values
// generate a 500Hz tone (sine wave) when running
// at a sample rate of 48KHz.
//
// Even though the calculations are done in floating
// point, this function returns a short value since
// this is what's needed by a 16-bit codec (DAC).

// ======== Prototypes ===================================
void blockSine(short *buf, int len);
short sineGen(void);

// ======== Definitions ==================================
// Initial values
#define Y1 0.0654031    // = sin((f_tone/f_samp) * 360)
                        // = sin((500Hz / 48KHz) * 360)
                        // = sin (3.75)
#define AA 1.9957178    // = 2 * cos(3.75)

// ======== Globals =====================================
static float y[3] = {0,Y1,0};
static float A = AA;

// ======== sineGen ======================================
// Generate a single element of sine data
short sineGen(void)
{
    y[0] = y[1] * A - y[2];
    y[2] = y[1];
    y[1] = y[0];

    // To scale full 16-bit range we would multiply y[0]
    // by 32768 using a number slightly less than this
    // (such as 32000) helps to prevent overflow.
    y[0] *= 32000;
    // We recast the result to a short value upon returning it
    // since the D/A converter is programmed to accept 16-bit
    // signed values.
    return((short)y[0]);
}

// ======== blockSine ========
// Generate a block of sine data using sineGen
void blockSine(short *buf, int len)
{
    int i = 0;
    for (i = 0;i < len; i++) {
        buf[i] = sineGen();
    }
}
```

3.3 CCS Edit perspective

Please note that CCS may be set up for a "different perspective". Look up in the right-hand corner to see the two default perspectives (Figure 9). Note that "CCS Edit" is highlighted. We are editing files and working with projects. The windows and views you see in this perspective are "default" and can be changed to your liking later.

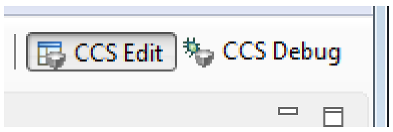


Figure 9: Selecting the "CCS Edit" perspective.

3.4 Check the includes directory

So, you have just created a CCS6 project (Figure 10). Click on the arrow next to "Includes". Notice that CCS has "automatically" included all of the compiler header files. If you click on the arrow next to this directory of header files, you will see the standard TI compiler header files.

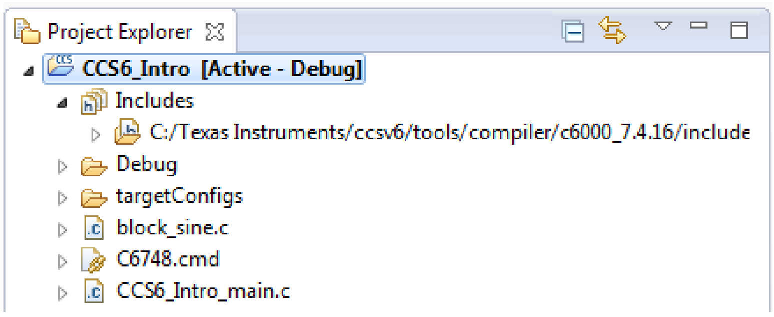

Project Explorer
CCS6_Intro [Active - Debug]
Includes
C:/Texas Instruments/ccsv6/tools/compiler/c6000_7.4.16/include
Debug
targetConfigs
block_sine.c
C6748.cmd
CCS6_Intro_main.c

Figure 10: Project structure.

3.5 View files created in your directory

What you see in the project view in CCS (for the most part) directly reflects what Windows Explorer view will look like. When a project is created, CCS will create a set of files that contain your project settings. These are the "bare bones" files you need to configure your project. Together these files form an "Eclipse" project. As we add files to our project, we will watch the Windows Explorer view reflect our actions.

3.6 Cmd file

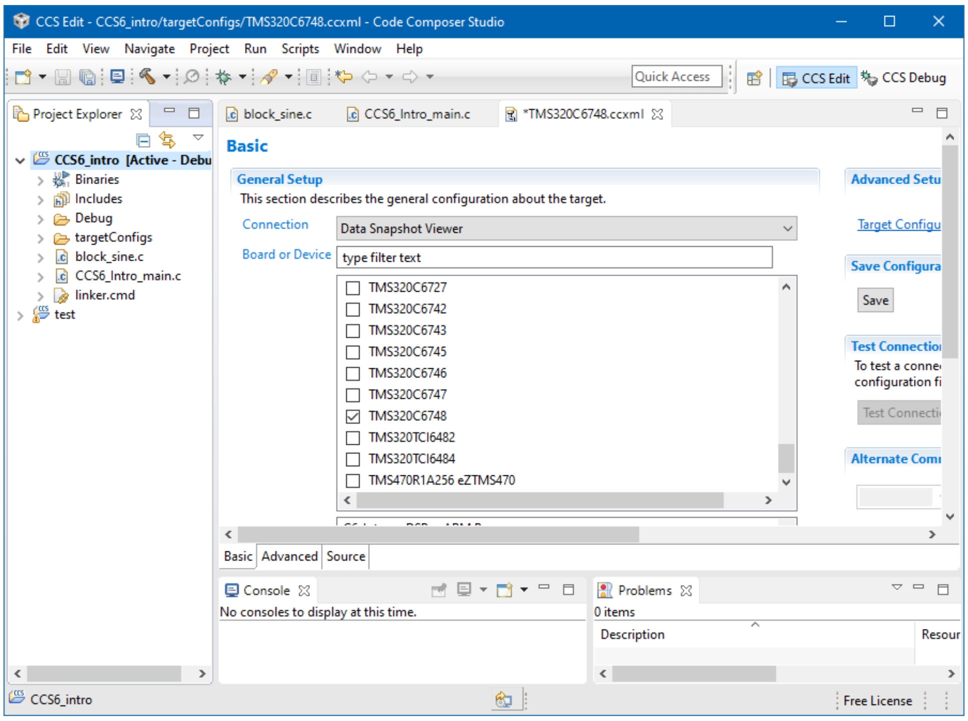
Create a new cmd file by selecting "File" > "New" > "Source File"; give a name with ".cmd" extension, as for example "linker.cmd", and make sure to select "<None>" as "Template". In the new source file, paste the following code:

```
/***************************************************************************
* linker.cmd for BIOS Workshop LAB 2A
***************************************************************************/

-stack 0x00010000
-heap 0x00000800

MEMORY
{
    IRAM:             ORIGIN = 0x11800000 LENGTH = 0x00040000
    L3_shared_RAM:    ORIGIN = 0x80000000 LENGTH = 0x00020000
    DDR2_data:        ORIGIN = 0xC0000000 LENGTH = 0x08000000
}
SECTIONS
{
    .text > IRAM
    .const > IRAM
    .bss > IRAM
    .far > IRAM
    .switch > IRAM
    .stack > IRAM
    .data > IRAM
    .cinit > IRAM
    .sysmem > IRAM
    .cio > IRAM
}
```

Save the cmd file in your project folder. The cmd file will be automatically added to your project. The cmd file is required to set up the program code in the board memory.



General Setup

This section describes the general configuration about the target.

Connection: Data Snapshot Viewer

Board or Device: type filter text

- [ ] TMS320C6727
- [ ] TMS320C6742
- [ ] TMS320C6743
- [ ] TMS320C6745
- [ ] TMS320C6746
- [ ] TMS320C6747
- [x] TMS320C6748
- [ ] TMS320TCI6482
- [ ] TMS320TCI6484
- [ ] TMS470R1A256 eZTMS470

Figure 11: Target configuration setup.

3.7 Build configuration

Notice the word "[Active - Debug]" next to your project name: "Active" means that this project is the "Active project"; "Debug" means that the Debug build configuration is being used - i.e. no optimizations are turned on. Right-click on the project and select "Build Configuration" > "Set Active" > "Release". Now you can see that "Debug" has changed to "Release" in the project window. This is how you change build configurations (set of build options) when you build your code. Build Configurations not only contain standard build options like levels of optimization and debug symbols, but also contain specific "file search paths" for libraries (-l) and "include search paths" (-i) for include directories. If you specify these paths in a Debug configuration and you switch to Release, those paths do NOT copy over to the next configuration. This is because, in general, you have a "debug" or "instrumented" library you are using during initial debug, and a completely different "optimized" library when attempting to optimize your code. Now, you can change the build configuration back to "Debug".

3.8 Target configuration for the project

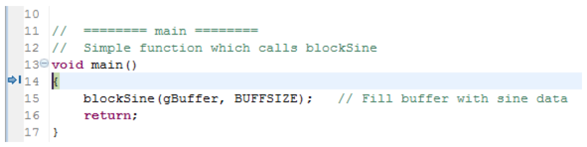
To specify which device to use, CCS6 uses target configuration files. An appropriate target configuration file has already been created automatically when the project was initially set up. The target configuration file "TMS320C6748.ccxml" can be found in the folder "targetConfigs" of the project (Figure 11).

3.9 Load code into target

In order to run the code, you will need to:

1. Compile the project;

Figure 12: Debugger icon.



```
//  ======== main ========
//  Simple function which calls blockSine
void main()
{
    blockSine(gBuffer, BUFFSIZE);   // Fill buffer with sine data
    return;
}
```

Figure 13: Program ready to execute.

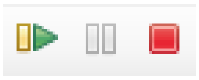


Figure 14: Debugger action button.

2. Run the debugger;

3. Connect to the target;

4. Load the program.

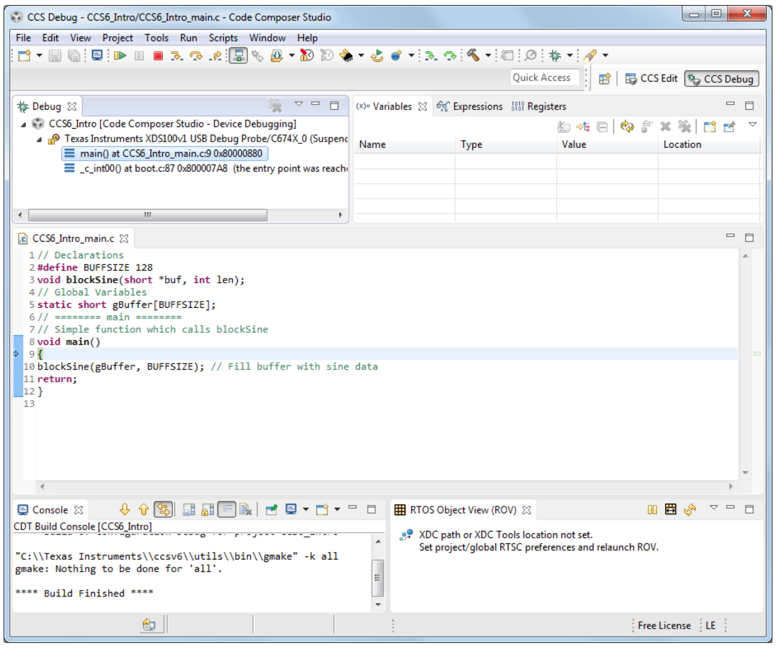
There is a handy way to accomplish all four with one selection. In the window tool bar, you will see the button shown in Figure 12.

You can just click the “Bug” itself. The “Debug” perspective will open. A blue arrow should now point to the line after “void main()”, in file “CCS6_Intro_main.c” (Figure 13). This is an indication that the machine code has been downloaded properly into the target (in this case the C6748 DSP). The processor is “at main()” and ready to run. Near the top of the screen, locate the action buttons (Figure 14).

- “Resume” (green) is to “run” the program;
- “Suspend” (yellow) is to “halt” the program;
- “Terminate” (red) is to stop your debug session (Figure 15). The next time you build code, you will need to re-launch the TI Debugger and re-connect to the target.

3.10 Watch variables

Select and highlight the variable “gBuffer” in the “CCS6_Intro_main.c” window. Right-click on “gBuffer”, choose “Add Watch Expression...” and click “OK” (Figure 16). The “Expressions” window will appear (Figure 17). Adding a variable to the “Expressions” window opens it automatically. Click on the arrow sign next to “gBuffer” to see the individual elements of the array. At some point, if the “Epressions” window shows an error “unknown identifier” for a variable, it is due to the scope of the variable. Local variables do not exist (and thus, they do not have a value) until their function is called. If requested, Code Composer will add local variables to the “Expressions” window, but will indicate they are not valid until the appropriate function is reached.



CCS6_Intro [Code Composer Studio - Device Debugging]
Texas Instruments XDS100v1 USB Debug Probe/C674X_0 (Suspenc
main() at CCS6_Intro_main.c:9 0x80000880
_c_int00() at boot.c:87 0x800007A8 (the entry point was reach

```
// Declarations
#define BUFFSIZE 128
void blockSine(short *buf, int len);
// Global Variables
static short gBuffer[BUFFSIZE];
// ======== main ========
// Simple function which calls blockSine
void main()
{
blockSine(gBuffer, BUFFSIZE); // Fill buffer with sine data
return;
}
```

CDT Build Console [CCS6_Intro]

```
"C:\\Texas Instruments\\ccsv6\\utils\\bin\\gmake" -k all
gmake: Nothing to be done for 'all'.

**** Build Finished ****
```

XDC path or XDC Tools location not set.
Set project/global RTSC preferences and relaunch ROV.

Figure 15: Debug perspective.

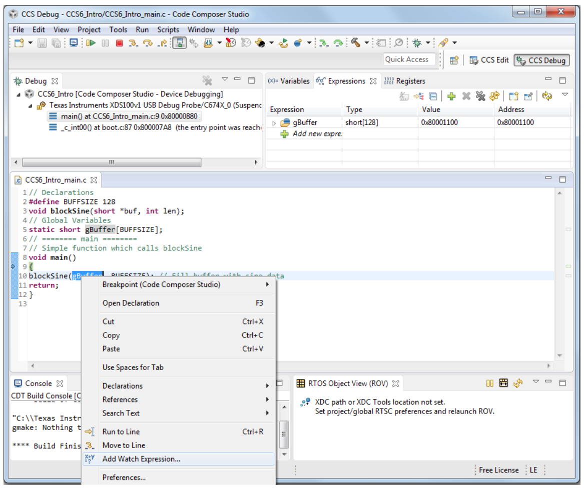
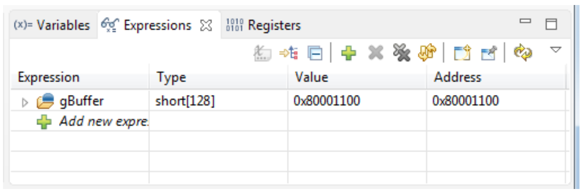


Figure 16: Adding the “Expressions” Window.

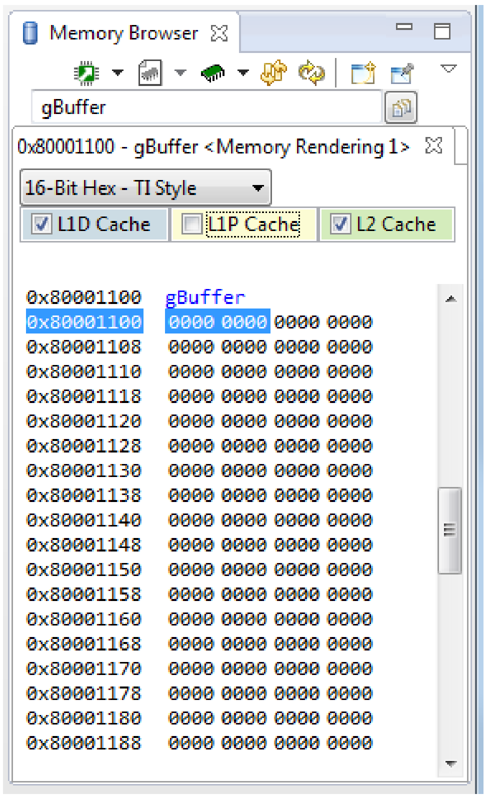


Expression	Type	Value	Address
gBuffer	short[128]	0x80001100	0x80001100
Add new expre			

Figure 17: The “Expressions” Window.

3.11 Viewing memory

Another way to view values in memory is to use a “Memory” window. Select: “View” > “Memory Browser”. Next, type “gBuffer” as the location and select the format as “16-Bit UnSigned Int”, as shown in Figure 18. You may need to resize the windows so that you can see your code and the buffer. Because we have just come out of reset and this memory area was not initialized, you might see random values.



```
0x80001100   gBuffer
0x80001100   0000 0000 0000 0000
0x80001108   0000 0000 0000 0000
0x80001110   0000 0000 0000 0000
0x80001118   0000 0000 0000 0000
0x80001120   0000 0000 0000 0000
0x80001128   0000 0000 0000 0000
0x80001130   0000 0000 0000 0000
0x80001138   0000 0000 0000 0000
0x80001140   0000 0000 0000 0000
0x80001148   0000 0000 0000 0000
0x80001150   0000 0000 0000 0000
0x80001158   0000 0000 0000 0000
0x80001160   0000 0000 0000 0000
0x80001168   0000 0000 0000 0000
0x80001170   0000 0000 0000 0000
0x80001178   0000 0000 0000 0000
0x80001180   0000 0000 0000 0000
0x80001188   0000 0000 0000 0000
```

Figure 18: Memory view.

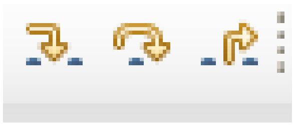


Figure 19: Single-stepping options.

3.12 Single-stepping code

Use the toolbar to single-step the debugger (Figure 19) until you reach the "blockSine()" function; it contains local variables. Step-over the "blockSine()" function, until the blue arrow is positioned after the "return;" line. The values of "gBuffer" should be the same in both the "Memory Browser" and the "Expressions" tab.

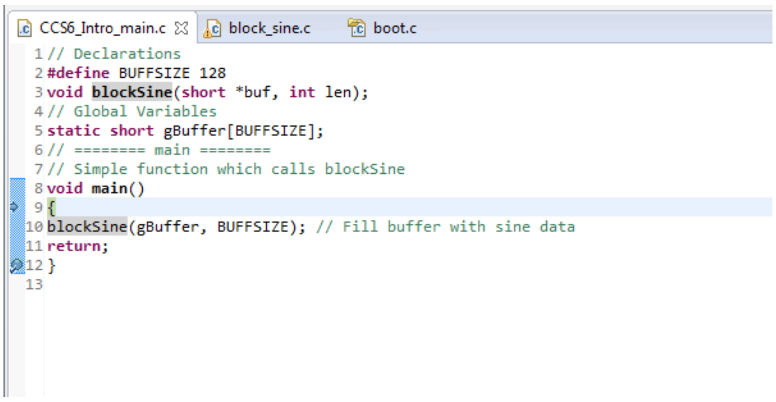
Now restart the program by selecting "Run" > "Restart".

3.13 Setting breakpoints

To set a break point, place the cursor at the end of main() and double-click the line number (Figure 20).

3.14 Running code

Run your code up to the breakpoint, by clicking the "Resume" button. The processor will halt at the breakpoint that you set previously. Notice that the "Expressions" window changes to show the new values of "gBuffer".



```
// Declarations
#define BUFFSIZE 128
void blockSine(short *buf, int len);
// Global Variables
static short gBuffer[BUFFSIZE];
// ======== main ========
// Simple function which calls blockSine
void main()
{
blockSine(gBuffer, BUFFSIZE); // Fill buffer with sine data
return;
}
```

Figure 20: Setting a breakpoint.

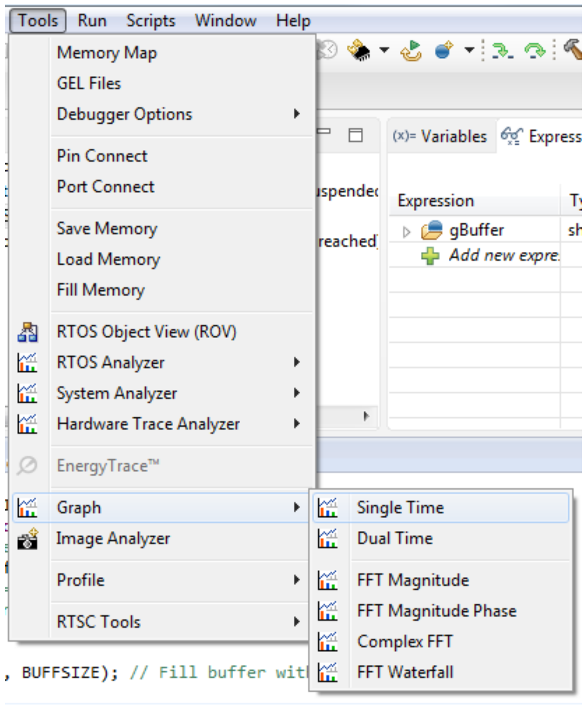


Figure 21: Selecting the time domain display.

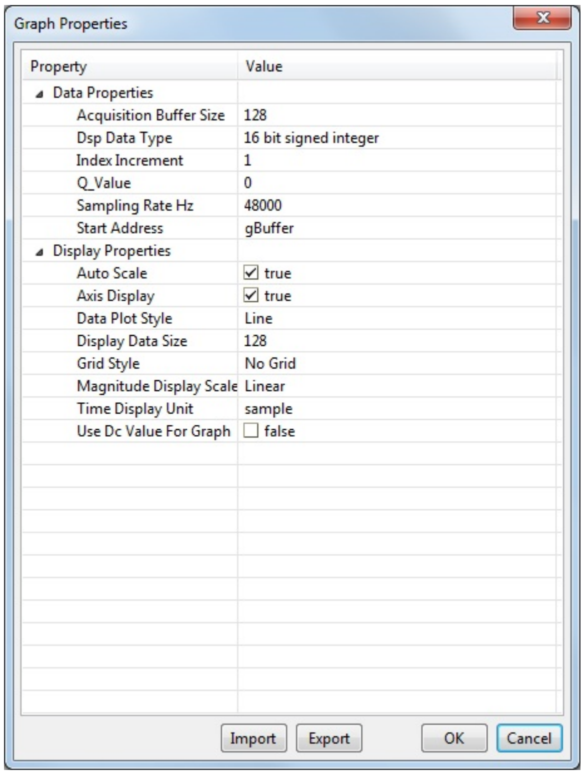
3.15 Graphing data

Your data can also be displayed as a graph both in the time and frequency domains. To display the sine data in the time domain, select “Tools” > “Graph” > “Single Time” (Figure 21). Modify the following values:

- Start Address: gBuffer
- Acquisition Buffer Size: 128
- Display Data Size: 128
- DSP Data Type: 16-bit signed integer
- Sampling Rate: 48000 Hz

Click “OK” when finished. Your graph should look like the one in Figure 23.

To display the sine data in the frequency domain, select “Tools” > “Graph > “FFT Magnitude”, and modify the following values:



Graph Properties

Property	Value
◢ Data Properties	
Acquisition Buffer Size	128
Dsp Data Type	16 bit signed integer
Index Increment	1
Q_Value	0
Sampling Rate Hz	48000
Start Address	gBuffer
◢ Display Properties	
Auto Scale	✓ true
Axis Display	✓ true
Data Plot Style	Line
Display Data Size	128
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	☐ false

Figure 22: Time domain display parameters.

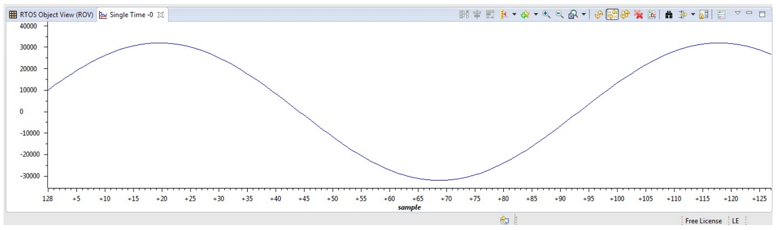


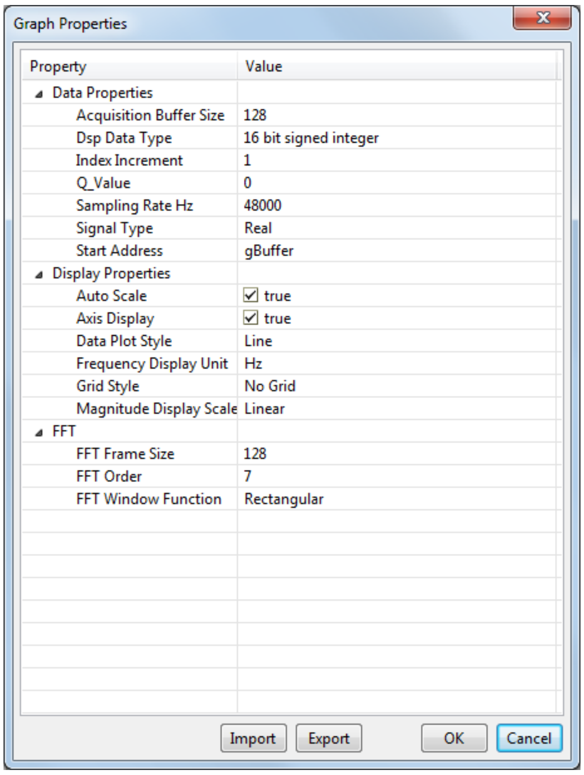
Figure 23: Time domain display.

- Start Address: gBuffer
- Acquisition Buffer Size: 128
- DSP Data Type: 16-bit signed integer
- Sample Rate: 48000 Hz
- FFT Frame Size: 128
- FFT Order: 7

Click "OK" when finished. You can now see the frequency spectrum of the wave (Figure 25).
To terminate, click the red "Terminate" button.

4 Additional documentation

Further information can be found at:



Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	128
Dsp Data Type	16 bit signed integer
Index Increment	1
Q_Value	0
Sampling Rate Hz	48000
Signal Type	Real
Start Address	gBuffer
Display Properties	
Auto Scale	✓ true
Axis Display	✓ true
Data Plot Style	Line
Frequency Display Unit	Hz
Grid Style	No Grid
Magnitude Display Scale	Linear
FFT	
FFT Frame Size	128
FFT Order	7
FFT Window Function	Rectangular

Figure 24: Frequency domain display parameters.

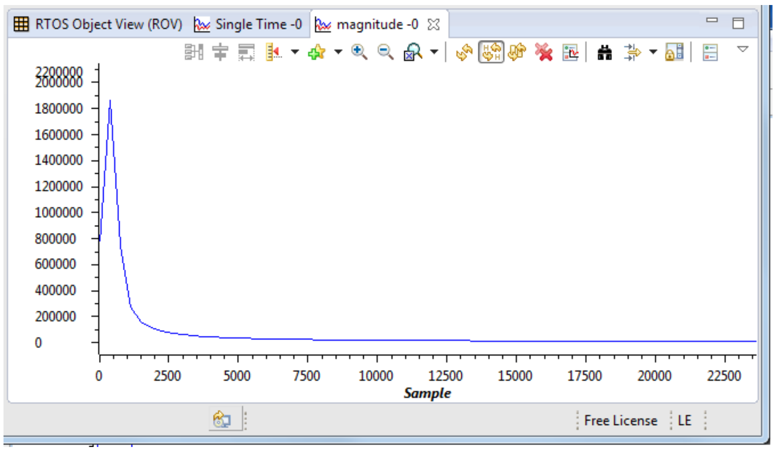


Figure 25: Frequency domain display.

- "*Digital Signal Processing and Applications with the OMAP- L138 eXperimenter*", by Donald Reay, March 2011, Wiley Publishing, ISBN: 978-0-470-93686-3.
- "*Zoom OMAP-L138 Experimenter Kit – User Guide*", from Logic PD: http://www.logicpd.com.