<Ex14>: Fixed-Point Implementation

Objective: Write a program to implement Fixed-Point Implementation using EPB_C5515 target board

Workflows you learned in the **previous lab**

- Connecting your DSP kit EPB_C5515 to CCS5.3
- Creating a new project or copying an existing project into the workspace
- Configuring the linker options and file-search paths
- Building/Compiling and running/Executing a project on the kit EPB_C5515
- Making use of breakpoints for debugging the code and using watch window to track variable values.
- Storing the data read from PC to the buffer in the CCS5.3 and view buffer data runtime.
- Generating new sine wave signal from lookup table
- Re-generating sine wave using loop-back method
- Controlling Gain of encoder for AIC3204 audio codec
- Profiling for FIR filter using linear buffering by C language and assembly language
- Profiling for FIR filter using Circular buffering by C language and assembly language

After reading **this section** you will be able to,

- Understand Fixed-Point Implementation using EPB_C5515 kit.

Part List:

- PC
- Code Composer Studio
- +5v DC Power supply
- EPB_C5515
- Emulator + Emulator cable (USB A to Mini-A Cable, 14 pin FRC Flat cable)

List of Files Required:

• fixed_float.c	(Program application main.c file)
• lnkx.cmd	(Command file)
• usbstk5515bsl.lib	(Library file)

Lab Goals:

- Understand Fixed-point representation of numbers and Fixed-point computations
- Compare Fixed-point and Floating-point implementations of an FIR Filter in terms of cycle count and accuracy

1. Introduction:

TMS320C5515 is a Fixed-point processor. Fixed-point arithmetic is generally used when hardware resources are limited and we can afford a reduction in accuracy in return for higher execution speed. Fixed-point processors are either 16-bit or 24-bit devices, while floating point processors are usually 32-bit devices. A typical 16-bit processor such as the TMS320C55x, stores data as a 16-bit integer or a fraction format in a Fixed range. Although signals are only stored with 16-bit precision, intermediate values during the arithmetic operations may be kept at 32-bit precision. This is done using the internal 40-bit accumulator, which reduces cumulative
round-o_ errors. Fixed-point DSP devices are usually cheaper and faster than their Floating-point counterparts because they use less silicon, have lower power consumption, and require fewer external pins. Most high volume low cost embedded applications, such as appliance control, cellular phones, hard disk drives, modems, audio players, and digital cameras use Fixed-point processors.

Floating-point arithmetic greatly expands the dynamic range of numbers. A typical 32-bit Floating point DSP processor, such as the TMS320C67x, represents number with a 24-bit mantissa and an 8-bit exponent. The mantissa represents a fraction in the range -1.0 to +1.0, while the exponent is an integer that represents the number of places that the binary point must be shifted left or right in order to obtain the true value. For example, in decimal number system we have some number like 10.2345. We can write this in the form of 0:102345 _ 102. So in this format, 0.102345 is called mantissa and 2 is called exponent.

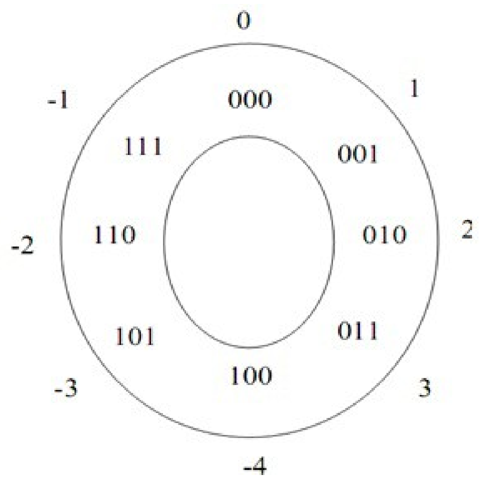
A 32-bit Floating-point format covers a large dynamic range, thus the data dynamic range restriction may be virtually ignored in a design using Floating-point DSP processors. But in Fixed-point format, the designer has to apply scaling factors and other techniques to prevent arithmetic overflow. This is usually a difficult and time consuming process. As a result, Floating-point DSP processors are generally easy to program and use, but are more expensive and have higher power consumption. In this session, you will learn the issues related to Fixed-point arithmetic

Two's complement Integer Representation

This representation is most common method for representing a signed integer. The two's complement of a binary number is defined as the value obtained by subtracting the number from a large power of two (specifically, from 2N for an N-bit two's complement). As far as the hardware is concerned, Fixed-point number systems represent data as B-bit integers.
The two's complement number system usually used is

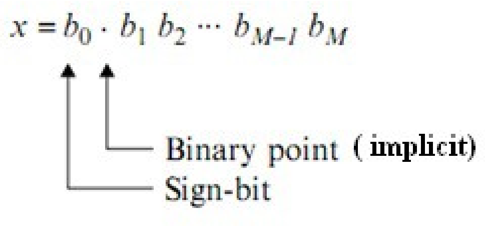
$$k = \begin{cases} binary\ integer\ representation, & \text{if } 0 \le k \le 2^{B-1} - 1 \\ bitwise\ complement\ of\ k+1, & if - (2^{B-1}) \le k \le 0 \end{cases}$$

The most significant bit is known as the sign bit. It is 0 when the number is non-negative and 1 when the number is negative.
Below Figure 1 is an easy way to visualize two's complement representation.



Fractional Fixed-point number representation

For the purposes of signal processing, we often regard the Fixed-point numbers as binary fractions in the range [-1, 1), by implicitly placing a decimal point after the sign bit. Fixed-point representation of a fractional number x is illustrated in Figure 2 below



The word-length is B(= M + 1) bits, i.e. M magnitude bits and one sign bit. The most significant bit (MSB) is the sign bit, which represents the sign of the number as follows.

$$b_0 = \begin{cases} 0, & \text{if } x \geq 0 \\ 1, & \text{otherwise} \end{cases}$$

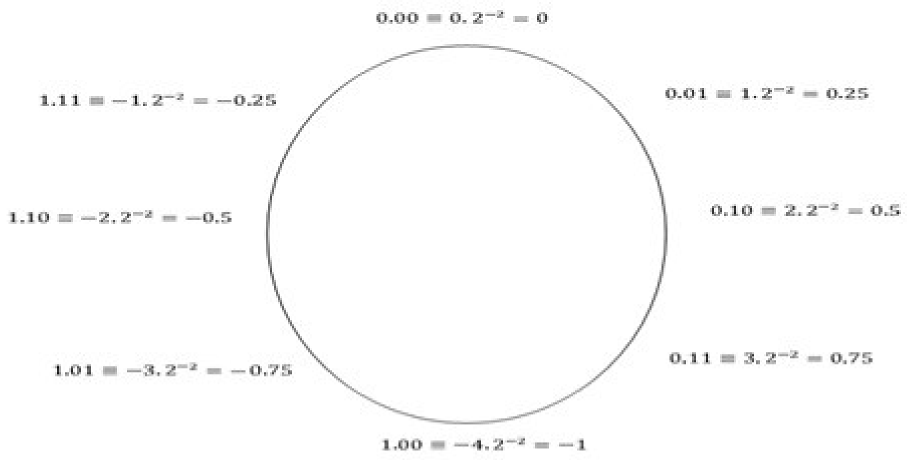
The remaining M bits give the magnitude of the number. The rightmost bit bM is called the least significant bit (LSB), which represents precision of the number.

The decimal value corresponding to a binary fractional number x can be expressed as,

$$x = b_0 + b_1 2^{-1} + b_2 2^{-2} + \cdots + b_M 2^{-M}$$

$$x = b_0 + \sum_{m=1}^{M} b_m 2^{-m}$$

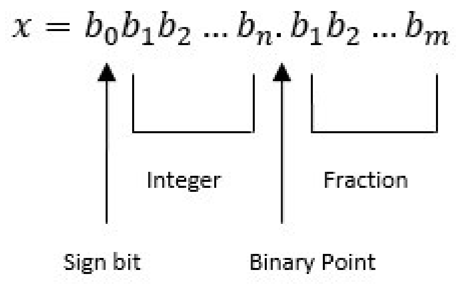
Below figure 3 provides a visual representation for a 3 bit fractional representation.


$0.00 \equiv 0.2^{-2} = 0$
$1.11 \equiv -1.2^{-2} = -0.25$
$0.01 \equiv 1.2^{-2} = 0.25$
$1.10 \equiv -2.2^{-2} = -0.5$
$0.10 \equiv 2.2^{-2} = 0.5$
$1.01 \equiv -3.2^{-2} = -0.75$
$0.11 \equiv 3.2^{-2} = 0.75$
$1.00 \equiv -4.2^{-2} = -1$

For example, from Figure 2, the easiest way to convert a normalized 16-bit fractional binary number into an integer that can be used by C55x assembler is to move the binary point to the right by 15 bits (at the right of bM). Since shifting the binary point 1 bit right is equivalent to multiplying the fractional number by 2, moving the binary point to the right by 15 bits can be done by multiplying the decimal value by 2^{15} = 32768.

Q-format

Q-format is a formal mechanism to keep track of radix or Fixed point. Format Qnm is illustrated in Figure 4 below, where m = M = B - 1 . There are n bits to the left of the binary point representing integer portion, and m bits to the right, representing fractional value.



The most popular fractional number format is Q0.15 format (n = 0 and m = 15), which is simply referred to as Q15 format since there are 15 fractional bits.

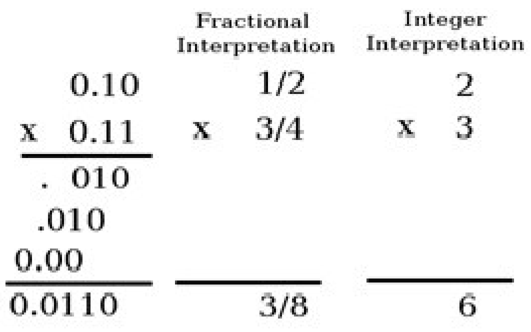
Integer Word-Length (IWL)

The number of bits required to express the integer part of a Floating point number is called as the integer word-length of that number. IWL of a number gives us an idea of the maximum precision with which it can be represented in a finite sized shift register. For example, the IWL of 56:25 is 6 (Why?). Therefore, it can be stored in a 16 bit register just by using the 6 LSBs. But, this results in wastage of the 9 MSBs which all are zeros. Thus, to store 56:25 in a 16 bit register, we will have to multiply it by 2^{15-6}, i.e., $2^9 = 512$, so that it occupies the whole 16 bit register. Note that, the decimal point is implicit while storing the Floating point number. To understand this better, carry out the multiplication yourself and represent the product in signed binary format. Compare this representation with that of 56. This concept of IWL and its use for effective storage is used in section 5.

Round-off Error in Fixed Point Implementation

Fixed-point arithmetic is often used with DSP hardware for real-time processing because it offers fast operation and relatively economical implementation. Its drawbacks include a small dynamic range and low resolution. A Fixed-point representation also leads to arithmetic errors, which if not handled correctly will lead to erroneous operations. These errors have to be handled in such a way as to have some trade-o_ among accuracy, speed and cost.

Consider the multiplication of two binary fractions, as shown here.



	Fractional Interpretation	Integer Interpretation
0.10	1/2	2
x 0.11	x 3/4	x 3
. 010		
.010		
0.00		
0.0110	3/8	6

From above calculation, we see that full-precision multiplication almost doubles the number of bits. If we wish to return the product to a b-bit representation, we must truncate the (b 1) least signi_cant bits. However this introduces truncation error. Truncation error is also known as round o_ error (the number is rounded to the nearest b-bit fractional value rather than truncated). This type of error occurs after the multiplication.

Example code:
Go through below example. Two arrays x floating and y floating of size 100 each, are defined globally.
We want to perform the operation:
y floating[i] = Ax floating[i] + B ;where A = 3:2 and B = 2:7.

The function compute floating() carries out this operation with the help of floating point variables. On the other hand, the compute fixed() deals with only the finite precision data-types such as short int, long int etc. It demonstrates the steps involved in the conversion of floating-point computations to fixed-point. This code is self explanatory.

For better understanding, go to:
https://www.youtube.com/watch?v=om7Bvk7WzJg&list=PLV5Sz2W8vMyagrItexjFRw2X_X99qpX6v

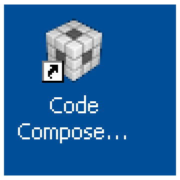
This is another method of looking at the calculations implemented in compute fixed.

Try:

- Function compute_floating() computes y[i] = Ax[i]+B using regular floating-point arithmetic. Firstly, call only this function in main and perform profiling to record the clock-cycle count of its execution.
- Repeat this exercise with compute_fixed() function (Call only one of the two functions at a time in main).

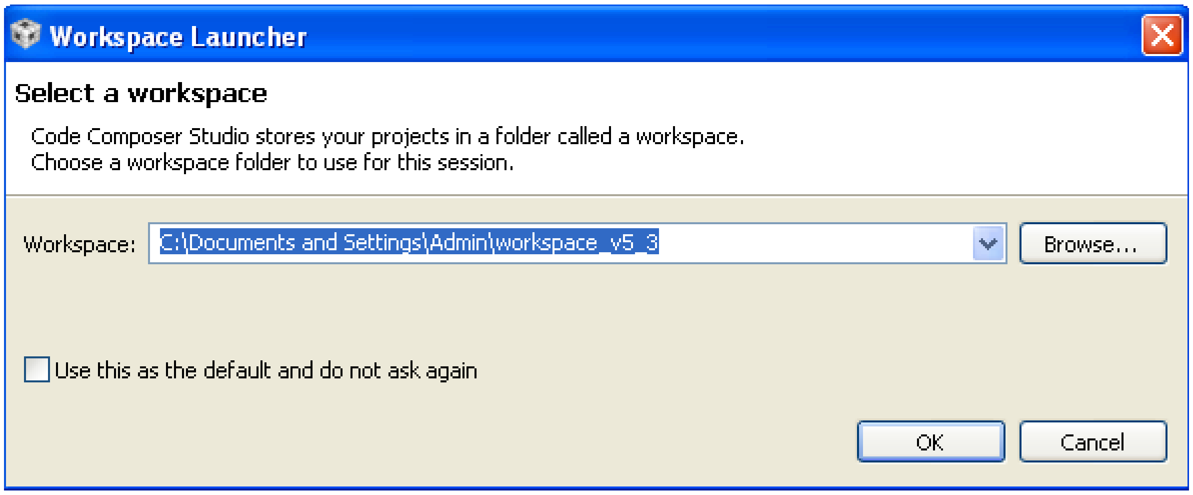
Steps for importing new project:

Open *CCS V5.3* from desktop shortcut



It will open default *CCS V5* screen.

Then it will ask for workspace path
Select path “*C:\Documents and Settings\<User Name>\workspace_v5_3*” for windows XP OS
Select path “*C:\Users\<User Name>\workspace_v5_3*” for windows7 OS

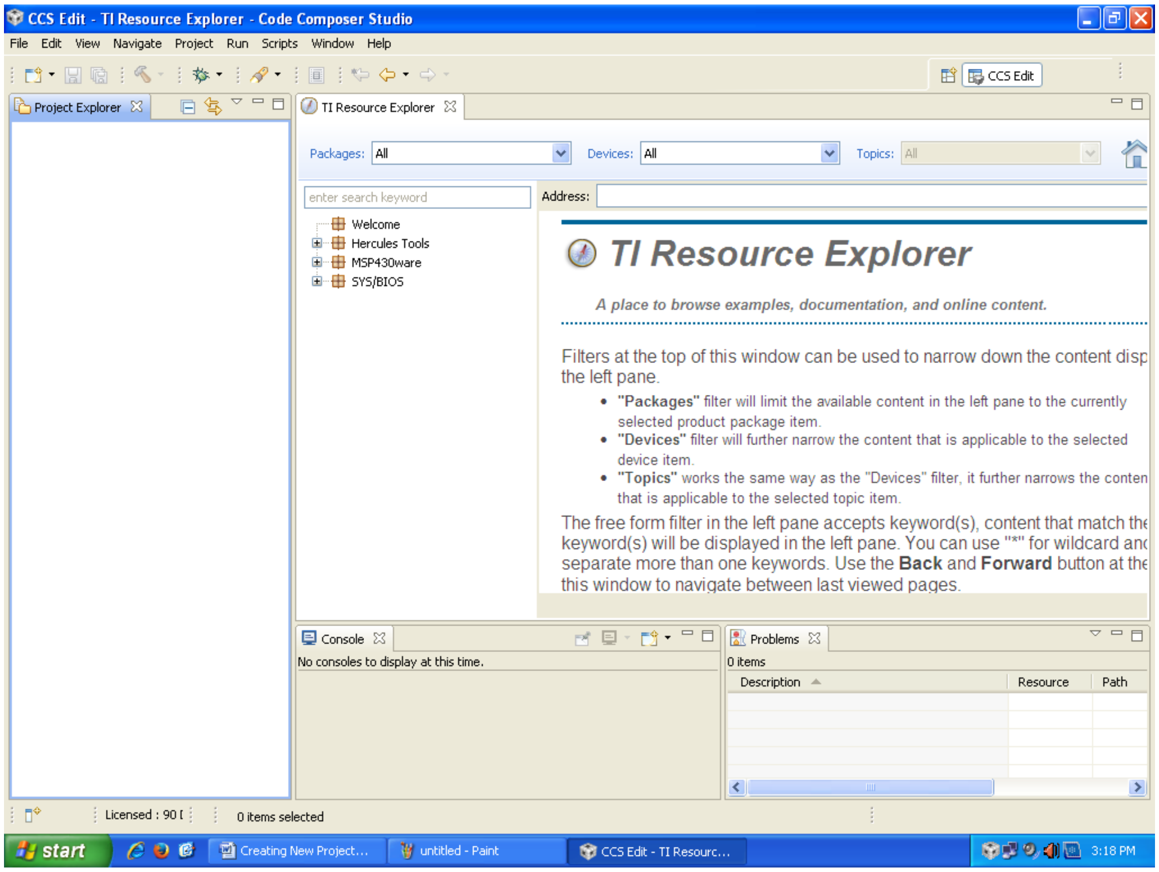

Workspace Launcher

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: C:\Documents and Settings\Admin\workspace_v5_3

Then it will open Default CCS5 screen as shown below


CCS Edit - TI Resource Explorer - Code Composer Studio

TI Resource Explorer

A place to browse examples, documentation, and online content.

Filters at the top of this window can be used to narrow down the content disp the left pane.

- **"Packages"** filter will limit the available content in the left pane to the currently selected product package item.
- **"Devices"** filter will further narrow the content that is applicable to the selected device item.
- **"Topics"** works the same way as the "Devices" filter, it further narrows the conten that is applicable to the selected topic item.

The free form filter in the left pane accepts keyword(s), content that match the keyword(s) will be displayed in the left pane. You can use "*" for wildcard anc separate more than one keywords. Use the **Back** and **Forward** button at the this window to navigate between last viewed pages.

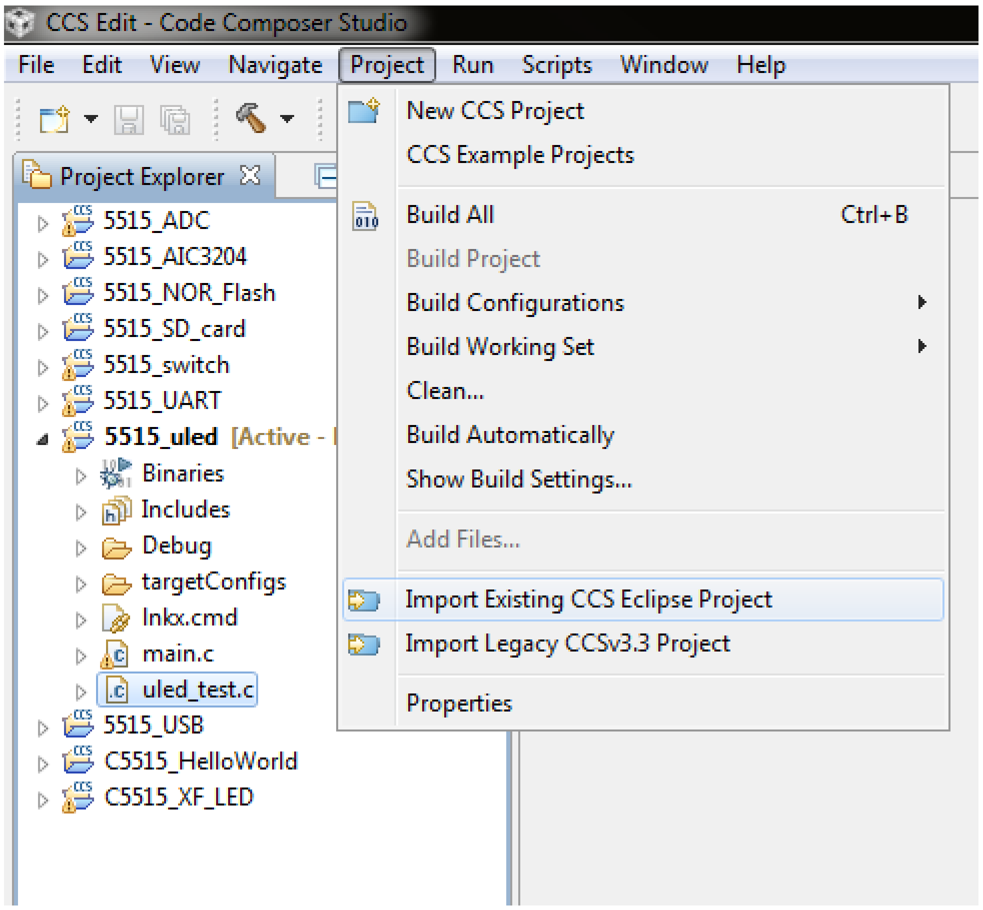
Console
No consoles to display at this time.

Problems
0 items

Description	Resource	Path

Licensed : 90 I
0 items selected

Click “*Project -> Import Existing CCS Eclipse Project*” menu.



- Now Copy the project path and paste it to “Select search-directory” location and press enter. Or you can browse the path also by clicking browse.
- Select project “IIT_LAB5.1” and also select “Copy projects into workspace” and **Finish**
- Here project is already imported and it can be seen from “**project explorer**”
- Compile the program by “*right click-> build project*” or “*right click-> rebuild project*” as shown. It will generate “IIT_ LAB5.1.*out”* file in “*debug*” folder.
- It will generate IIT_ LAB5.1.*out* file in “*debug*” folder.

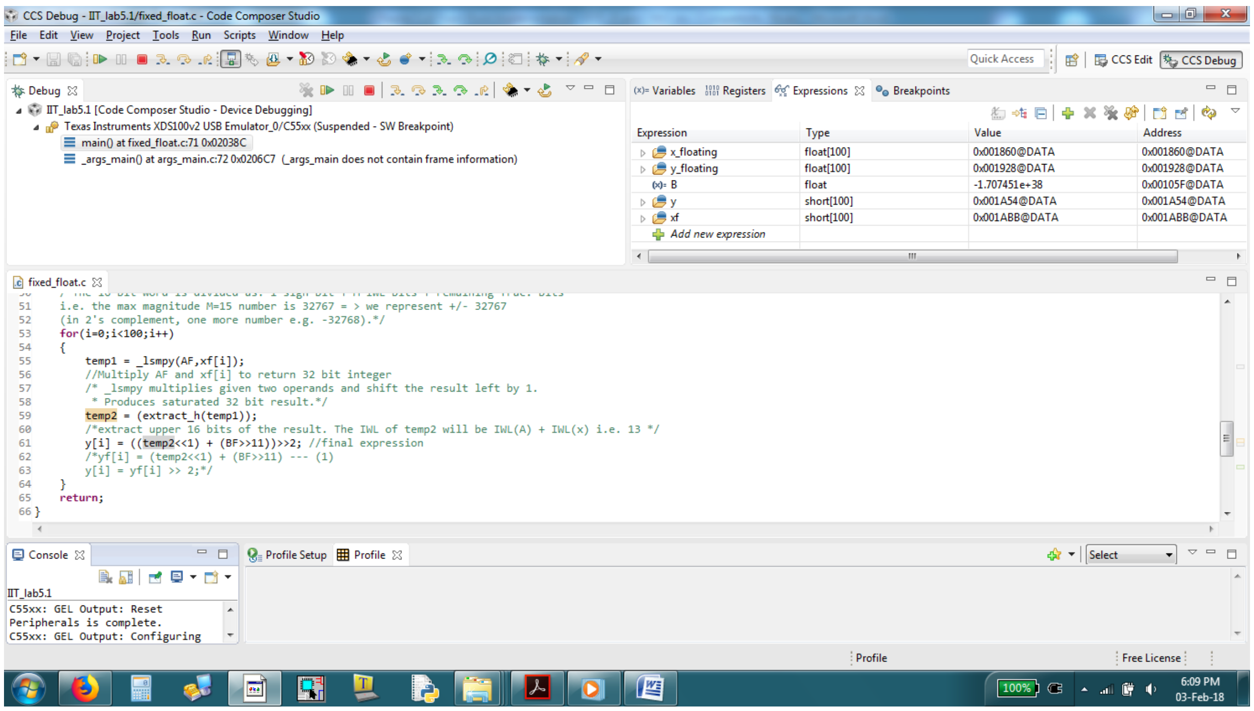
How to Run Program:

Hardware Connection:

- Power on EPB_C5515 hardware using +5V Power supply **or** USB A-to-B cable
- Connect XDS100V2 with EPB_C5515 using USB A-to-miniA cable with CPU
- Reset DSP kit EPB_C5515 by pressing RESET switch.

Steps to Run Program:

- Now to debug the program click “**debug**” icon **OR** from “**run->debug**” menu.
- It will configure/connect EPB_C5515 kit with the CCSV5 using XDS100V2 and download the program in C5515 CPU. It will be done automatically.



IIT_lab5.1 [Code Composer Studio - Device Debugging]
Texas Instruments XDS100v2 USB Emulator_0/C55xx (Suspended - SW Breakpoint)
main() at fixed_float.c:71 0x02038C
_args_main() at args_main.c:72 0x0206C7 (_args_main does not contain frame information)

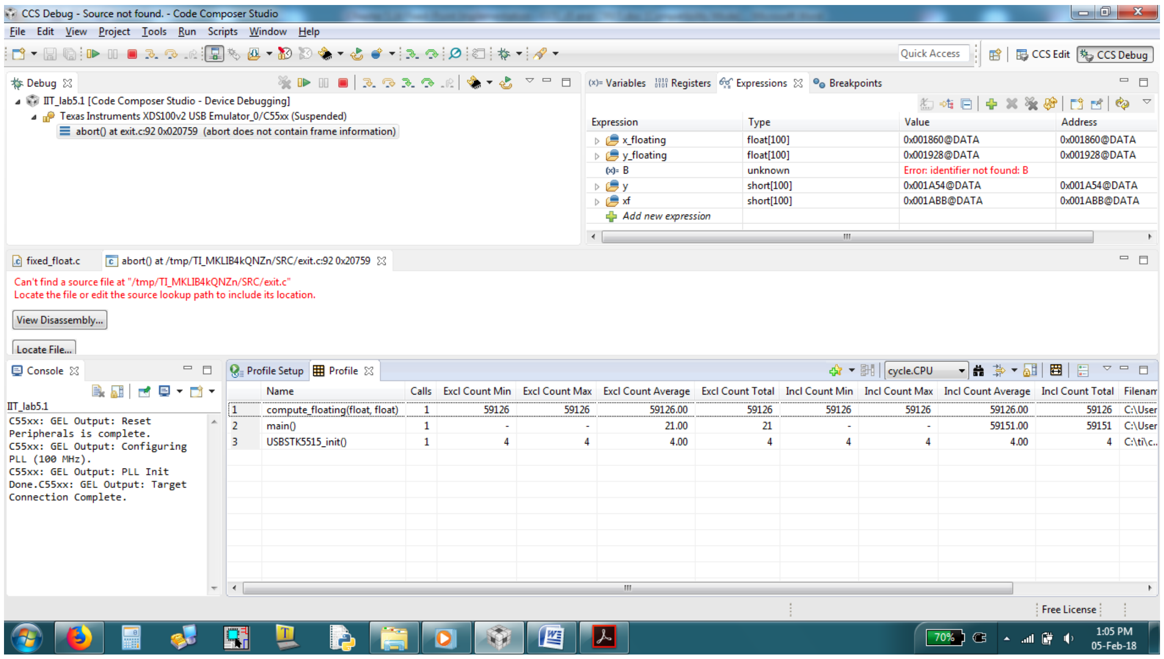
Expression	Type	Value	Address
x_floating	float[100]	0x001860@DATA	0x001860@DATA
y_floating	float[100]	0x001928@DATA	0x001928@DATA
B	float	-1.707451e+38	0x00105F@DATA
y	short[100]	0x001A54@DATA	0x001A54@DATA
xf	short[100]	0x001ABB@DATA	0x001ABB@DATA
Add new expression			

```
    i.e. the max magnitude M=15 number is 32767 = > we represent +/- 32767
    (in 2's complement, one more number e.g. -32768).*/
    for(i=0;i<100;i++)
    {
        temp1 = _lsmpy(AF,xf[i]);
        //Multiply AF and xf[i] to return 32 bit integer
        /* _lsmpy multiplies given two operands and shift the result left by 1.
         * Produces saturated 32 bit result.*/
        temp2 = (extract_h(temp1));
        /*extract upper 16 bits of the result. The IWL of temp2 will be IWL(A) + IWL(x) i.e. 13 */
        y[i] = ((temp2<<1) + (BF>>11))>>2; //final expression
        /*yf[i] = (temp2<<1) + (BF>>11) --- (1)
        y[i] = yf[i] >> 2;*/
    }
    return;
}
```

IIT_lab5.1

```
C55xx: GEL Output: Reset
Peripherals is complete.
C55xx: GEL Output: Configuring
```

Apply profiling for “compute_floating(A,B)” and “compute_fixed()” function.
Apply profiling steps as per the previous lab-4 and check the result for “compute_floating(A,B)” as shown below


CCS Debug - Source not found. - Code Composer Studio

IIT_lab5.1 [Code Composer Studio - Device Debugging]
Texas Instruments XDS100v2 USB Emulator_0/C55xx (Suspended)
abort() at exit.c:92 0x020759 (abort does not contain frame information)

Expression	Type	Value	Address
x_floating	float[100]	0x001860@DATA	0x001860@DATA
y_floating	float[100]	0x001928@DATA	0x001928@DATA
B	unknown	Error: identifier not found: B	
y	short[100]	0x001A54@DATA	0x001A54@DATA
xf	short[100]	0x001ABB@DATA	0x001ABB@DATA
Add new expression			

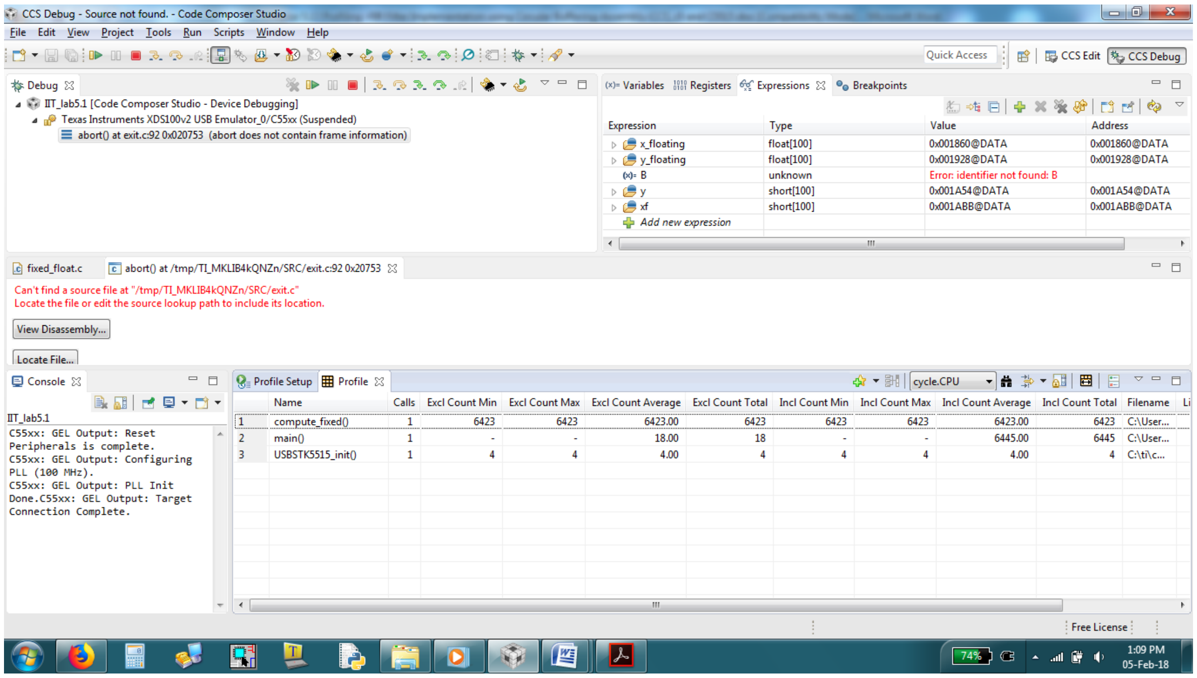
abort() at /tmp/TI_MKLIB4kQNZn/SRC/exit.c:92 0x20759

Can't find a source file at "/tmp/TI_MKLIB4kQNZn/SRC/exit.c"
Locate the file or edit the source lookup path to include its location.

```
IIT_lab5.1
C55xx: GEL Output: Reset
Peripherals is complete.
C55xx: GEL Output: Configuring
PLL (100 MHz).
C55xx: GEL Output: PLL Init
Done.C55xx: GEL Output: Target
Connection Complete.
```

	Name	Calls	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	Filenam
1	compute_floating(float, float)	1	59126	59126	59126.00	59126	59126	59126	59126.00	59126	C:\User
2	main()	1	-	-	21.00	21	-	-	59151.00	59151	C:\User
3	USBSTK5515_init()	1	4	4	4.00	4	4	4	4.00	4	C:\ti\c..

Apply profiling steps as per the previous lab-4 and check the result for “compute_fixed()” as shown below


CCS Debug - Source not found. - Code Composer Studio

IIT_lab5.1 [Code Composer Studio - Device Debugging]
Texas Instruments XDS100v2 USB Emulator_0/C55xx (Suspended)
abort() at exit.c:92 0x020753 (abort does not contain frame information)

Expression	Type	Value	Address
x_floating	float[100]	0x001860@DATA	0x001860@DATA
y_floating	float[100]	0x001928@DATA	0x001928@DATA
B	unknown	Error: identifier not found: B	
y	short[100]	0x001A54@DATA	0x001A54@DATA
xf	short[100]	0x001ABB@DATA	0x001ABB@DATA
Add new expression			

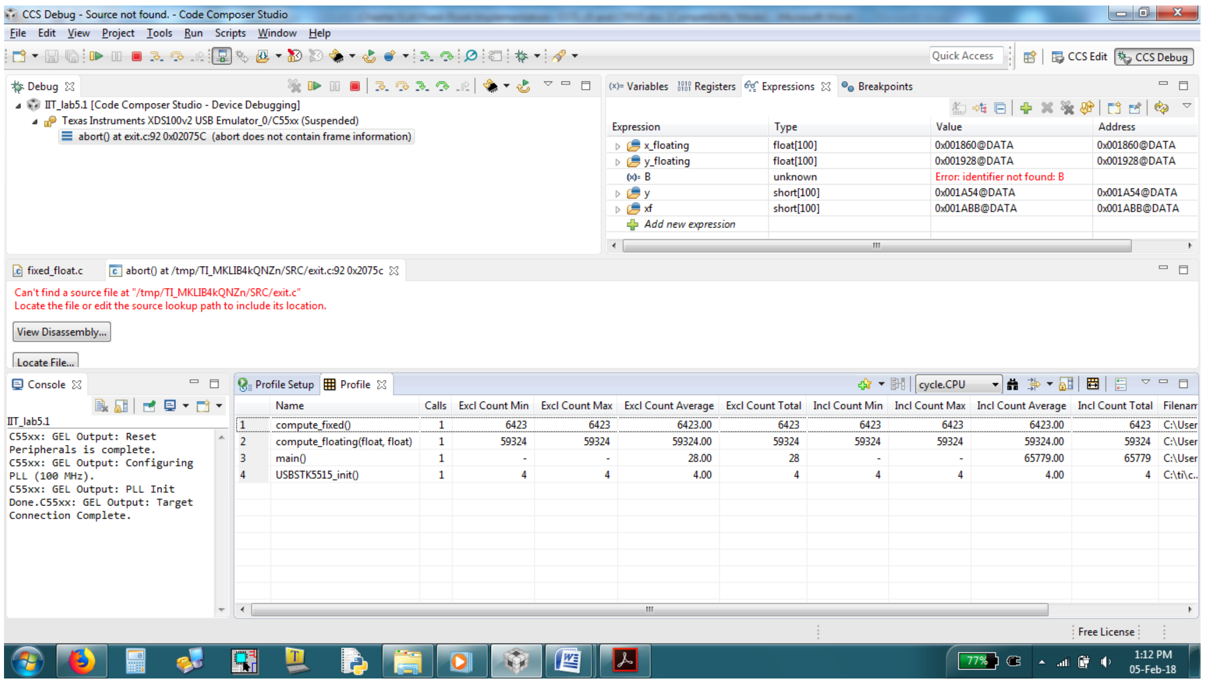
abort() at /tmp/TI_MKLIB4kQNZn/SRC/exit.c:92 0x20753

Can't find a source file at "/tmp/TI_MKLIB4kQNZn/SRC/exit.c"
Locate the file or edit the source lookup path to include its location.

```
IIT_lab5.1
C55xx: GEL Output: Reset
Peripherals is complete.
C55xx: GEL Output: Configuring
PLL (100 MHz).
C55xx: GEL Output: PLL Init
Done.C55xx: GEL Output: Target
Connection Complete.
```

	Name	Calls	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	Filename
1	compute_fixed()	1	6423	6423	6423.00	6423	6423	6423	6423.00	6423	C:\User...
2	main()	1	-	-	18.00	18	-	-	6445.00	6445	C:\User...
3	USBSTK5515_init()	1	4	4	4.00	4	4	4	4.00	4	C:\ti\c...

Combine results are as shown here.



CCS Debug - Source not found. - Code Composer Studio

IIT_lab5.1 [Code Composer Studio - Device Debugging]
Texas Instruments XDS100v2 USB Emulator_0/C55xx (Suspended)
abort() at exit.c:92 0x02075C (abort does not contain frame information)

Expression	Type	Value	Address
x_floating	float[100]	0x001860@DATA	0x001860@DATA
y_floating	float[100]	0x001928@DATA	0x001928@DATA
B	unknown	Error: identifier not found: B	
y	short[100]	0x001A54@DATA	0x001A54@DATA
xf	short[100]	0x001ABB@DATA	0x001ABB@DATA
Add new expression			

abort() at /tmp/TI_MKLIB4kQNZn/SRC/exit.c:92 0x2075c

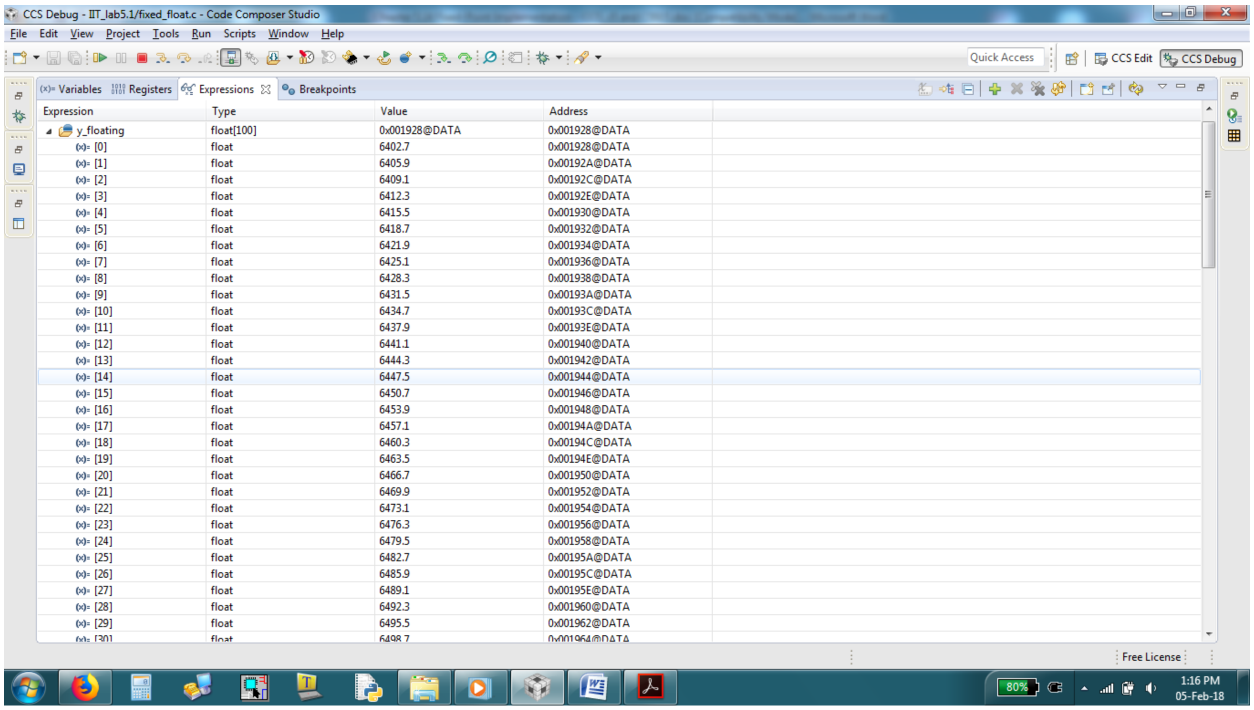
Can't find a source file at "/tmp/TI_MKLIB4kQNZn/SRC/exit.c"
Locate the file or edit the source lookup path to include its location.

```
IIT_lab5.1
C55xx: GEL Output: Reset
Peripherals is complete.
C55xx: GEL Output: Configuring
PLL (100 MHz).
C55xx: GEL Output: PLL Init
Done.C55xx: GEL Output: Target
Connection Complete.
```

	Name	Calls	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total	Incl Count Min	Incl Count Max	Incl Count Average	Incl Count Total	Filenam
1	compute_fixed()	1	6423	6423	6423.00	6423	6423	6423	6423.00	6423	C:\User
2	compute_floating(float, float)	1	59324	59324	59324.00	59324	59324	59324	59324.00	59324	C:\User
3	main()	1	-	-	28.00	28	-	-	65779.00	65779	C:\User
4	USBSTK5515_init()	1	4	4	4.00	4	4	4	4.00	4	C:\ti\c..

At this time check the variable "y_floating" and "y" in watch window and compare results

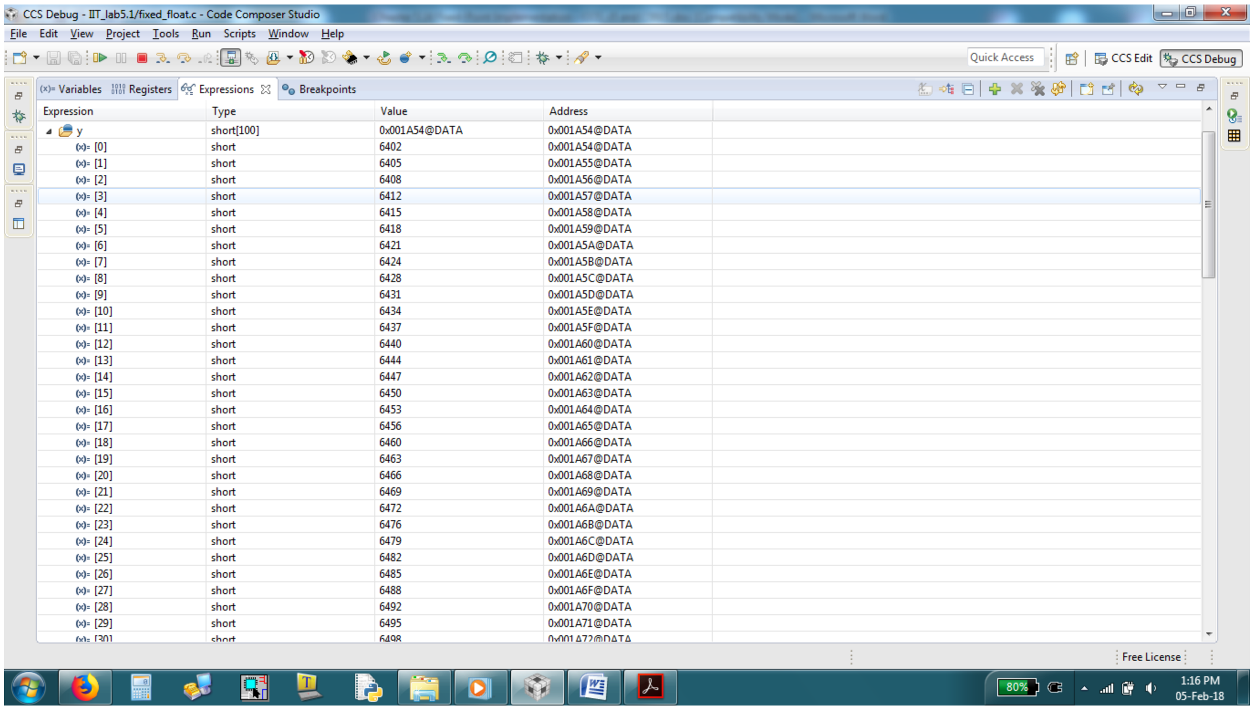
- y_floating



CCS Debug - IIT_lab5.1/fixed_float.c - Code Composer Studio

Expression	Type	Value	Address
y_floating	float[100]	0x001928@DATA	0x001928@DATA
[0]	float	6402.7	0x001928@DATA
[1]	float	6405.9	0x00192A@DATA
[2]	float	6409.1	0x00192C@DATA
[3]	float	6412.3	0x00192E@DATA
[4]	float	6415.5	0x001930@DATA
[5]	float	6418.7	0x001932@DATA
[6]	float	6421.9	0x001934@DATA
[7]	float	6425.1	0x001936@DATA
[8]	float	6428.3	0x001938@DATA
[9]	float	6431.5	0x00193A@DATA
[10]	float	6434.7	0x00193C@DATA
[11]	float	6437.9	0x00193E@DATA
[12]	float	6441.1	0x001940@DATA
[13]	float	6444.3	0x001942@DATA
[14]	float	6447.5	0x001944@DATA
[15]	float	6450.7	0x001946@DATA
[16]	float	6453.9	0x001948@DATA
[17]	float	6457.1	0x00194A@DATA
[18]	float	6460.3	0x00194C@DATA
[19]	float	6463.5	0x00194E@DATA
[20]	float	6466.7	0x001950@DATA
[21]	float	6469.9	0x001952@DATA
[22]	float	6473.1	0x001954@DATA
[23]	float	6476.3	0x001956@DATA
[24]	float	6479.5	0x001958@DATA
[25]	float	6482.7	0x00195A@DATA
[26]	float	6485.9	0x00195C@DATA
[27]	float	6489.1	0x00195E@DATA
[28]	float	6492.3	0x001960@DATA
[29]	float	6495.5	0x001962@DATA
[30]	float	6498.7	0x001964@DATA

- y


CCS Debug - IIT_lab5.1/fixed_float.c - Code Composer Studio
File Edit View Project Tools Run Scripts Window Help
Quick Access
CCS Edit
CCS Debug
Variables Registers Expressions Breakpoints
Free License
1:16 PM
05-Feb-18

Expression	Type	Value	Address
y	short[100]	0x001A54@DATA	0x001A54@DATA
[0]	short	6402	0x001A54@DATA
[1]	short	6405	0x001A55@DATA
[2]	short	6408	0x001A56@DATA
[3]	short	6412	0x001A57@DATA
[4]	short	6415	0x001A58@DATA
[5]	short	6418	0x001A59@DATA
[6]	short	6421	0x001A5A@DATA
[7]	short	6424	0x001A5B@DATA
[8]	short	6428	0x001A5C@DATA
[9]	short	6431	0x001A5D@DATA
[10]	short	6434	0x001A5E@DATA
[11]	short	6437	0x001A5F@DATA
[12]	short	6440	0x001A60@DATA
[13]	short	6444	0x001A61@DATA
[14]	short	6447	0x001A62@DATA
[15]	short	6450	0x001A63@DATA
[16]	short	6453	0x001A64@DATA
[17]	short	6456	0x001A65@DATA
[18]	short	6460	0x001A66@DATA
[19]	short	6463	0x001A67@DATA
[20]	short	6466	0x001A68@DATA
[21]	short	6469	0x001A69@DATA
[22]	short	6472	0x001A6A@DATA
[23]	short	6476	0x001A6B@DATA
[24]	short	6479	0x001A6C@DATA
[25]	short	6482	0x001A6D@DATA
[26]	short	6485	0x001A6E@DATA
[27]	short	6488	0x001A6F@DATA
[28]	short	6492	0x001A70@DATA
[29]	short	6495	0x001A71@DATA
[30]	short	6498	0x001A72@DATA

Exercise:
Modify the given C code (compute floating()) with the values of x floating[i] taking floating point values between 4000 and 4100 (as specified below) and convert this code to fixed point inside the function compute fixed(). The floating point values may be assigned to x floating[] as shown in the following code snippet.

```
for (i=4000; i<4100; i++)
{
        x_floating[i-4000] = i + 0.875;
}
```

So, instead of [2000; 2099], x[i] will take values from the range [4000:875; 4099:875] and you will have to calculate y[i] = Ax[i] + B. Note that, since x floating[i] itself contains floating point values, we cannot use the trick of calculating xf[i] by just bitwise right shifting the values in x[i]. You will have to actually multiply the array x floating[i] by correct value to get xf[i].