

Lab 9: FPU

Objective

In this lab you will enable the FPU to run and profile floating-point code.

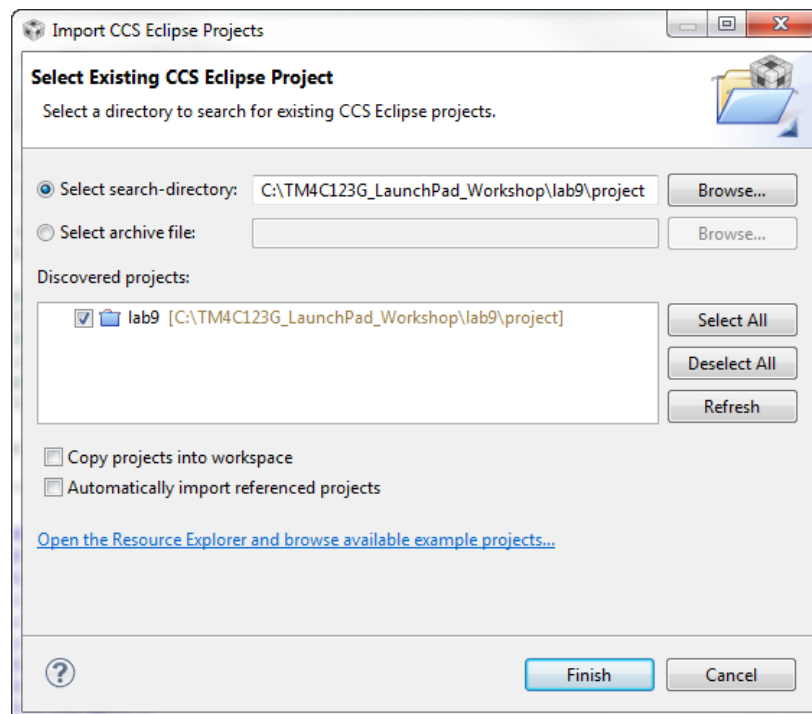
Procedure

Import lab9

1. We have already created the lab9 project for you with `main.c`, a startup file and all necessary project and build options set.

► Maximize Code Composer and click Project → Import Existing CCS Eclipse Project. Make the settings shown below and click Finish.

Make sure that the “Copy projects into workspace” checkbox is **unchecked**.



► Expand the project.

Browse the Code

2. In order to save some time, we're going to browse existing code rather than enter it line by line. ► Open `main.c` in the editor pane and copy/paste the code below into it. The code is fairly simple. We'll use the FPU to calculate a full sine wave cycle inside a 100 datapoint long array. This file is saved in your `lab9/project` folder as `main.txt`.

```
#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/fpu.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#define SERIES_LENGTH 100
float gSeriesData[SERIES_LENGTH];

int32_t i32DataCount = 0;

int main(void)
{
    float fRadians;

    ROM_FPULazyStackingEnable();
    ROM_FPUEnable();

    ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN);

    fRadians = ((2 * M_PI) / SERIES_LENGTH);

    while(i32DataCount < SERIES_LENGTH)
    {
        gSeriesData[i32DataCount] = sinf(fRadians * i32DataCount);
        i32DataCount++;
    }

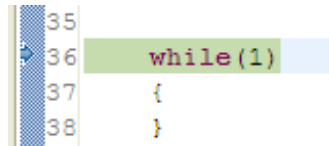
    while(1)
    {
    }
}
```

3. At the top of `main.c`, look first at the includes, because there are a couple of new ones:
 - **math.h** – the code uses the `sinf()` function prototyped by this header file
 - **fpu.h** – support for Floating Point Unit
4. Next is an `ifndef` construct. Just in case `M_PI` is not already defined, this code will do that for us.
5. Types and defines are next:
 - **SERIES_LENGTH** – this is the depth of our data buffer
 - **float gSeriesData[SERIES_LENGTH]** – an array of floats `SERIES_LENGTH` long
 - **i32dataCount** – a counter for our computation loop

6. Now we've reached `main()`:
 - We'll need a variable of type float called `fRadians` to calculate sine
 - Turn on Lazy Stacking (as covered in the presentation)
 - Turn on the FPU (remember that from reset it is off)
 - Set up the system clock for 50MHz
 - A full sine wave cycle is 2π radians. Divide 2π by the depth of the array.
 - The `while()` loop will calculate the sine value for each of the 100 values of the angle and place them in our data array
 - An endless loop at the end

Build, Download and Run the Code

7. ► Click the Debug button to build and download the code to the TM4C123GH6PM flash memory. When the process completes, ► click the Resume button to run the code.
8. ► Click the Suspend button to halt code execution. Note that execution was trapped in the `while(1)` loop.

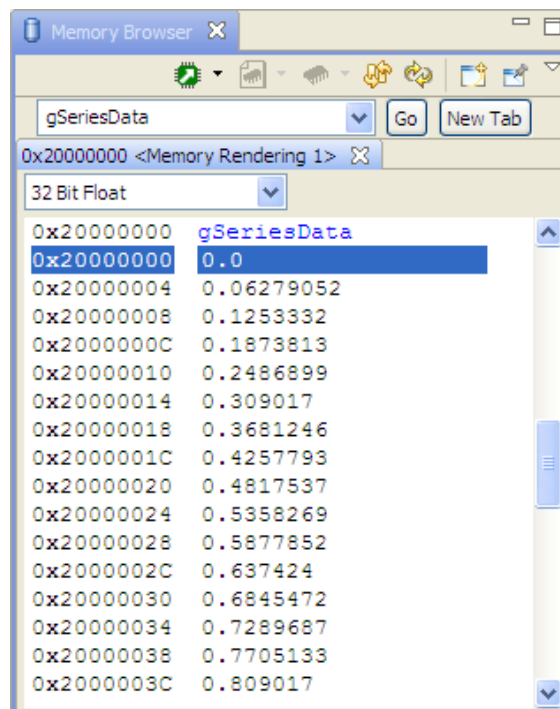


```

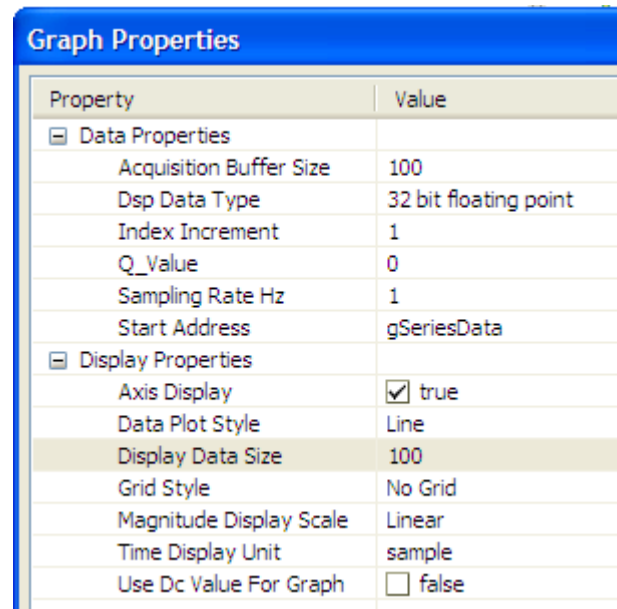
35
36 while(1)
37 {
38

```

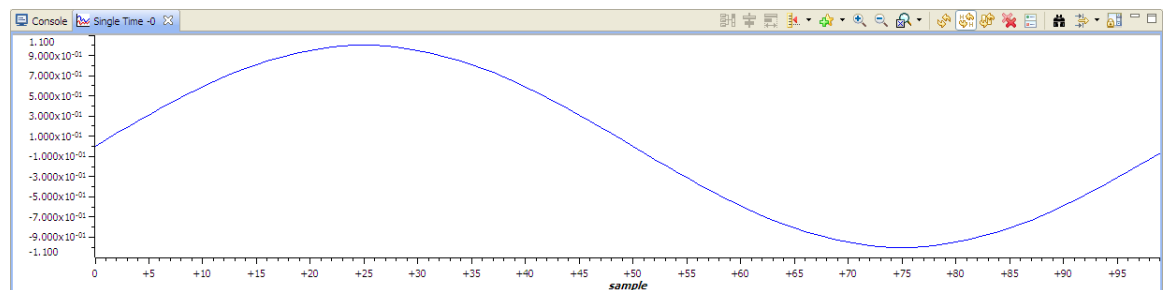
9. ► If your Memory Browser isn't currently visible, Click View → Memory Browser on the CCS menu bar. Enter `gSeriesData` in the address box and click Go. In the box that says Hex 32 Bit – TI Style, click the down arrow and select 32 Bit Floating Point. You will see the sine wave data in memory like the screen capture below:



10. Is that a sine wave? It's hard to see from numbers alone. We can fix that. On the CCS menu bar, click Tools → Graph → Single Time. When the Graph Properties dialog appears, make the selections show below and click OK.



You will see the graph below at the bottom of your screen:



Profiling the Code

11. An interesting thing to know would be the amount of time it takes to calculate those 100 sine values.

► On the CCS menu bar, click View → Breakpoints. Look in the upper right area of the CCS display for the Breakpoints tab.

12. ► Remove any existing breakpoints by clicking Run → Remove All Breakpoints. In the `main.c`, set a breakpoint by double-clicking in the gray area to the left of the line containing:

```
fRadians = ((2 * M_PI) / SERIES_LENGTH);
```

```
fRadians = ((2 * M_PI) / SERIES_LENGTH);
while(i32DataCount < SERIES_LENGTH)
{
    gSeriesData[i32DataCount] = sinf(fRadians * i32DataCount);
    i32DataCount++;
}
```

13. ► Click the Restart button to restart the code from `main()`, and then click the Resume button to run to the breakpoint.
14. ► Right-click in the Breakpoints pane and Select Breakpoint (Code Composer Studio) → Count event. Leave the Event to Count as Clock Cycles in the next dialog and click OK.
15. ► Set another Breakpoint on the line containing `while(1)` at the end of the code. This will allow us to measure the number of clock cycles that occur between the two breakpoints.
16. Note that the count is now 0 in the Breakpoints pane. ► Click the Resume button to run to the second breakpoint. When code execution reaches the breakpoint, execution will stop and the cycle count will be updated. Our result is show below:



Variables Expressions Registers Breakpoints					
Identity	Name		Condition	Count	Action
<input checked="" type="checkbox"/>	Count Event			34996	
<input checked="" type="checkbox"/>	main.c, line 27 Breakpoint			0 (0)	Remain Halted
<input checked="" type="checkbox"/>	main.c, line 36 Breakpoint			0 (0)	Remain Halted

17. A cycle count of 34996 means that it took about 350 clock cycles to run each calculation and update the `i32dataCount` variable (plus some looping overhead). Since the System Clock is running at 50 MHz, each loop took about 7μS, and the entire 100 sample loop required about 700 μS.
18. ► Right-click in the Breakpoints pane and select Remove All, and then click Yes to remove all of your breakpoints.
19. ► Click the Terminate button to return to the CCS Edit perspective. ► Right-click on Lab9 in the Project Explorer pane, close the project and minimize CCS.

Follow the submission guideline to be awarded points for this Lab.

Task01: Submit a comprehensive commented file of the original code.

Task 02: Modify the code to implement the below equation to generate a frequency of 5 Hz. Display the equation for 1 sec.

$$1.0 * \sin(2\pi 50t) + 0.5 * \cos(2\pi 200t)$$

Follow the submission guideline to be awarded points for this Lab.

Submit the following for all Labs:

1. In the document, for each task submit the modified or included code (only) with highlights and justifications of the modifications. Also include the comments.
2. Create a Github repository with a random name (no CPE/403, Lastname, Firstname). Place all labs under the root folder TIVAC, sub-folder named LABXX, with one document and one video link file for each lab, place modified c files named as LabXX-TYY.c.
3. If multiple c files or other libraries are used, create a folder LabXX-TYY and place these files inside the folder.
4. The folder should have a) Word document (see template), b) source code file(s) with startup_ccs.c and other include files, c) text file with youtube video links (see template).