CPE 490 Lab 5 S13

## Goals

1. Learn the power of having a hardware multiplier
2. Learn the how basic math functions effect the resources in the dsPIC.
3. Learn how to use the fixed point data type.

## Overview

This lab will allow you to explore what math statements in C will compile to and how long they will take to execute in real time in an embedded system. This investigation will help you balance precision of math calculations with real time performance constraint.

All times should be measured in actual hardware not the simulator. Timer 1 can be used to keep track of time of execution. Make sure that the primary external clock oscillator is used to drive Timer 1 at a rate of 16MHz (main oscillator frequency = 32 MhZ. Refer to Lab 3 for setting up the clock oscillator for this frequency.

You can put all three designs in one lab report. The goals, equipment used can cover all 3 designs. The sections on design specification, design (including flow chart and code), verification and conclusion should cover each design (i.e. flow charts and code printout for each design). I recommend making a file and project for each design in the MPLAB environment.

## Design 1

Write a C language program for the dsPIC33FJ256GP710A that will multiply two unsigned 8 bit numbers together using repetitive addition and place the answer in an unsigned 16 bit number. Realize that this is how we would do a multiply on a simple 8 but machine like what we used in CPE303. Using Timer1 measure the amount of time it takes to do the multiply of 255 \* 255 in this method. To get good resolution, remove the 256 prescale that we used in TMR1 in previous labs. Make sure you know what time each count of the timer represents. Of course since we have no input device the inputs to this multiply routine are just initialized in the code. Assign the time recorded to a variable and use the watch window to record time of execution. Be careful not to halt the running of the program with a breakpoint while timing how fast it executes.

After this routine executes do the multiply of 255 X 255 using a regular C assignment statement with a multiply operator, again using timer 1 to measure the time it takes to accomplish the multiply.

The data gathered with Timer1 should be from actual execution of the code on the Explorer 16 board.

Record this data for your lab and state it in the verification portion of your lab report. Draw an appropriate conclusion on the performance difference between a MCU with and without a hardware multiplier. Make sure you include a flow diagram and the actual code for design 1 in your lab book for later inclusion in your lab report. Make sure that you are getting reasonable results before going on.

## Design 2

In one C language program, using Timer 1, and the normal C assignments, find out the time that it takes to do the following multiplications:

Char = char \* char

Int = int \* int

Long = long \* long

Long long = long long \* long long

Float = float \* float

Long double = long double \* long double

Select the numbers that you will multiply so that overflow will not occur. Make sure to not select numbers that are even powers of 2, since the compiler might try to optimize the process by doing a shift. In each case record the numbers you multiplied and verify the answer that you got.

**Also check the time it takes to do division** using the same combinations as for multiply. Your lab report should have a table or graph that clearly shows this information. This table should help you write a conclusion to your lab.

## Design 3

In this last design you will convert some floating point numbers to fixed integer format and then write a multiply routine for multiplying to fix point numbers together.

Floating point numbers can be changed to Q16. 16 by including this header file:

#include <libq.h>

The information for this function comes from the 16 library documentation is reprinted next: 

This floating point number will have a sign bit, 15 integer bits, and 16 fractional bits (Q15.16 format). When you use the conversion function the number in Q16 should make sense to you. That is the integer part will be straight forward, the decimal part can be found by finding the integer equivalent of the lower 16 bits and multiplying that by 2-16 .

The function you write should accept two \_Q16 type inputs (see header file libq.h for definition), and return a \_Q16 . This function does not check or doing anything about an overflow. Be aware that even when there is not overflow you will need to think about an integer that is scaled by 2-16 being multiplied with and other number scaled by 2-16 the answer being now scaled by 2-32. What will you have to do to get the answer back to a format that is scaled by 2-16?

Verify that your routine works when an overflow does not occur. Write code to use your new function and measure how much time it takes to execute the function. Compare this to how long a float \* float took from design two above. Draw meaningful conclusions and put it into your report.

Next, do find the time it takes to add together two floating point numbers, and compare it to the time it takes to add two \_Q16 numbers. (Realize that a \_Q16 is really just a long type and that addition does not affect binary points.

None of these designs need to be witnessed by an instructor, but if you are to get credit you must state you data and conclusions for each of the designs clearly in the turned in report.