1. **OBJECTIVES**

Lab 1 aimed to revive our EE 319K skills and develop a set of useful fixed point output routines. The fixed-point routines used to display graphics onto the LCD will be used in subsequent labs. A secondary objective of this lab was to compare the engineering tradeoffs between fixed point and floating point implementations.

1. **ANALYSIS AND DISCUSSION**
   1. **In what way is it good design to minimize the number of arrows in the call graph for your system?**

The more arrows there are, the more complex the call graph and system will be and the harder they will be to read/understand (which would ultimately defeat the purpose of using a call graph).

* 1. **Why is it important for the decimal point to be in the exact same physical position independent of the number being displayed? Think about how this routine could be used with the ST7735\_SetCursor command.**

Without a fixed location, smaller and larger numbers may take up differing lengths of the screen when printed. When iterating through the string (of the number) to be printed, ST7735\_SetCursor can be called at the same index every time to reduce code complexity.

* 1. **When should you use fixed-point over floating point? When should you use**

**floating-point over fixed-point?**

You can use fixed point if you know the range of values you need to represent ahead of time and if this range is small. If you don’t know the range of values ahead of time or if the range is too large, use floating point. If you need to optimize for speed, then you have to see if your processor has the necessary floating point hardware support (otherwise the floating point operations will run very slowly). For systems that don’t optimize for floating point operations, fixed point will be (generally) faster. In general, floating point operations consume more power due to the extra hardware support and overhead. Floating point operations also push extra registers onto the stack, which consumes space and results in more overhead. For this class, we will generally be using fixed point.

* 1. **When should you use binary fixed-point over decimal fixed-point? When should you use decimal fixed-point over binary fixed-point?**

You should use binary fixed point if you need faster mathematical operations (multiplications and divisions will be carried out through shifts, which are very efficient and take zero clock cycles according to Dr. Valvano). You should use decimal fixed point if you want to work with human readable numbers.

* 1. **Give an example application (not mentioned in the book) for fixed-point. Describe the problem and choose an appropriate fixed-point format.**

Advanced robotics need precise fractional rotations of a motor. The number representing the rotation can be used internally as a binary fixed-point number and converted to fractional rotations by the motor driver.

* 1. **Can we use floating point on the ARM Cortex M4? If so, what is the cost?**

Yes but we must enable compiler options and uncomment the appropriate lines in startup.s to enable the floating point unit. The cost would be pushing extra registers onto the stack for floating point operations, increased overhead, and probably extra power usage.

1. **EXTRA-CREDIT FLOATING POINT ANALYSIS**

For this section of the lab, we used systick to time how long each of four different implementations of a temperature-converting subroutine take to operate.

The four tests consisted of a floating-point c program, a fixed-point c program, a floating-point assembly program, and a fixed-point assembly program. The raw cycle counts for each were, 2,457,762; 118,807; 57,359; and 53,273 cycles respectively.

The floating-point c program took the longest of the four, running at 20 times the cycles of the next-fastest implementation, the fixed-point c program. The floating-point assembly program ran surprisingly faster than the fixed-point c program at around half the cycles; this is probably due to the amount of optimization that can be done working in pure assembly. The fixed-point assembly program only ran marginally faster than the floating-point assembly program at 0.08 times faster. With better bench tests (i.e. ones that used more computationally heavy floating point operations, such as division, and many more operations), we would probably see a more significant difference in the run times between the two assembly tests.

Optimally, we should write calculations using fixed-point in assembly, but for relatively simple calculation like this one, optimized floating-point assembly can be used. Using c for floating-point calculations on this microcontroller is out of the question for time sensitive calculations.