Lab #11 – Cycle Counter

Goals:

Write an assembly code function on the Discovery Board that consumes a calibrated number of processor cycles to be used as a time delay function

Use the processor cycle counter peripheral to calibrate delay times

Use your delay function to generate some simple animated graphics

For each step enter the necessary responses into the Canvas assignment text entry tool to let the TA follow your work.

Remember the '`SVC #0`' debugging instruction available with your Discover Board library. That can be inserted in your assembly code any time you need to clarify exactly what is in the core registers R0 - R12.

Refer to the Lecture slides for the following:

Part 1:

a) Download Lab11-DelayMain.c and Lab11-delay.s from the Canvas files area, and in the LabCode folder. Consider the function 'delay' in the assembler source file of this general form that includes an iterative loop:

```
delay:
          ldr  r1, =#10000
          mul  r0, r1
loop:
          nop
          nop
          …
          nop
          nop
          subs r0, #1
          bgt  loop
          bx   lr
```

The body of the loop includes several "NOP" instructions. These are "blank" instructions that go through the Fetch-Decode-Execute phases in the instruction pipeline, but do not actually perform any operation during their Execute phase. (The opcode "NOP" means "no operation".)

Our goal is to design this function so that it takes a known amount of time to execute, and can therefore cause a time delay in the program execution when called. The delay time should be specified by the single unsigned integer function argument in R0, in units of milliseconds. That is, calling `delay(10);` from C code (or from another assembly function) will delay the program 10 milliseconds.

Note that the function argument passed in R0 is first multiplied by 10,000, and then that R0 register value is used as a loop iteration counter. At the bottom of the loop, 1 is subtracted from R0, and the loop continues if the result in R0 is > 0. If there are *k* '`NOP`' instructions in the body of the loop, then

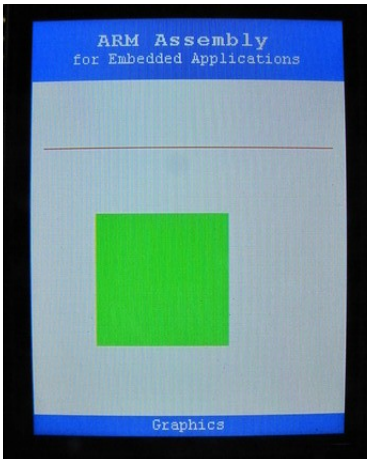
each iteration, between the first ‘NOP’ at the label ‘loop’ and the ‘BGT’ instruction, will run through *k*+2 instructions, including the ‘SUBS’ and ‘BGT’ instructions.

How do we design this function so that its delay time is calibrated in milliseconds? If we were to pass an argument of 1, asking for 1 msec delay, then the loop iteration count in R0 would start out at 10,000 and be decremented by 1 each iteration down to zero. Since we know that the STM32F429ZI processor runs at a cycle time of 180MHz, to consume 1 msec of run time we would have to occupy 180,000 clock cycles. Let’s first assume that all of these *k*+2 instructions require one clock cycle each in the pipeline. What is the required value of k to occupy 180,000 cycles, over a total of 10,000 loops? Edit the assembly code for your expected value of *k* and run the program. You could copy and paste *k* lines of ‘NOP’ instructions, or use the convenient .rept assembler directive, as in, for example:

```
.rept 10
nop
.endr
```

if your expected *k* were 10. Note that the C main program simply makes a record of the processor cycle count before and after your delay function is called with an argument of 1. The difference between these counts is the cycle delay. (There will be one or two dozen extra cycles to account for the various assignment statements and the function call as well, but we can ignore these.) How close did you come to the target of 180,000 cycles? What instruction in the loop is the one requiring more than one pipeline cycle time? Why?

b) Adjust the number of NOP instructions in the loop body to make the cycle count about 180,000 (plus a dozen or two more), which makes a 1 msec delay, when called with an argument of 1. Upload your code and a cell phone photo.



c) Return to a function you wrote in earlier labs that draws some graphical figure on the display screen. Pick either the horizontal or vertical line in Lab #6, or the filled rectangle in Lab #7, as shown in the photo here, and make a copy of its drawing function assembly source file. Modify your drawing function to now take a single unsigned integer argument of the color pixel to write to the display memory. Download the second C main program Lab11-FlashMain.c and edit it to call the drawing function you are using, either line() or rect(). Note that between the two calls to your drawing function with different colors, your delay function is called with an argument of 1000. Build this main C program together with your delay.s assembly code and your graphics drawing function assembly code. This should alternate the color of the geometry at a one second rate. Upload your code and a brief cell phone video clip.

Part 2: Time to work on the current programming project assignment.