Lab 1: Timed LEDs

[10 marks]

Your task, should you not fail, is to make a program that continuously counts up, in binary, on the three LEDs at a rate of two counts per second (i.e. 0.5 s per count). You will be submitting your code to your instructor via a Moodle drop-box, so that you can be graded for your documentation and commenting as well as the functionality of your software.

Here's the rubric:

3 marks: Descriptive file headers, comments throughout, comments are meaningful

7 marks: Your program meets the specifications above. A mark deducted for each error or failure to meet specifications.

Here are some useful clues:

1. You will need to create simple delay loops based on the number of clock cycles per instruction. Since the ANSI C compiler doesn't let you know in advance what instructions it will use, to get full control of the timing you will need to embed some Assembly Language into your code. The following is an example of a timing loop created in this way – you will need to determine the correct values to make a 0.5 s delay loop.

   ```
   asm   LDX   #26667;     //26667 x 3 cycles x 125 ns = 10 ms
   asm   DBNE  X,*;
   ```

   In this example, you should be aware that 26,667 cycles only produced 10 ms of delay, not 500 ms. So, you say, just multiply by 50 to get 500 ms. But wait … that gives you something like 1,333,333 – a number that's too big for the 16-bit X accumulator!

   You've got a few choices available to you at this point. One is to make an ANSI C "for" loop, say, for example, one that uses a 10 ms delay loop 50 times. That's probably good enough, even though we don't know how long it will take to exercise the "for" loop, because it will be a pretty short time, and won't noticeably extend the overall loop.

   A second way is to create a nested loop in Assembly Language, where you use another accumulator (A, B, or Y) as an outside loop counter. This will still add a bit of time to the overall delay, but at least you'll be able to determine how much extra time that is, and maybe even compensate for it (say, for example, by subtracting three clock cycles from the inside loop counter since the outside loop counter will have a 3-cycle DBNE of its own). By the way, you can use labels in an "asm" command (e.g.) `asm  Big_Loop:  LDX #26664;`.

2. How do you get the LEDs to count up? Here's a clue. 000 is actually 0000 0000 in PT1AD1, and 001 is actually 0010 0000 in PT1AD1. What did you add to PT1AD1 to increase from 000 to 001? You might want to keep your result to this in binary, as that should help make your code more understandable.

3. What happens when you get to 7, or 111? Do you need to do anything special, like set limits to flag the end of the count so you can start over? Before you go crazy, try adding 1 to 111, and see what you get as a 3-bit binary value.

4. If you pay close attention to the above clues, your program should be really short. One version has two lines of initialization (one is a function call from SwLED_Lib and the other is an optional initialization of the LEDs to 000), and then there are five lines of code inside the endless loop – that's all! If you find yourself generating pages of code, you haven't thought this assignment through very well!

5. You can check your timing with the oscilloscope at your bench. Use your schematic to determine which CPU pin is associated with the Green LED (or you can probe the resistor connected to the Green LED), and measure the period of the signal on this pin. It should be 1.000 s.

   NOTE: Make sure you put the oscilloscope Ground clip on "DGND" and not one of the power supply pins! More than one student has killed the "–10 V" supply by misplacing the Ground clip!

   Your instructor can help you figure out how to use the oscilloscope in the lab, if you're not comfortable with trying to apply what you know from other scopes.