ECE 367 - Experiment #5
Time-Multiplexed Dual Seven Segment Displays

Spring 2006 Semester

Introduction

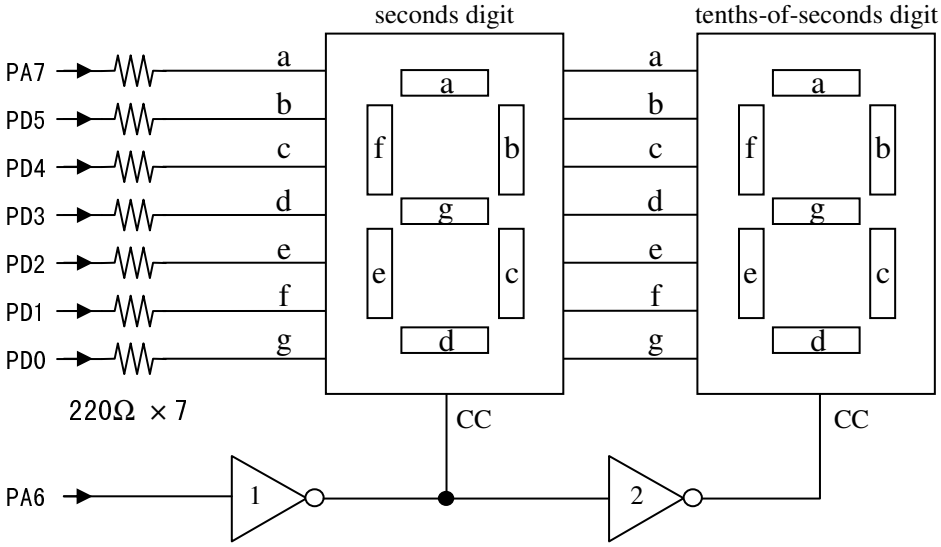
This experiment requires that you construct a circuit interfacing the MicroStamp11 module with two seven segment displays, and write assembly language code to drive them in time-multiplexed fashion to display a count of seconds and tenths-of-seconds.

Required Hardware

In addition to the MicroStamp11 module, this experiment requires two seven segment LED displays, two TTL hex inverters (one 74LS04 IC) and the 220Ω DIP resistors.

Wiring Diagram

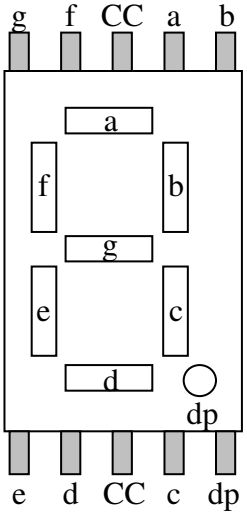
Build the following circuit on a solderless breadboard – the circuit is very similar to that in Experiment 4. PA6 is an output-only pin, and all the other lines from the micro-controller must be configured as outputs. Except for their common-cathode inputs, the two seven segment displays are wired in parallel (a to a, b to b, etc.):

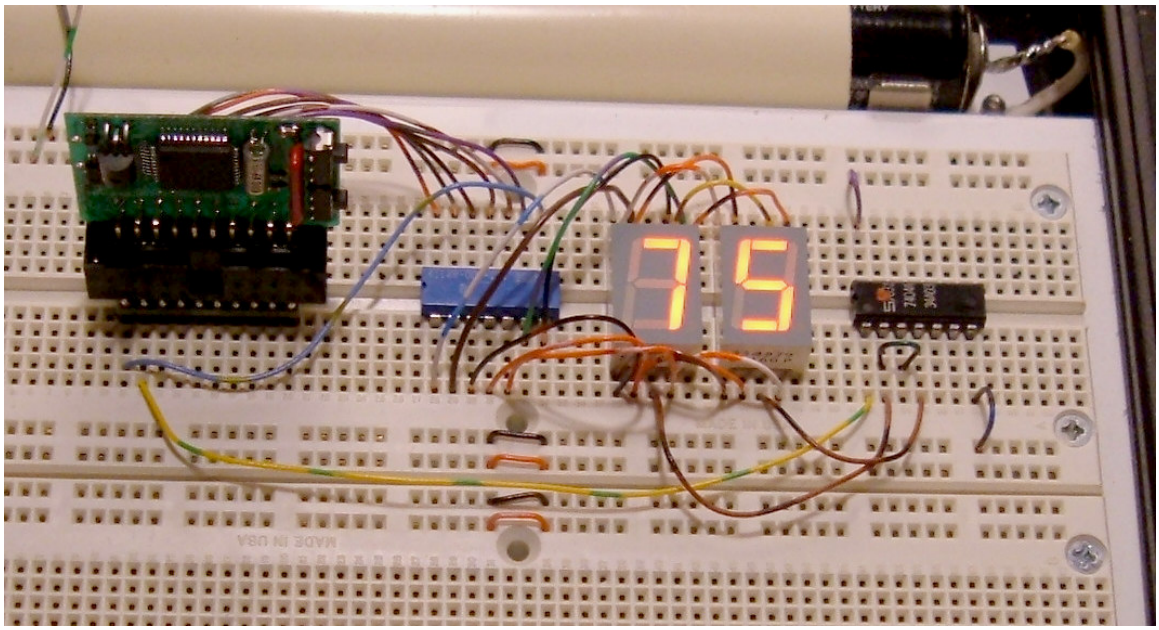


Both inverters must be included in the circuit. Inverter #1 buffers PA6 from having to sink up to 7 LED on-currents; without it the microcontroller would be damaged. Inverter #2 acts as a decoder to select which of the two digits is active according to the state of PA6.

When PA6 is high, data appearing on the a-g segment lines will be displayed on the seconds digit. When PA6 is low, the tenths-of-seconds digit will be on. You may wonder – how then may we display both digits at the same time? We don't. When rapidly displaying one digit and then the other (repeating this at about 100 cycles per second) the human eye perceives both digits to be on. Thus, at the cost of only one extra output line, we have added another digit to the display from that in Experiment 4.

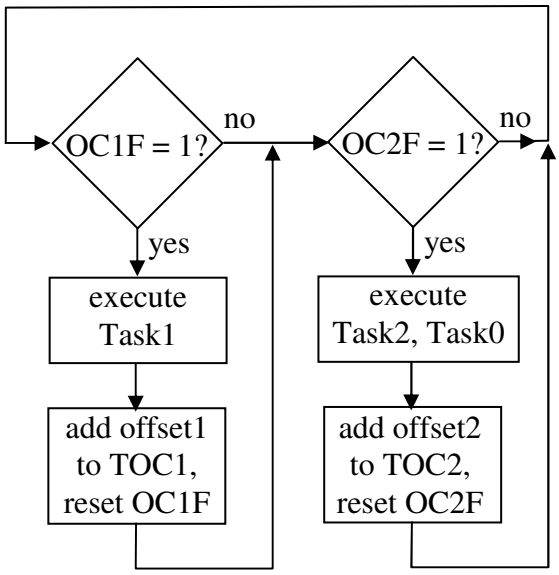
For your reference:





Sample circuit layout for Laboratory Experiment #5.

Software Design



In this experiment you will be using TCNT, TOC1 and TOC2 to execute certain tasks at different frequencies. Task0 and Task1 are almost identical to those in Experiment 4. Task2 is new to this experiment – its purpose is to change the digit being displayed every 1/200 sec:

Task1 pseudocode:
(executed every 0.1 sec)

```
Digit1 ← Digit1 + 1
if (Digit1 = 10)
    Digit1 ← 0
    Digit2 ← Digit2 + 1
    if (Digit2 = 10)
        Digit2 ← 0
    end if
end if
```

(initially Digit1 = 0, Digit2 = 0)

Task2 pseudocode:
(executed every 0.005 sec)

```
if (PA6 = 0)
    PA6 ← 1
    Digit ← Digit2
else
    PA6 ← 0
    Digit ← Digit1
end if
```

Task0: (executed every 0.005 sec)

Send 7-seg. data corresponding to memory value Digit to the output lines as in Experiment 4

(initially Digit = 0)

Debugging Procedure

1. Wire up the circuit as shown in the schematic diagram, except do not connect anything to PA6. Send either 0v or +5v to the input of Inverter #1. Execute the code you had for Experiment 4 (single-digit seconds counter), and depending on Inverter #1 input state you will see either the left or the right digit counting off seconds.

2. Now that you know that the hardware is correctly wired, hook up Inverter #1 input to PA6. Write the software excluding Task1; if done correctly your display should show two zeros.

3. Finally include Task1 (identical to Task1 in the previous experiment but with different variable names) in the code that executes every 0.1 sec. You should now see a 2-digit counter. If the digits are reversed then switch the wires to the two CC seven segment LED displays.

Here is what the main program will look like when Task0, Task1 and Task2 are written as procedures:

```
Program_Loop:

        ;check to see if TCNT has reached or surpassed TOC1 value
        BRCLR   TFLG1,X,$80,B0              ; check if OC1F is set
        JSR     Task1

        ;check to see if TCNT has reached or surpassed TOC2 value
B0:     BRCLR   TFLG1,X,$40,Program_Loop    ; check if OC2F is set
        JSR     Task2
        JSR     Task0
        JMP     Program_Loop
```

Common errors:

- Load your code beginning at $E000; if you begin at $FF00 you may run out of room and overwrite the interrupt vector table in EEPROM ($FFC0 to $FFFF).

- Instructions BRA or BSR (branch, call to subroutine) use relative addressing, and as such may only jump backward by -128 bytes and forward by +127 bytes of machine code. Try not to use these instructions unless speed is critical. Instead use JMP and JSR.

- Never use the BCLR instruction to reset an OCxF bit (output timer compare flag). Read the text and look and the hardware description given in Experiment 4 to understand why this doesn't work.