ECE 367 - Experiment #4
Seven Segment Display Interfacing and Timing

Spring 2006 Semester

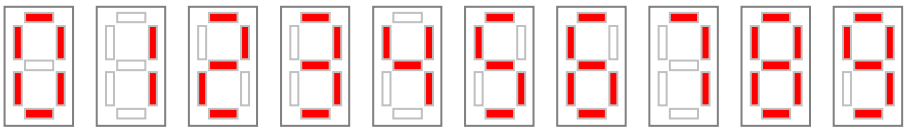
Introduction

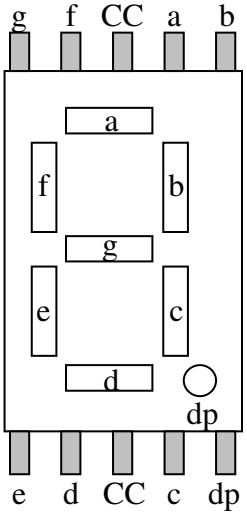
This experiment requires that you construct a circuit interfacing the MicroStamp11 module with a seven segment display, and write assembly language code to continuously display a decimal digit that increments exactly[1] once per second.

Required Hardware

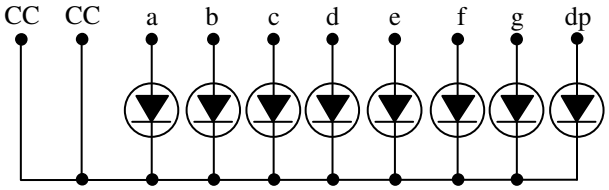
In addition to the MicroStamp11 module, this experiment requires one single-digit seven segment LED display and the 220Ω DIP resistors.

Here are the decimal digits 0 through 9 as displayed on a seven segment display:





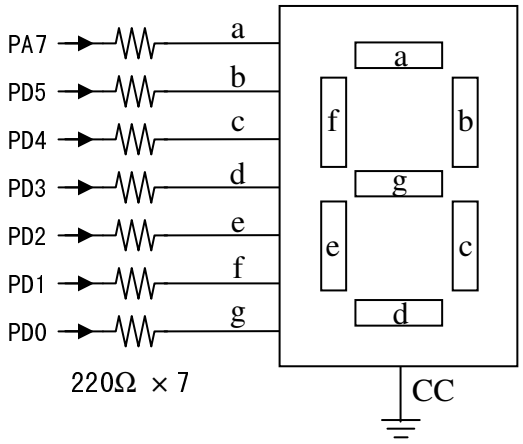
The pinout diagram on the left is for the Ledtech UC5641-11 (Jameco #334879) seven segment plus decimal point LED display that is in your parts kit. The two common cathode (CC) terminals are tied to all seven LEDs as shown below:

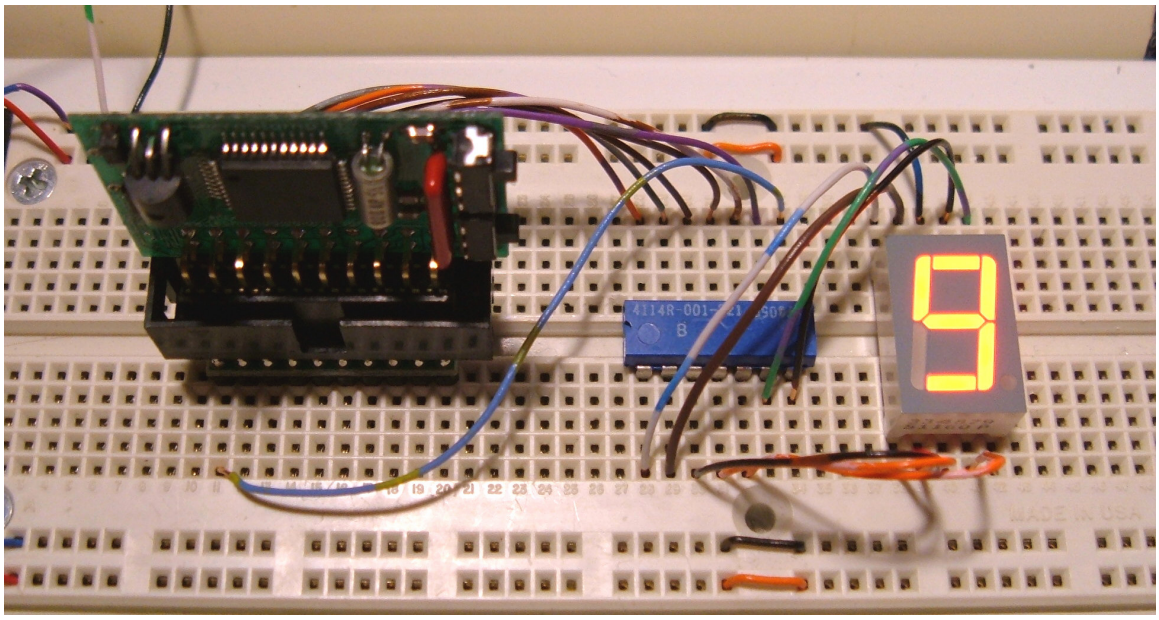


[1] as determined by the accuracy of the microcontroller's master clock crystal oscillator

Wiring Diagram

Build the following circuit on a solderless breadboard. Note that PA7 and all six bits in PortD are configured to operate as output lines:



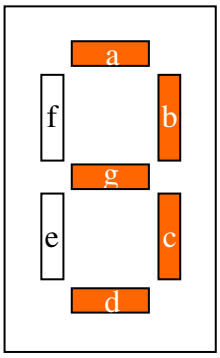


Sample circuit layout for Laboratory Experiment #4.

Software Design

First consider the task of displaying a decimal digit using seven segments. For example, the wiring diagram above tells us that we need to have the following output data on PA7, PD5 ... PD0 in order to display digit “3”:

Output line	logical state to display "3"	corresponding segment label
PA7	1	a
PD5	1	b
PD4	1	c
PD3	1	d
PD2	0	e
PD1	0	f
PD0	1	g



Assuming that PA7, PD5 ... PD0 have already been configured to be outputs, here are a few lines of code that will cause "3" to be displayed:

```
        LDAA    #$79            ; 7-segment code for "3"
        STAA    PortD,X         ;   sent to PD5...PD0
        BSET    PortA,X,$80     ; PA7<--1 (high bit of the 7-seg code)
```

(As usual, register X has been initialized to the starting address of the register block and PortA/PortD labels have been equated with their respective offset values.)

The easiest way to display the decimal digit whose value is in memory variable Digit, for example, is by using a sequence of if/then blocks such as this one:

```
Check_if_3:

        LDAA    Digit
        CMPA    #3              ; compare Digit to the value 3
        BNE     Check_if_4
        LDAA    #$79            ; send 7-seg. code for 3 to the output
        STAA    PortD,X
        BSET    PortA,X,$80
        JMP     Finished_Task

Check_if_4:
```

(etc.)

Incrementing the Displayed Digit Once Per Second

Now that you know how to show a decimal digit on a seven segment display device, next we will cover the timing aspects of this experiment. In previous experiments all delays were achieved using software loops, so it is natural to consider taking the same approach here. But we will take this opportunity to learn about a special subsystem inside the Motorola 68HC11 microcontroller – the hardware timer.

A software delay loop requires that your code do the counting, and nothing else gets done in the meantime. A hardware timer, on the other hand, does the counting for you while the CPU (being controlled by your program code) is free to do other things.

16 bit Timer Register TCNT

The 68HC11 hardware timer "TCNT" has 16 bits and is constantly counting up in binary (cycling through 2^{16} unique states). It is called a free-running counter because it is always running independently of what the CPU is doing. TCNT may not be initialized, cleared, nor may its counting process be stopped. It essentially functions as a read-only 16 bit register whose contents change every {1, 4, 8 or 16} master clock cycles, depending on how things are configured.

TCNT may be configured to count in one of four ways:

TCNT Prescaling Factor *N* :	Time it takes TCNT to increment once:	Time it takes TCNT to increment 2^{16} times:
1	0.5 usec = one clock period[2]	0.032768 sec
4	2.0 usec = four clock periods	0.131072 sec
8	4.0 usec = eight clock periods	0.262144 sec
16	8.0 usec = sixteen clock periods	0.524288 sec

For example, we can achieve 0.25 sec delay in these four ways:

- Wait for TCNT to increment 500,000 times when $N = 1$;
- Wait for TCNT to increment 125,000 times when $N = 4$;
- Wait for TCNT to increment 62,500 times when $N = 8$;
- Wait for TCNT to increment 31,250 times when $N = 16$.

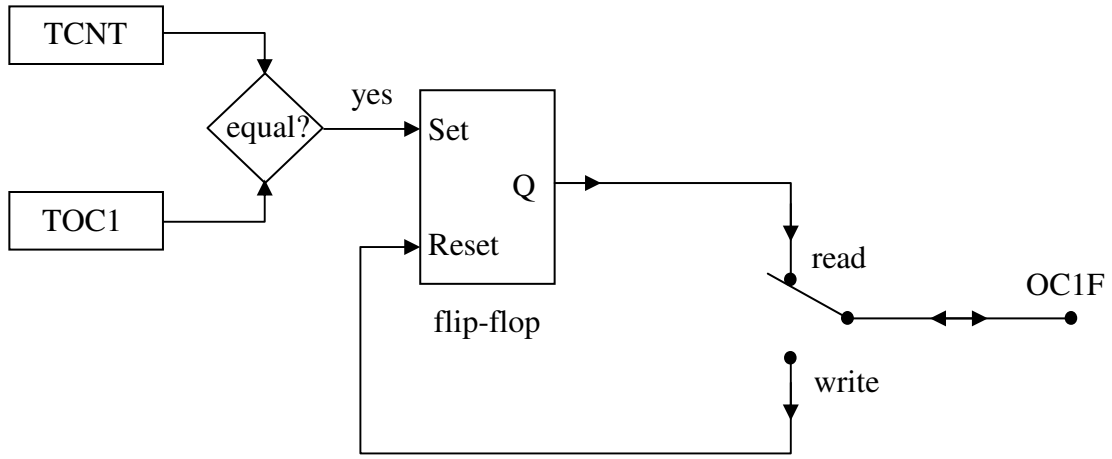
However, by selecting prescaling factor *N* so that the number of times TCNT needs to increment is less than $2^{16} = 65{,}536$ (the number of states in a complete count cycle), we greatly simplify the task of detecting when the desired delay has passed. Thus either N=8 or N=16 would be used to realize a 0.25 sec delay. To minimize microcontroller power consumption one should always slow down the TCNT counter as much as possible (choosing the largest possible *N*), so the choice of N=16 is best in the example above.

There also exist four Timer Output Compare registers TOC1–TOC4, whose 16-bit contents are constantly being compared to the TCNT counter state. When hardware detects that TCNT has incremented to a value equal to that stored in one of the TOC registers, then a corresponding flag is set.[3] For example, when TCNT = TOC1 then OC1F (Output Compare 1 Flag) becomes set.

[2] Since the MicroStamp11 that you have purchased has an internal clock ("E Clock") frequency = 2 MHz

[3] Also, a hardware interrupt may be triggered or an output pin may be made to change state. TCNT along with its output compare registers is one of the most useful subsystems on the microcontroller.

The following hardware diagram describes the interaction between TCNT, TOC1 and OC1F:



OC1F automatically becomes set when TOC1 = TCNT, and only a software write operation can reset it. One must write 1 to OC1F in order to reset it; writing 0 has no effect.

These steps are taken to realize 0.25 sec delay using the TCNT timer subsystem:

a) configure[4] prescaling factor $N = 16$;
b) read[5] the value of TCNT;
c) add 31,250 to this value and store it in TOC1[6];
d) clear OC1F[7];
e) wait until OC1F = 1 (polling this flag every so often while performing other tasks);
f) execute the task that has been waiting for the 0.25 sec delay to pass.

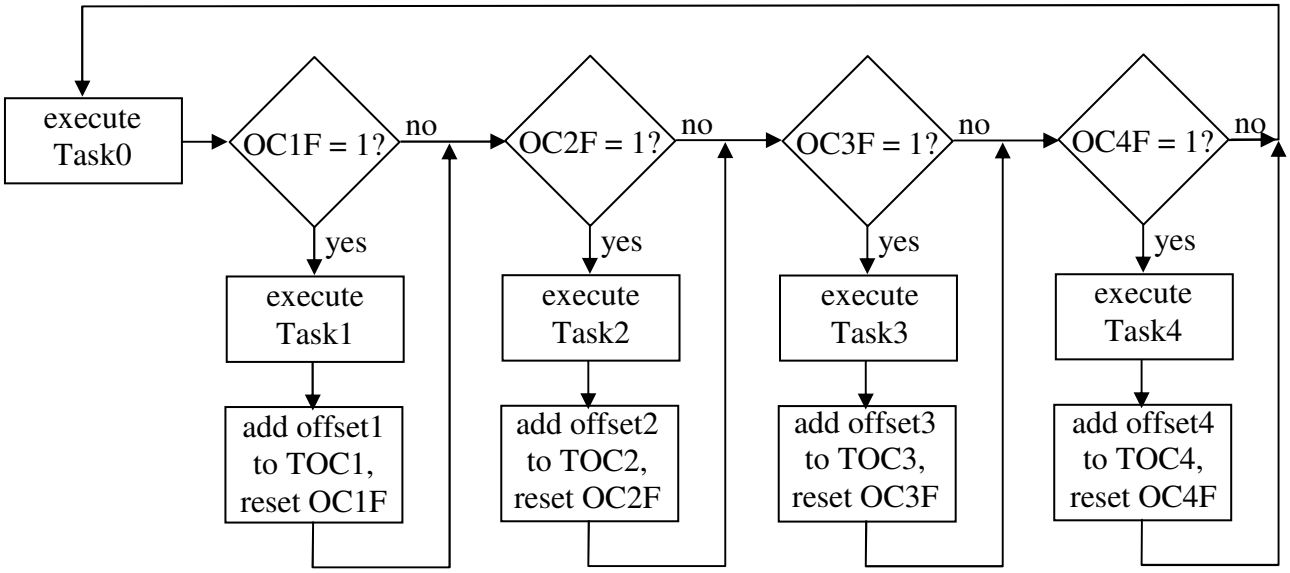
Usually the TCNT timer is used to trigger execution of a particular task at uniform increments of time, such as every 0.25 sec. In such case one would append the following steps to those above:

g) increment the value of TOC1 by 31,250;
h) jump to step (d)

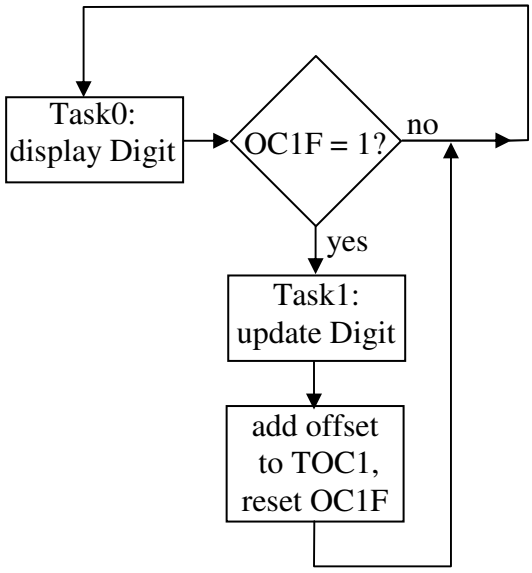
Answer this question: why is step (g) doing TOC1 ← TOC1 + 31,250 instead of TOC1 ← TCNT + 31,250 ?

[4] PR1, PR0 ←1 in TMSK2 register; this must be done within the first 64 clock cycles after power-on reset.
[5] TCNT is at addresses Regbas+[$0E,$0F]
[6] TOC1 is at addresses Regbas+[$16,$17]
[7] OC1F = bit 7 of TFLG1, found at Regbas+$23.

The following software flowchart shows how TCNT may be used to control the timing of up to five tasks at once: one running most of the time (Task0) and the others (Task1, Task2, Task3 and Task4) executing only at certain uniformly-spaced times[8]:



In this experiment you will be using TCNT and TOC1 to execute only two tasks. Here is a simplified flowchart:



Task0: Display memory value Digit on a 7-seg. LED display

Task1 pseudocode:
(executed every 0.1 sec)

```
Count ← Count + 1
if (Count = 10)
        Count ← 0
        Digit ← Digit + 1
        if (Digit = 10)
                Digit ← 0
        end if
end if
```

(initially Count = 0, Digit = 0)

[8] Assume that the time required to execute each task is much shorter than the delays administered by TCNT

The pseudocode description of Task1 shows the integer Count being incremented. When Count reaches 10 it is reset to 0, and then integer Digit is incremented. When Digit reaches 10 it is reset to 0. Thus Count is a decimal counter for tenths of a second and Digit is a decimal counter for seconds. We do this is because TCNT cycles too quickly (0.524 sec at its slowest pace) to directly realize 1.0 sec delay times.

Finally, here is some useful code that you may use for this lab:

1. Defining memory labels ($0040 - $00FF is RAM on the MicroStamp11):

```
Count    EQU   $0040          ; RAM byte address: tenth-of-sec counter
Digit    EQU   $0041          ; RAM byte address: displayed digit value
```

2. Configuring the ports and prescaling factor:

```
         LDAA   #$01          ; PR1,PR0 <-- 0,1
         STAA   TMSK2,X       ; (slow down TCNT by factor 4)
         BSET   PACTL,X,$80   ; configure PA7 as output
         BSET   DDRD,X,$3F    ; configure PD0-PD6 as outputs
```

3. Manipulating Count as part of Task1:

```
         INC    Count
         LDAA   Count
         CMPA   #10
         BNE    Label_1      ; if Count = 10,
         CLR    Count        ;  then reset it back to zero
```

(etc.)

4. Adding 1/10 sec offset to TOC1 (Incr must be equated with the offset value):

```
         LDD    TOC1,X       ; D <-- TOC1
         ADDD   #Incr        ; D <-- D + Incr
         STD    TOC1,X       ; TOC1 <-- D
         LDAA   #$80
         STAA   TFLG1,X      ; Clear the TCNT Output Compare 1 flag
```

Don't try to get everything working at once – begin by displaying the digit "8" on the 7 segments.[9] This will check both your circuit wiring and correctness of port configuration. Only then proceed to writing and debugging the timing algorithm. All timing must be done using TCNT – no software delays are allowed.

Demonstrate the working circuit to your T.A.

[9] Don't display "8" for very long as this microcontroller is not rated to drive that much current at once. To do it correctly we should buffer the output lines before driving the 7-seg. LED display (e.g. TTL inverters).