The assembly programs for the completed tasks can be found here

Introduction

In this experiment, we aim to extend the preliminary behavior of the P0.asm program provided. Namely, the goal is to generate five train pulses with different rise and fall (or high and low) times denoted by T_H and T_L respectively. However, prior to setting T_H and T_L, we must first configure the PIC18F420 chip such that the pulse train is generated at the RA2 bit of PORTA. The default setting from the P0.asm program had set RA4 as an input.

	RC7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
PORTA	0	0	0	0	0	1	0	0

Figure 1: Configuring of pin 2 of PORTA

In terms of programming, we apply this change in line 73 each program file (5 total).

```
63  ; This subroutine performs all initializations of variables and registers.
64
65  Initial
66          MOVLF   B'10001110',ADCON1  ;Enable PORTA & PORTE digital I/O pins
67          MOVLF   B'11100001',TRISA  ;Set I/O for PORTA 0 = output, 1 = input
68          MOVLF   B'11011100',TRISB  ;Set I/O for PORTB
69          MOVLF   B'11010000',TRISC  ;Set I/O for PORTC
70          MOVLF   B'00001111',TRISD  ;Set I/O for PORTD
71          MOVLF   B'00000000',TRISE  ;Set I/O for PORTE
72          MOVLF   B'10001000',T0CON  ;Set up Timer0 for a looptime of 10 ms;  bit7=1 enables timer; bit3=1 bypass prescaler
73          MOVLF   B'00000100',PORTA  ;Turn off all four LEDs driven from PORTA ; See pin diagrams of Page 5 in DataSheet
74
75          MOVLF   B'11111111',TMR0H ;ADDED by AC
76          MOVLF   B'00000000',TMR0L ;ADDED by AC
77          MOVLF   B'00000010',ALIVECNT ;ADDED by AC
78  return
```

Figure 2: Initialization of asm program

In addition, we replace instances of RA4 with RA2 (because pin RA4 is not being used) in the `BlinkAlive` subroutine which is renamed to `PulseTrainGen` (see below) for the purposes of this experiment.

```
;;;;;;; Mainline program ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Mainline
        rcall   Initial            ;Initialize everything
Loop
        ;btg   PORTC,RC2            ;Toggle pin, to support measuring loop time (Not needed)
        rcall   PulseTrainGen       ;Subroutine
        rcall   LoopTime
        bra   Loop
```

Figure 3: Mainline Program

Pulse Train Generation

To generate a pulse train, we must set the T_H and T_L values which are already given. We refer to the P0.asm program whose purpose was to toggle an LED where the action of toggling is simply a mechanism to generating pulses. In most of the cases, to achieve the required specifications, we must simply modify the values of the `Bignum` and `ALIVECNT` variables. Their significance was described in detail in experiment I. Recall that their values are defined by the following expressions:

```
Bignum = 65536 – x + 12 + 2

ALIVECNT = C
```

Where `x` is the number of cycles to be removed to achieve some T_H value and `C` is the number of cycles we wish to perform. The value of `x` is calculated as such:

$$x = \frac{T_H}{0.4\mu s}$$

Recall that the instructions `bsf` and `bcf` are responsible for toggling 'ON' and 'OFF' respectively. With the `bsf` and `bcf` instructions swapped in the P0.asm program, the number of cycles needed to set a certain duty cycle can be determined by:

$$\frac{Duty\ Cycle\ (\%)}{100} = \frac{1}{C} \rightarrow C = \frac{100}{Duty\ Cycle\ (\%)}$$

It should also be noted that `C` can only be an integer. Lastly, the duty cycle is given as a ratio of the rise time to the sum of the rise and fall times or mathematically speaking:

$$Duty\ Cycle = \frac{T_H}{T_H + T_L} \times 100\%$$

Generally speaking, we have to set the appropriate values for `Bignum` and in the `PulseTrainGen` subroutine which can be summarized by the following pseudocode:

```
PulseTrainGen
      bcf PORTA,RA2
      dec ALIVECNT,F
      bnz PTGend
      MOVLF C, ALIVECNT
      bsf PORTA,RA2
PTGend
      return
      end
```

Task 1: $T_H = T_L = 0.1ms$

The given rise and fall values require that the pulse train toggle at equal rates. Namely, to achieve a rise value of 0.1ms, the value of `Bignum` is modified such that number of cycles `x` to be removed is

$$\mathtt{x} = \frac{0.1ms}{0.4\mu s} = 250 \text{ cylces}$$

Since, the fall time is the same as the rise time, this implies that the pulse train has a duty cycle of 50%. Hence, reinitialization value of the `ALIVECNT` is set to 2 (i.e. `C = 2`) in the `PulseTrainGen` subroutine. It is important to note that it does not matter whether the RA2 pin is toggled ‘ON’ or ‘OFF’ first since the duty cycle is 50% (i.e., the pulse train is symmetrical). So, swapping the the `bsf` and `bcf` instructions is optional. To verify the correctness of this design, we will simulate it using the MPLAB IDE’s *Logic Analyzer*.

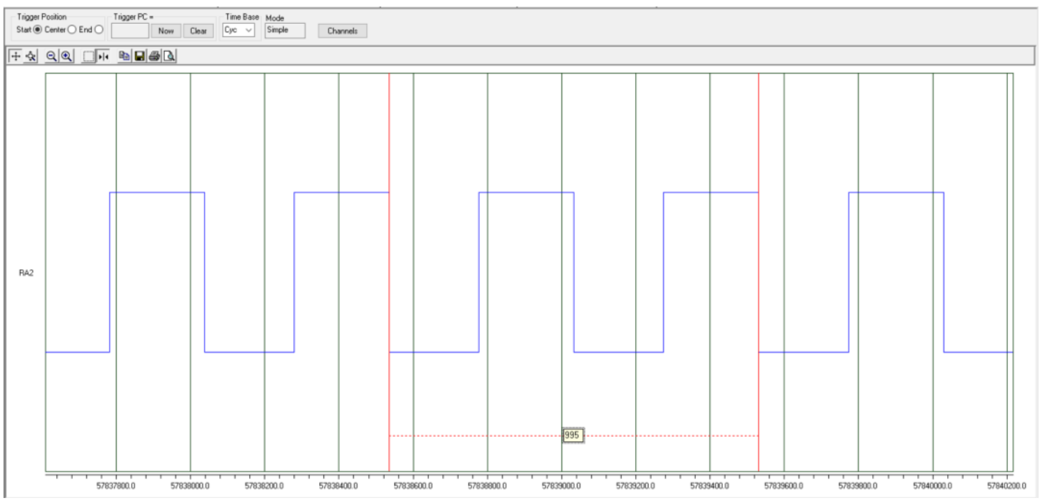

Figure 4: Waveform output of task 1

It is clear the output at the RA2 pin is indeed a pulse train with a duty cycle of 50% since both rise and fall times are equal.

Task 2: $T_H = 0.1ms, T_L = 0.3ms$

Given the T_H and T_L values, the expected duty cycle is 25%. That is, we require that the pulse train rises only once out of four cycles. Hence the value of `ALIVECNT` must be reinitialized to 4 (i.e. `C = 4`). The `Bignum` value required the removal of $\mathtt{x} = \frac{0.1ms}{0.4\mu s} = 250$ cylces. This time around, the `bsf` and `bcf` instructions are swapped. In doing so, we ensure that the output at pin RA2 goes HIGH only once since the `bsf` instruction sets (i.e. toggles ‘ON’) is executed just once every `C` cycles. Again, we verify the correctness of the program via the *Logic Analyzer*.

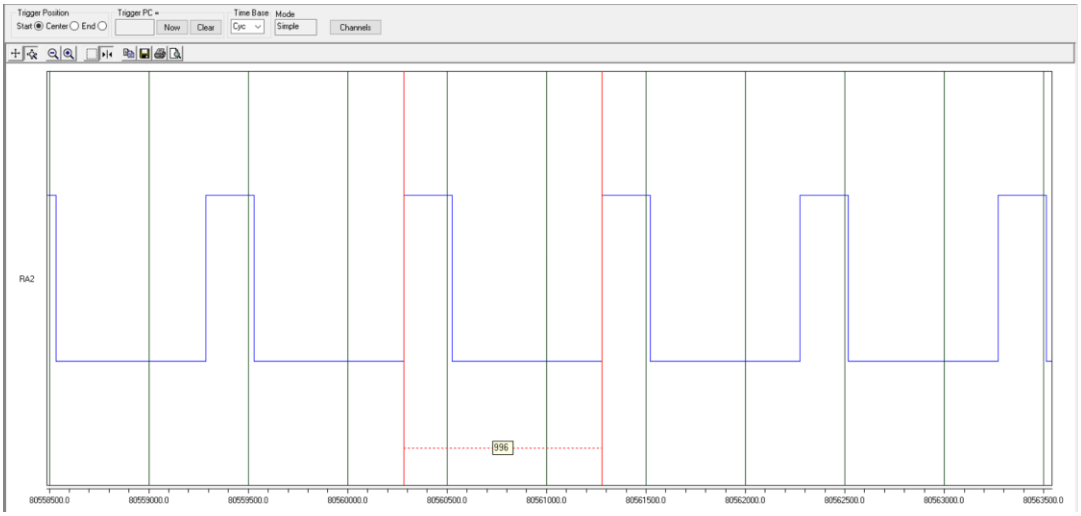

Figure 5: Waveform output of task 2

Task 3: $\boldsymbol{T_H = T_L = 0.2ms}$

Similar to task 1, the given rise and fall values yield a duty cycle of 50%. The `ALIVECNT` variable is once again set to 2 (i.e. `C = 2`). The `Bignum` value required the removal of `x` $= \frac{0.2ms}{0.4\mu s} = 500$ cylces. With these modifications, the code for task3.asm is not much different from task1.asm.

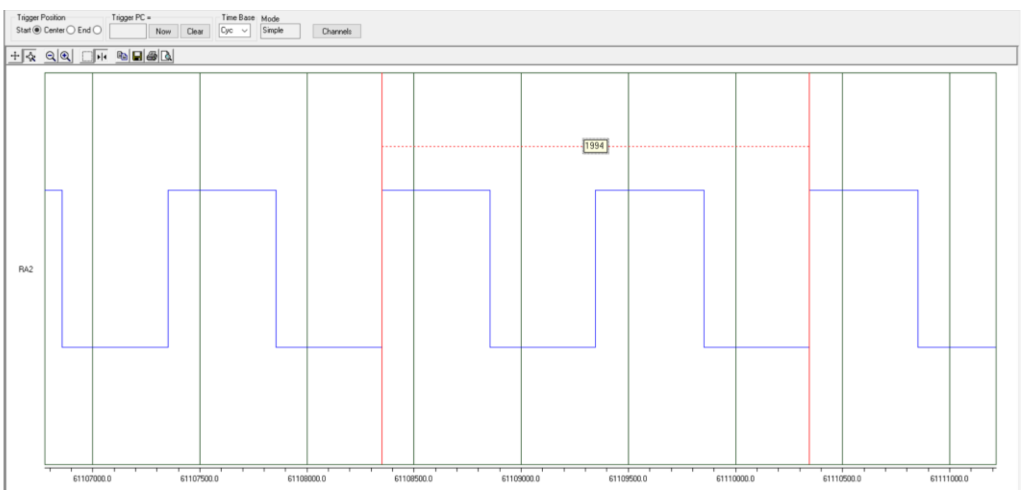

Figure 6: Waveform output of task 3

Task 4: $\boldsymbol{T_H = 0.3ms,\ T_L = 0.2ms}$

The T_H and T_L values set the pulse train's duty cycle to 60%. Unlike tasks 1-3, there is no integer `C` value that be assigned to the `ALIVECNT` variable since `C` $= \frac{100}{60\%} = \frac{5}{3}$. Thus, another method must be applied to achieve this duty cycle. While it the LoopTime subroutine could not be called more than once. However, I could not realize this task without doing so. Hence, the `PulseTrainGen` subroutine was modified by removing the dependence of the `ALIVECNT` variable and just toggling pin RA2 manually via the `bsf` and `bcf` instructions in a certain sequence. Specifically, the subroutine was written in the following way:

```
;;;;;;; PulseTrainGen subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This subroutine generates a pulse train output at pin RA2

PulseTrainGen
        bsf PORTA,RA2
        rcall LoopTime
        bsf PORTA,RA2
        rcall LoopTime
        bsf PORTA,RA2
        rcall LoopTime
        bcf PORTA,RA2
        rcall LoopTime

        return

        end
```

Figure 7: PusleTrainGen Subroutine for Task4

We verify that the duty cycle is indeed 60% by making the necessary measurements. Given the output below, the duty cycle is $\frac{2556}{2556+1504} \times 100 = 62\%$ implying some error.

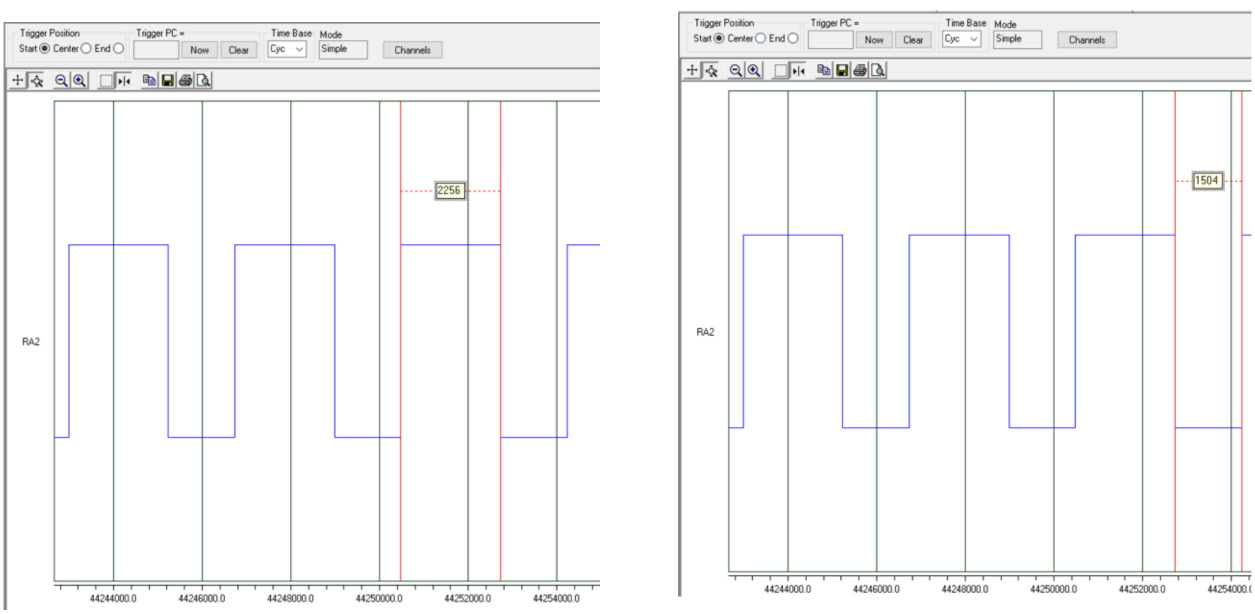

Figure 8: Waveform output for Task4 (Measuring duty cycle)

Task 5: $T_H = 0.25ms, T_L = 0.35ms$

Similar to task 4 above, there is no integer value C that can be assigned to ALIVECNT that will yield as duty cycle of $\approx 42\%$. Using the same method for the previous task, the subroutine was defined in the following way:

```
;;;;;;; PulseTrainGen subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This subroutine generates a pulse train output at pin RA2

PulseTrainGen
        bsf   PORTA,RA2
        rcall LoopTime
        bsf   PORTA,RA2
        rcall LoopTime
        bsf   PORTA,RA2
        rcall LoopTime
        bcf   PORTA,RA2
        rcall LoopTime
        bcf   PORTA,RA2
        rcall LoopTime
        bcf   PORTA,RA2
        rcall LoopTime

        return

        end
```

Figure 9: PusleTrainGen Subroutine for Task4

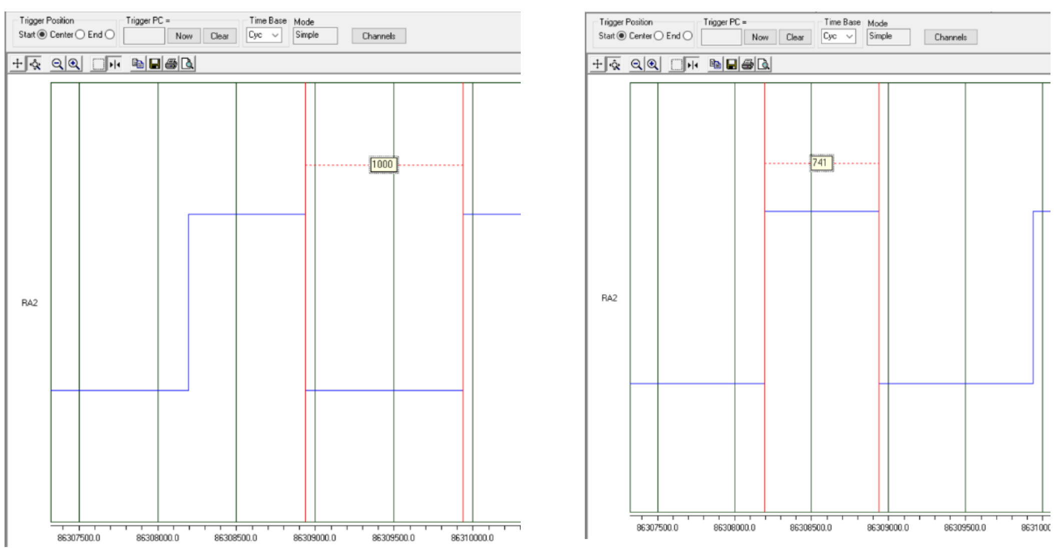

Figure 10: Waveform output for Task5 (Measuring Duty Cycle)

We verify that the duty cycle is about 42% with the given output above. The duty cycle is $\frac{741}{741+1000} \times 100 = 42.5\%$.

Conclusion:

This experiment has enabled us to generate pulse trains of various duty cycles. It challenged us to derive different methods to achieve the specifications of each task knowing that one method would not be able to satisfy all tasks. Needless to say, the approach for the last to tasks was rather crude and can be optimized.