Introduction

The objective of this experiment is to generate sequence of pulse trains based on predefined behaviors. In previous labs, pulses were created utilizing the `bcf` and `bsf` instructions to set and clear bits. This is considered a rather brute force approach and often leads to significantly longer code. Hence, there must exist an optimized approach to generating train pulses. This is what this report aims to demonstrate.

Task 1

The table below defines the sequence behaviors for each channel or each pin in terms of the microcontroller. In each task, the sequence behavior for channels 1-4 will correspond pins RC0 to RC3 of PORTC respectively. The expected waveform output is shown in figure 1.

CH. 4	CH. 3	CH. 2	CH. 1
0	0	0	1
0	0	1	0
0	1	0	0
1	0	0	0

Table 1: Pulse train behavior sequence 1

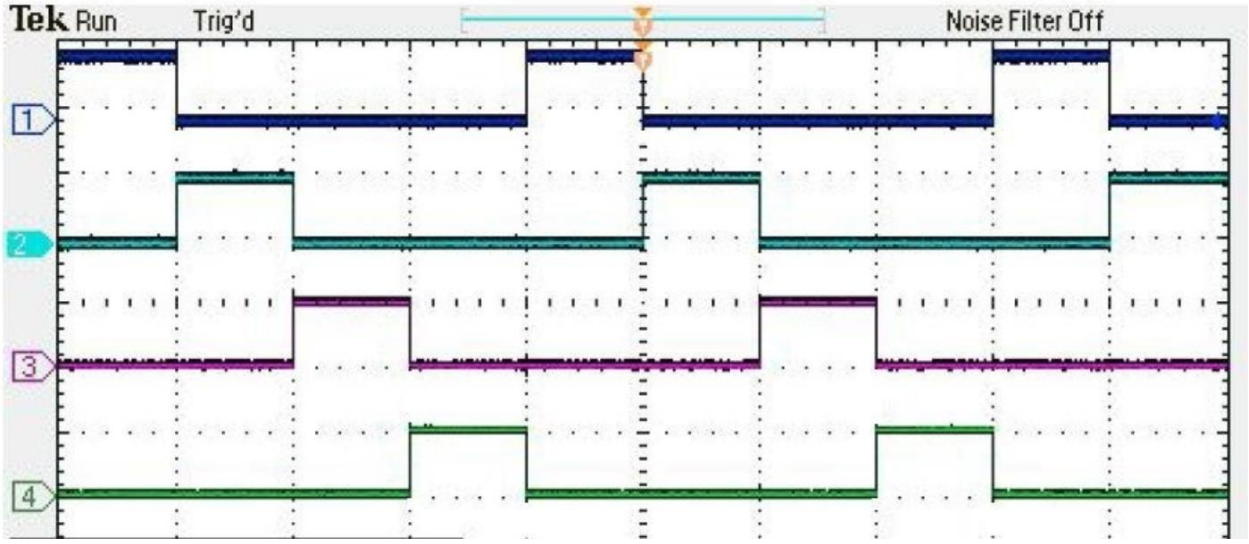

Figure 1: Expected oscilloscope waveforms for table 1

The following code on the left shows a brute force approach to generate the first pulse train sequence. The code on the right achieves the same with significantly less code by making use of rotations.

```
PulseTrainGen
    bsf PORTC,RC0 ;ON
    bcf PORTC,RC1 ;OFF
    bcf PORTC,RC2 ;OFF
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bsf PORTC,RC1 ;ON
    bcf PORTC,RC2 ;OFF
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bcf PORTC,RC1 ;OFF
    bsf PORTC,RC2 ;ON
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bcf PORTC,RC1 ;OFF
    bcf PORTC,RC2 ;OFF
    bsf PORTC,RC3 ;ON
    rcall LoopTime ;delay

    return
```

```
PulseTrainGen

    MOVLF B'00000001', PORTC
    MOVLF B'00000100', Count_Step ; Set count to 4
    PT
        rlncf PORTC, F
        decf Count_Step, F
        bnz PT
return                                  ; End of Subroutine
```

Figure 2: Assembly code (two approaches) to generate sequence in table 1

To verify the correctness of the assembly code above, we will simulate using MPLAB's Logic Analyzer, add the necessary pins to observe, and run the simulation to produce the results below.

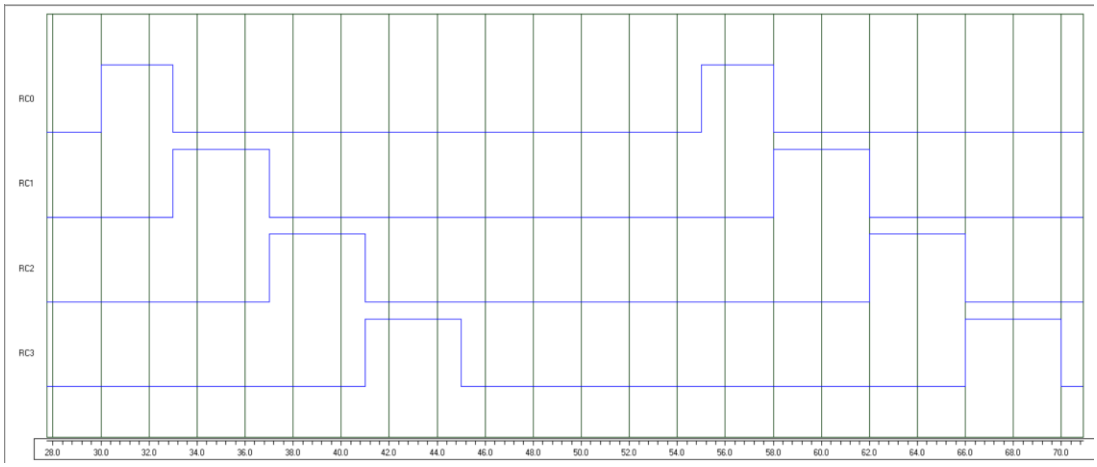

Figure 3: Waveform outputs for sequence 1

Task 2

The next sequence is given by table 2 below along with the expected output in figure 4.

CH. 4	CH. 3	CH. 2	CH. 1
0	0	1	1
0	1	1	0
1	1	0	0
1	0	0	1

Table 2: Pulse train behavior sequence 2

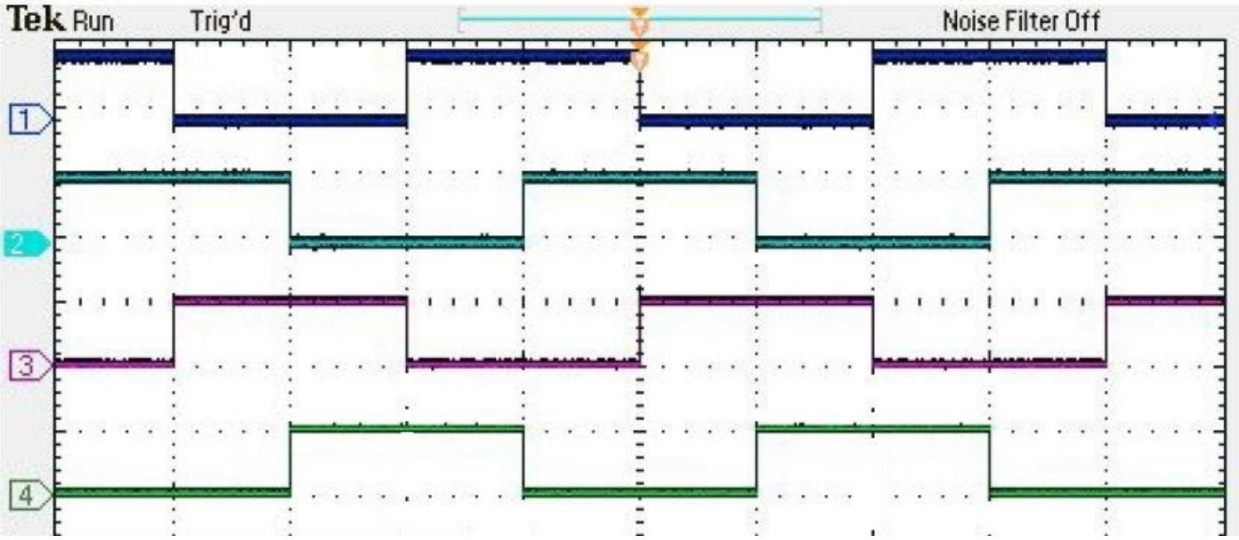

Figure 4: Expected oscilloscope waveforms for table 2

Again, the following code on the left shows a brute force approach to generate the first pulse train sequence. The code on the right achieves the same with less code by making use of three rotations from the initial value of PORTC (i.e. 00000011) followed by assigning the final value of sequence manually.

```
PulseTrainGen
    bsf PORTC,RC0 ;ON
    bsf PORTC,RC1 ;ON
    bcf PORTC,RC2 ;OFF
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bsf PORTC,RC1 ;ON
    bsf PORTC,RC2 ;ON
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bcf PORTC,RC1 ;OFF
    bsf PORTC,RC2 ;ON
    bsf PORTC,RC3 ;ON
    rcall LoopTime ;delay
    bsf PORTC,RC0 ;ON
    bcf PORTC,RC1 ;OFF
    bcf PORTC,RC2 ;OFF
    bsf PORTC,RC3 ;ON
    rcall LoopTime ;delay

    return
```

```
PulseTrainGen

    MOVLF B'00000011', PORTC
    MOVLF B'00000010', Count_Step
    PT
        rlncf PORTC, F
        decf Count_Step, F
        bnz PT
    MOVLF B'00001001', PORTC

return                                  ; End of Subroutine
```

Figure 5: Assembly code (two approaches) to generate sequence in table 2

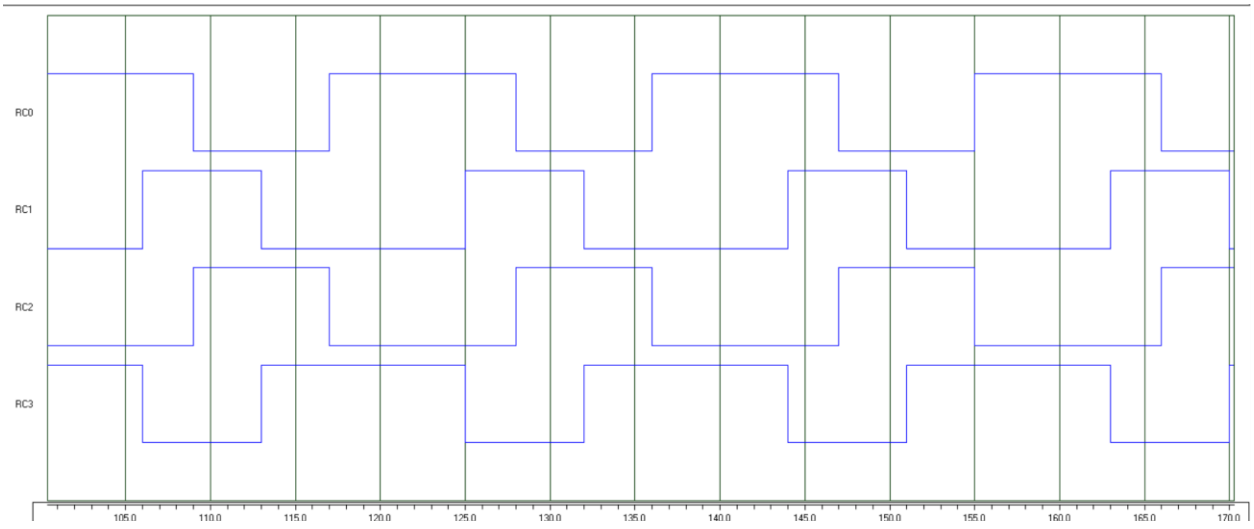

Figure 6: Simulation waveform outputs for sequence 2

Task 3

The next sequence is given by table 3 below along with the expected output in figure 7.

CH. 4	CH. 3	CH. 2	CH. 1
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	0	0
1	1	0	0
1	0	0	0
1	0	0	1

Table 3: Pulse train behavior sequence 3

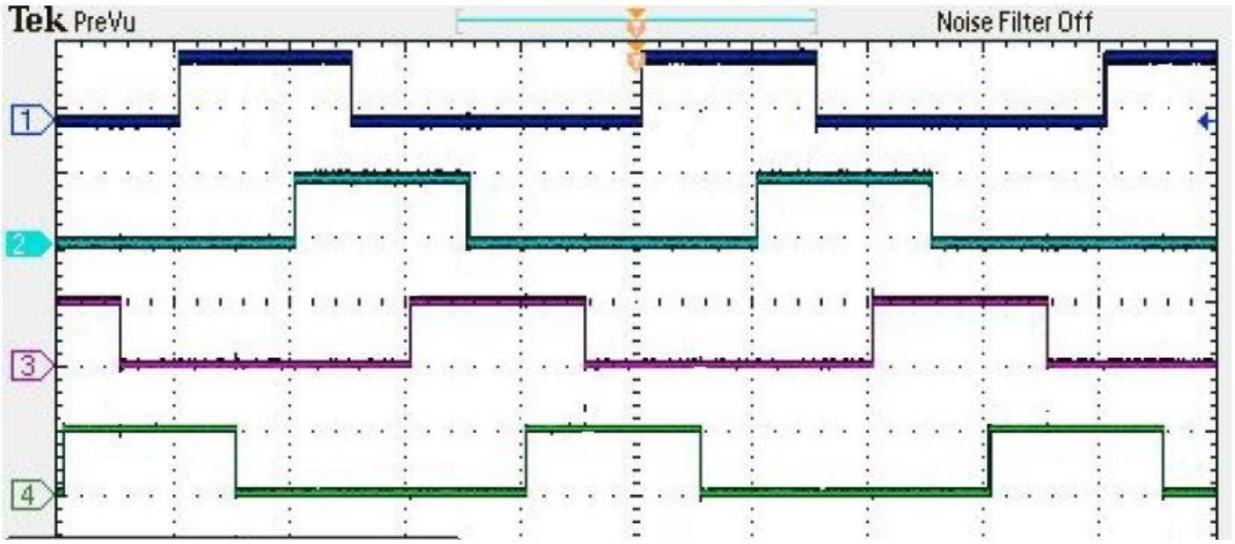

Figure 7: Expected oscilloscope waveforms for table 3

Initially, the sequence can be divided into two sub sequences. The first sub sequence occurs in steps 1, 3, 5, and 7. This is simply rotating the binary sequence 0001 one bit to the left. The second sub sequence occurs in steps 2, 4, 6, and 8. This is also another simple rotation of the binary sequence 0011 one bit to the left. These two sub sequences were implemented in previous tasks 1 and 2. However, generating the first sub sequence followed by the second subsequence does not produce the expected waveforms in figure 7 (*unless it's suppose too*). As a result, a brute force approach was utilized to generate the expected waveforms.

```
PulseTrainGen
    bsf PORTC,RC0 ;ON
    bcf PORTC,RC1 ;OFF
    bcf PORTC,RC2 ;OFF
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bsf PORTC,RC0 ;ON
    bsf PORTC,RC1 ;ON
    bcf PORTC,RC2 ;OFF
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bsf PORTC,RC1 ;ON
    bcf PORTC,RC2 ;OFF
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bsf PORTC,RC1 ;ON
    bsf PORTC,RC2 ;ON
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bcf PORTC,RC1 ;OFF
    bsf PORTC,RC2 ;ON
    bcf PORTC,RC3 ;OFF
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bcf PORTC,RC1 ;OFF
    bsf PORTC,RC2 ;ON
    bsf PORTC,RC3 ;ON
    rcall LoopTime ;delay
    bcf PORTC,RC0 ;OFF
    bcf PORTC,RC1 ;OFF
    bcf PORTC,RC2 ;OFF
    bsf PORTC,RC3 ;ON
    rcall LoopTime ;delay
    bsf PORTC,RC0 ;ON
    bcf PORTC,RC1 ;OFF
    bcf PORTC,RC2 ;OFF
    bsf PORTC,RC3 ;ON
    rcall LoopTime ;delay

    return
```

Figure 8: Assembly code to generate sequence in table 3 (brute force approach)

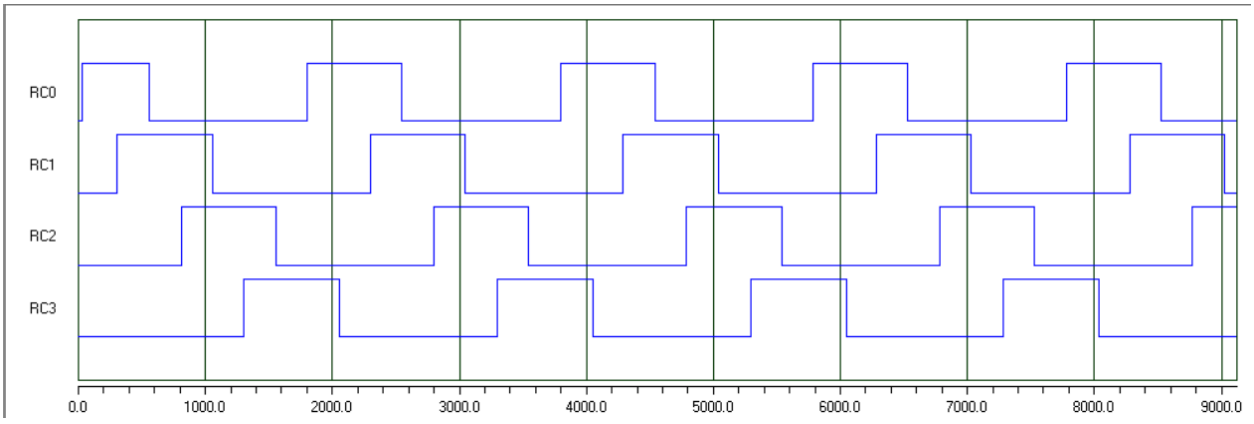

Figure 9: Simulation waveform outputs for sequence 3

Task 4

The next sequence is given by table 2 below along with the expected output in figure 10.

CH. 4	CH. 3	CH. 2	CH. 1
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	0
0	0	0	1
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

Table 4: Pulse train behavior sequence 4

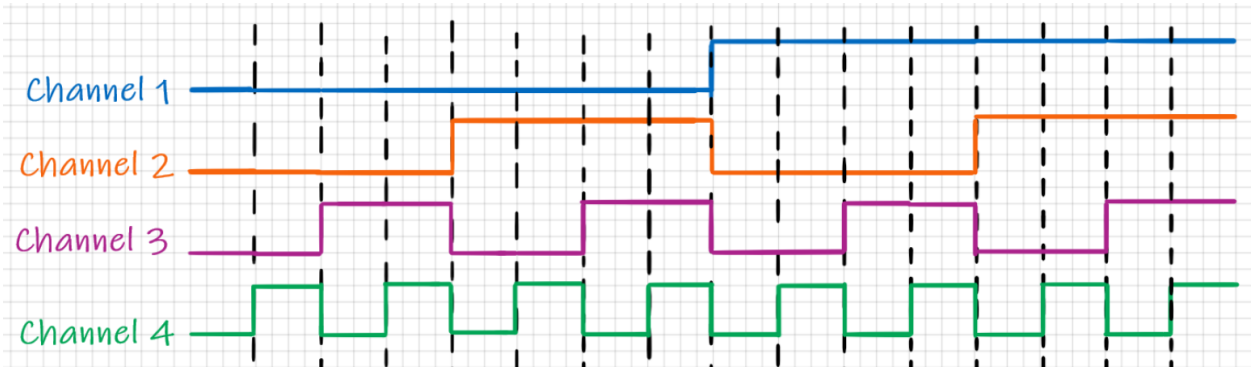

Figure 10: Expected oscilloscope waveforms for table 4

To implement this train pulse, the order of the columns will be arranged such that channel 1 sits at the first column, channel 2 in the second, and channels 3 and 4 in the third and fourth column respectively. In this way the sequence is just counting from 0 to 15 in binary. Utilizing the `incf` instruction, the assembly code to achieve this can be done quite easily. One can imagine attempting to set each bit using the `bcf` and `bsf` instruction to achieve the same results is not only tedious but also not practical.

```
PulseTrainGen
    MOVLF B'00000000', PORTC ; initialize the sequence
    MOVLF B'00010000', Count_Step
    PT
        incf PORTC, F
        decf Count_Step, F
        bnz PT

    return                                  ; End of Subroutine
```

Figure 11: Assembly code to generate sequence in table 4

The output waveforms are given below.

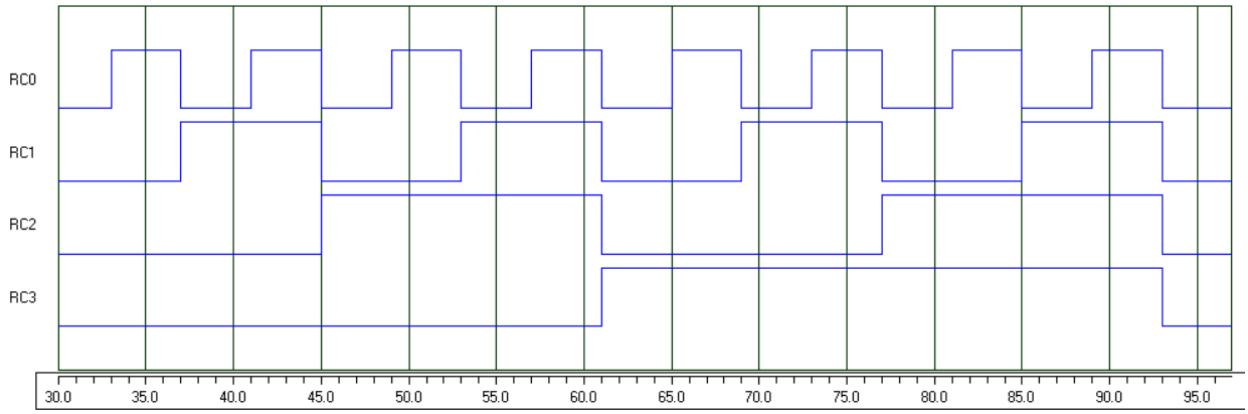

Figure 12: Simulation waveform outputs for sequence 4

Conclusion

Overall, we've studied the importance of optimization by generating train pulses. In utilizing both naïve and optimized approaches, it became clear that using rotations and incrementing on values instead of setting and clearing bits produced much less and cleaner code. Doing so also potentially produced faster code and a lower runtime. The concept of optimization is crucial in industry where software and hardware are constantly being worked on to be made faster, more compact, and more secure.