Digital Signal Processing: FIR Filter Implementation

Shan Qureshi

Contents

1 Introduction

Objective: This experiment is designed to show the digital signal processing capabilities of the PIC18F4520 microcontroller through the implementation of a Finite Impulse Response (FIR) filter.

1.1 Specific Tasks

We are to design the FIR Filter H(z) shown in the equation below

$$H(z) = \frac{1 + z^{-1} + z^{-2} + z^{-3}}{4} \tag{1}$$

To begin the implementation, we incorporate the AC/DC converter from Part 2 of **Experiment 2 - Data Acquisition: Analog-to-Digital and Digital-to-Analog Conversions**.

2 Theory

2.1 Mainline

The transfer function is made up of multiple parts: a Linear Memory Buffer, an Adder, and a Divider. These parts are between the AC and DC conversion.

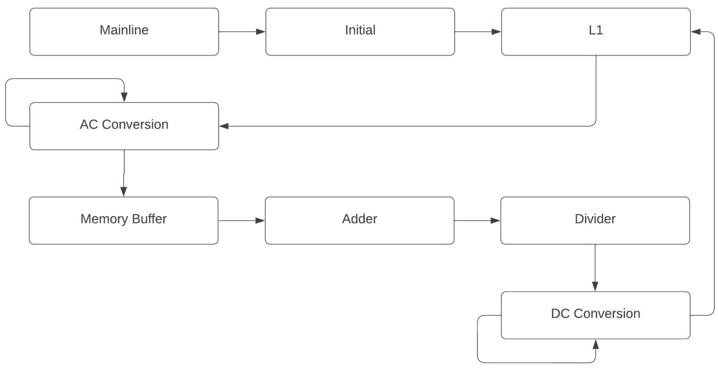


Figure 1: Structure of the code.

Fig.1 above shows how the AC signal is processed and converted into DC.

2.2 Linear Memory Buffer

The Linear Memory Buffer is used to store the terms $\{z^0,\ z^{-1},\ z^{-2},\ z^{-3}\}$. We know that the results of the AC Conversion are stored within the registers **ADRESL** and **ADRESH**, where L denotes the lower 8-bits and H denotes the higher 8-bits. The variables **VnL** and **VnH** ($n = [0, 1, 2, 3]$) are used to retain the values inside **ADRESL** and **ADRESH** and the next three values after.

Our initial approach to this problem was to store the initial term first and then move it down from $n = 0$ to $n = 3$ as shown below in **Fig.2**

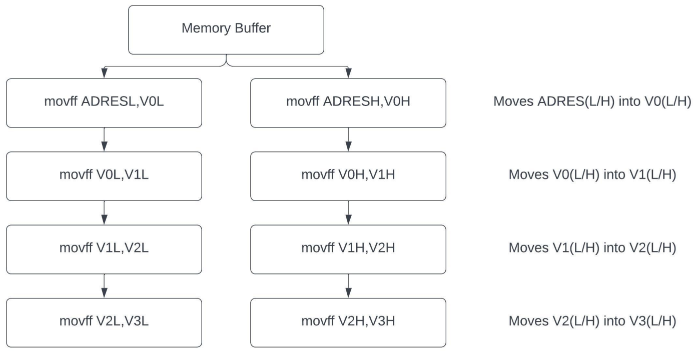


Figure 2: First attempt at the Linear Memory Buffer.

However, the problem that arises is that by the end of the Memory Buffer, we have z^{-0} stored inside **V3L** and **V3H** while all other registers are empty. To solve this problem, we reversed how the storage was done.

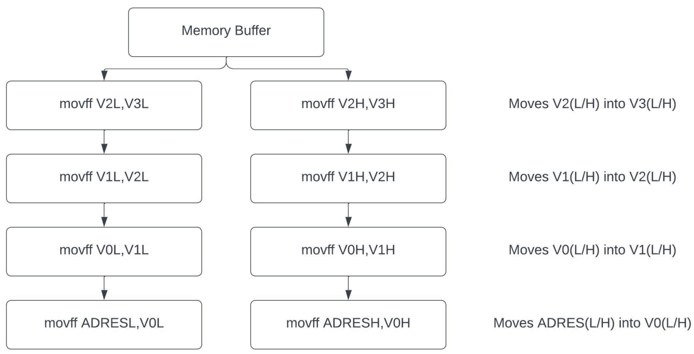


Figure 3: Structure of the Linear Memory Buffer.

Fig.3 above solves the problem of **Fig.2** by making all **VnL** and **VnH** independent of each other until the **next** iteration, while storing each register with its necessary time shift.

2.3 Adder

The Linear Memory buffer stores the terms $\{z^0, z^{-1}, z^{-2}, z^{-3}\}$, and the Adder sums them up

$$\sum_{n=0}^{3} z^n = z^0 + z^{-1} + z^{-2} + z^{-3} \tag{2}$$

We introduce the variable **SumTotal(L/H)**. As the name suggests, it will hold the sum of the lower and upper 8-bits.

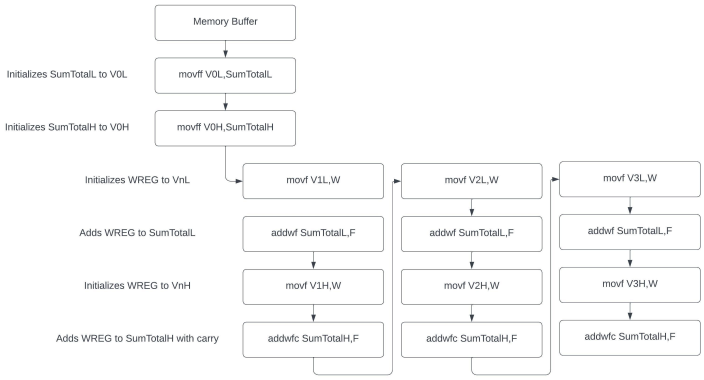


Figure 4: Structure of the Adder.

Although simple, a mistake that can occur is the existence of a carry. This carry occurs when adding two registers results in an overflow. To solve this problem we use the instruction *addwfc*. This instruction adds both carry bit and **WREG** to **SumTotalH** as shown in **Fig.4**.

The order in which we add the registers is crucial due to this carry. It must be in the order of L, H, L, H, L, H to preserve the respective carry bit.

2.4 Divider

To divide by two in binary is to shift right, as shown below

$$00001110_2 = 14_{10}$$
$$00000111_2 = 7_{10}$$

However, there is no shift instruction with the PIC18F4520[1] microcontroller. To solve for this, we use logical operators.

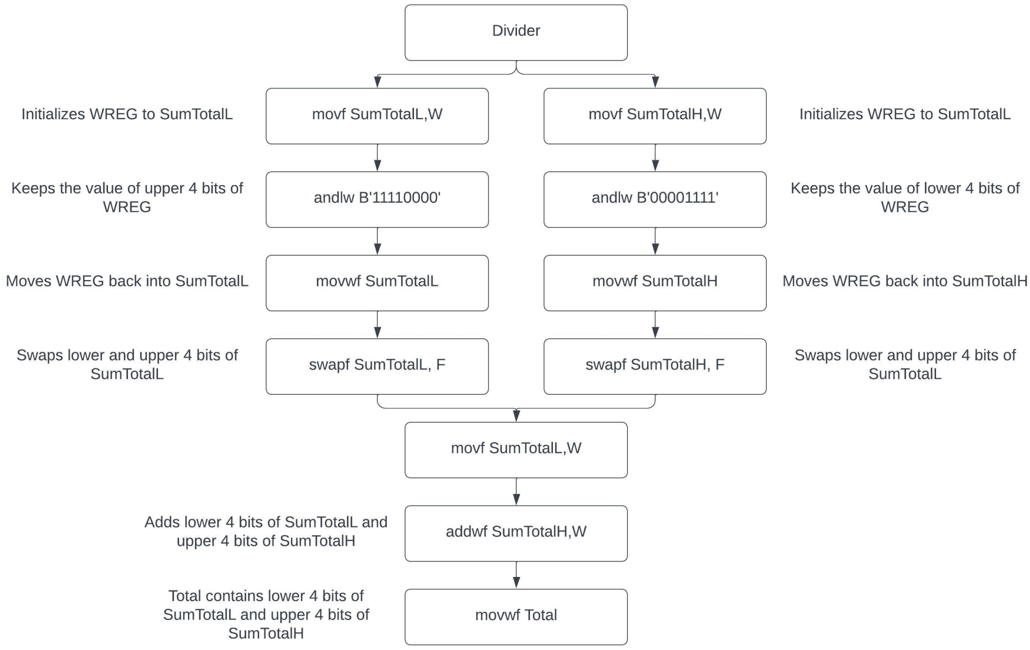


Figure 5: Structure of the Divider

To divide by four would be two shifts. However, we must consider how many times we're adding with respect to **VnL** and **VnH**. At the end of the Adder results in a 12-bit number. We can only store a maximum of 8-bits inside a register. This would mean instead of shifting twice, we have to shift four times to turn a 12-bit into an 8-bit.

Let's assume **SumTotalL** $= a_7a_6a_5a_4a_3a_2a_1a_0$ and **SumTotalH** $= b_7b_6b_5b_4b_3b_2b_1b_0$. If we want to shift these numbers four times we get

$$b_3b_2b_1b_0a_7a_6a_5a_4$$

If we use the AND operator with the numbers $B_1 = 11110000$ and $B_2 = 00001111$ respectively, we get

$$\textbf{SumTotalL} \text{ AND } B_1 = \textbf{TotalL} = a_7a_6a_5a_4 0000 \tag{3}$$

$$\textbf{SumTotalH} \text{ AND } B_2 = \textbf{TotalH} = 0000 b_3b_2b_1b_0 \tag{4}$$

If we swap the upper and lower 4 bits of the number we obtain

$$\textbf{TotalL} = 0000 a_7a_6a_5a_4 \tag{5}$$

$$\textbf{TotalH} = b_3b_2b_1b_0 0000 \tag{6}$$

Combining **Eq.5** and **Eq.6** results in

$$\mathbf{Total} = \mathbf{TotalH} + \mathbf{TotalL} = b_3 b_2 b_1 b_0 a_7 a_6 a_5 a_4 \tag{7}$$

This result is the same as shifting by four.

3 Results

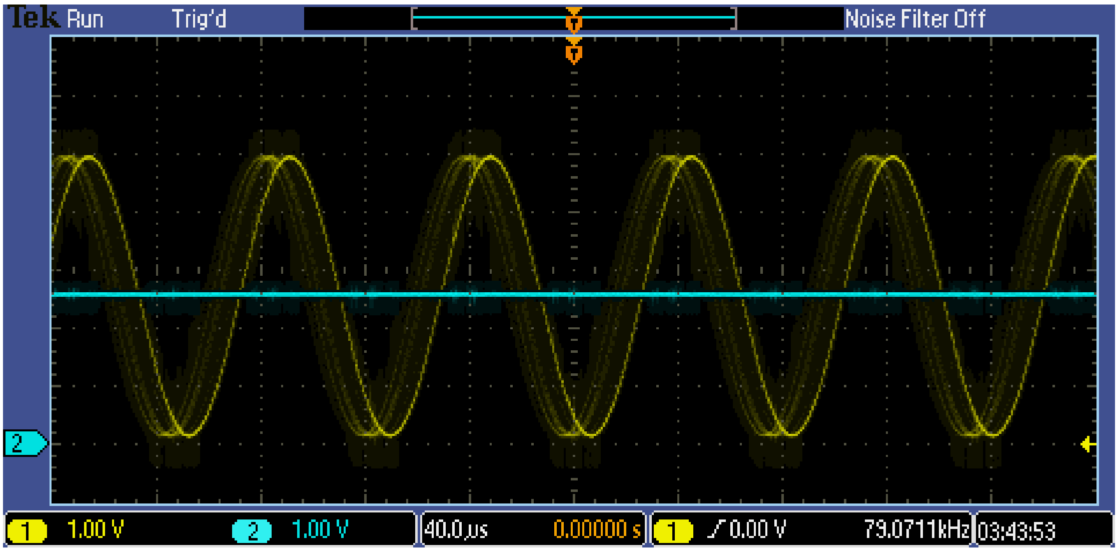


Figure 6: Nyquist Frequency found at 13kHz.

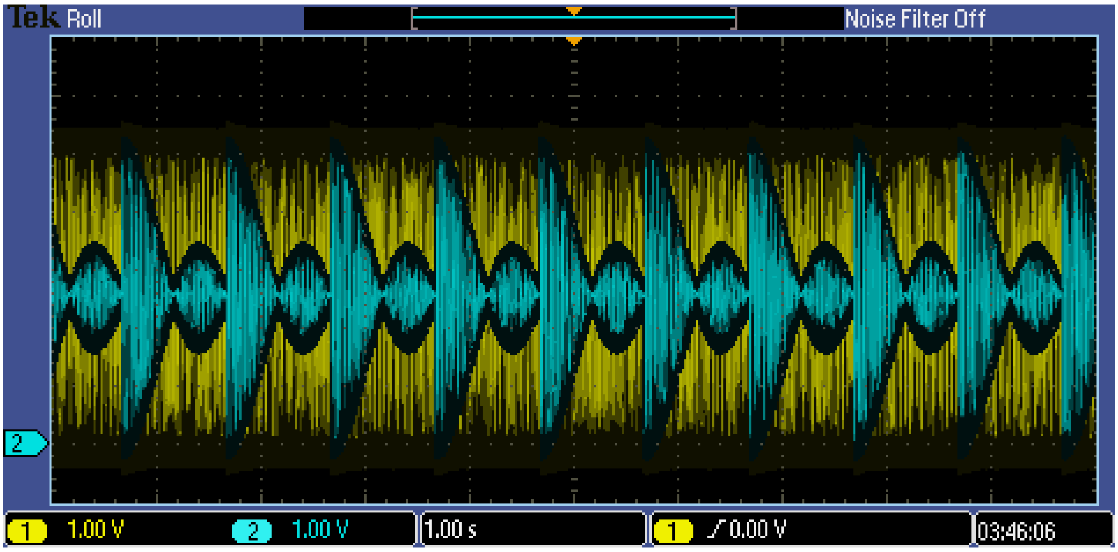


Figure 7: AC Sine Sweep from 1Hz to 13kHz.

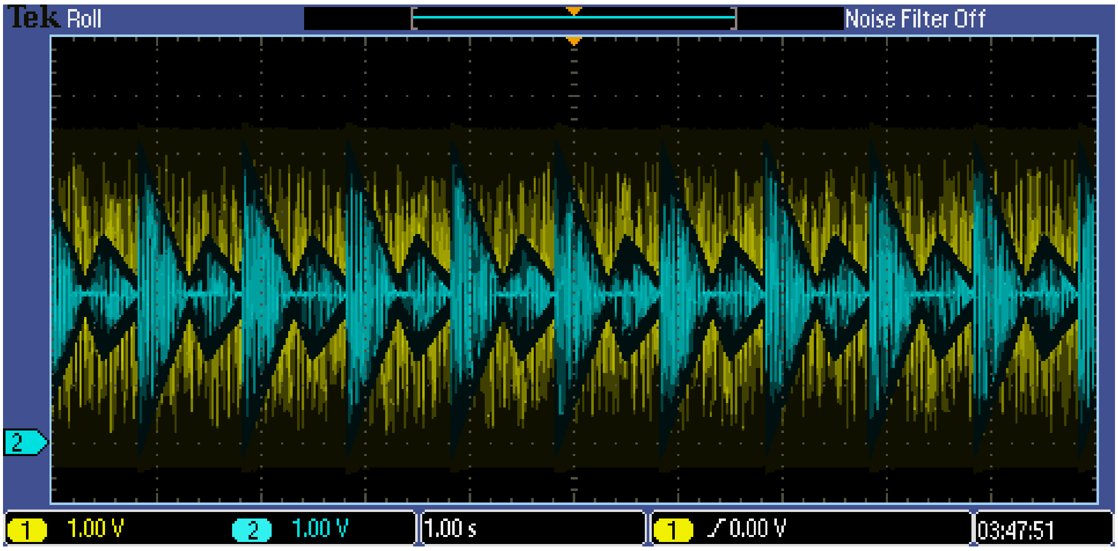


Figure 8: AC Triangle Sweep from 1Hz to 13kHz.

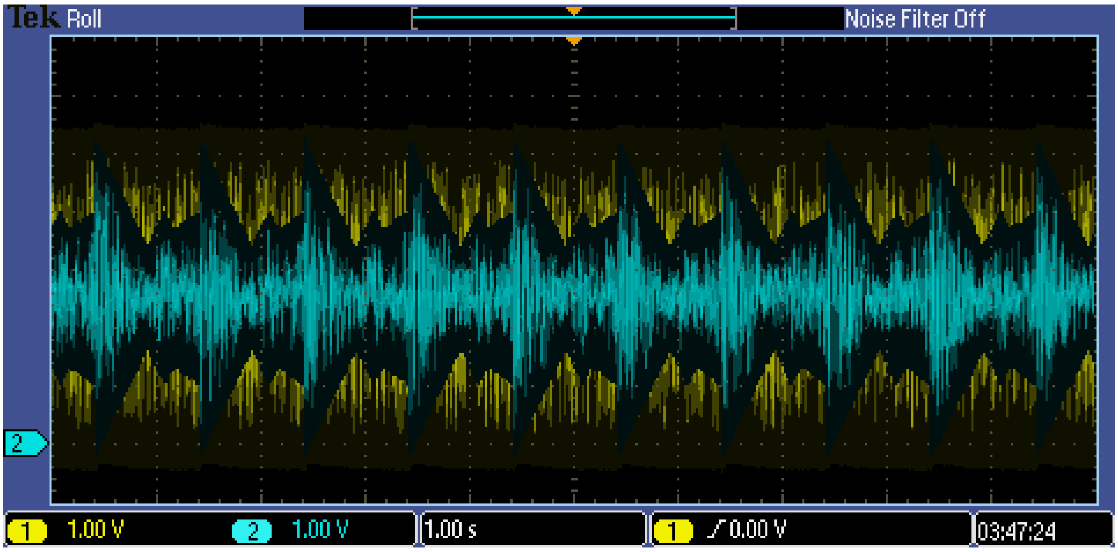


Figure 9: AC Ramp Sweep from 1Hz to 13kHz.

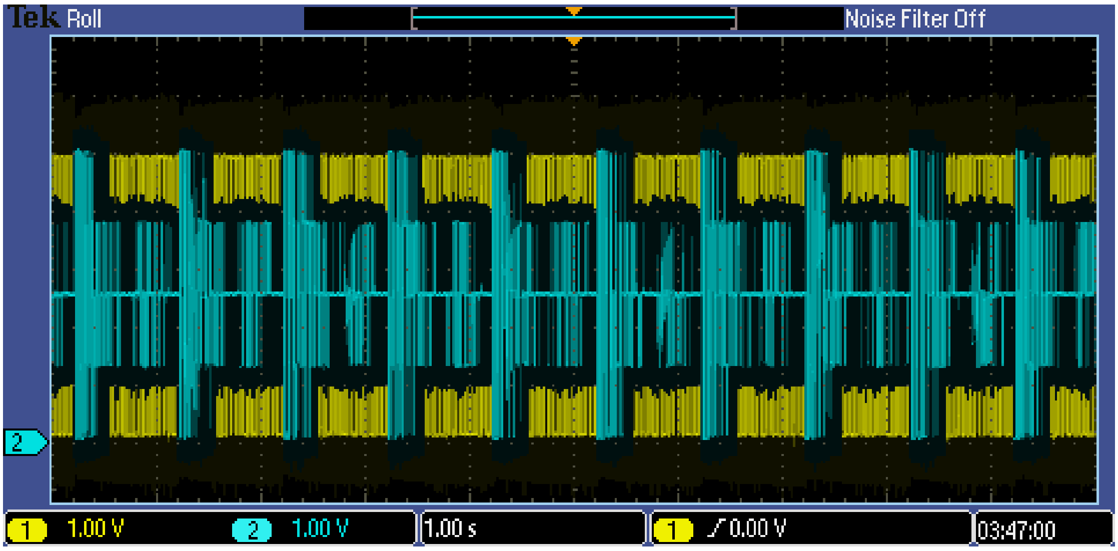
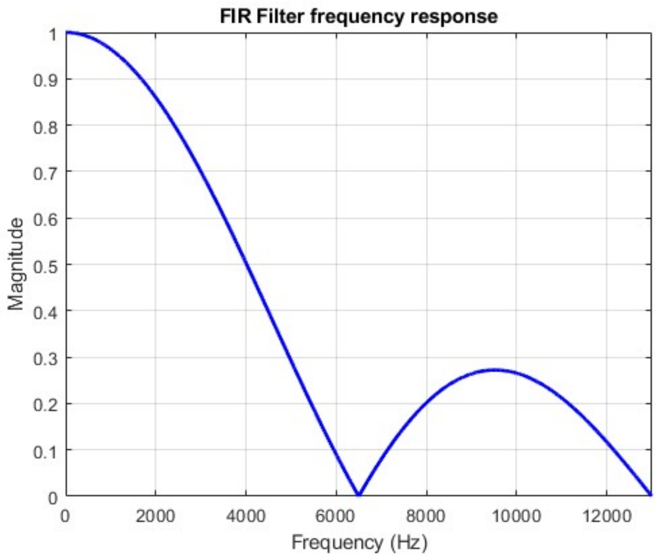
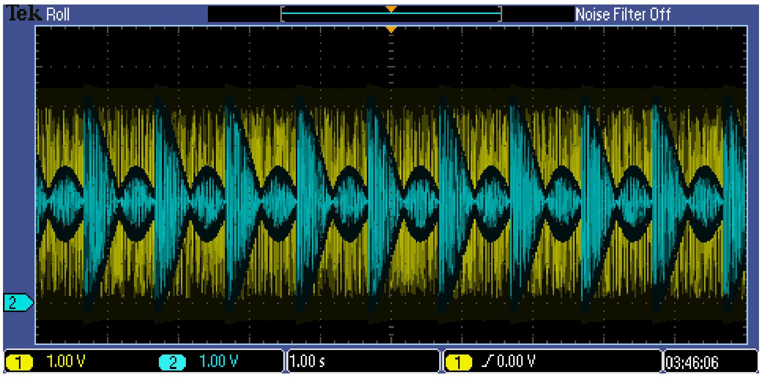


Figure 10: AC Square Sweep from 1Hz to 13kHz.

4 Discussion

The Nyquist Frequency was found to be 13kHz. The Sampling rate of the code is double the Nyquist at 26kHz.





If we plug our sampling rate, 26kHz, into the MATLAB code (Top), we see a similar frequency response as our Measured Data (Bottom).

5 References

[1] John B. Peatman. *Embedded Design with the PIC18F452 Microcontroller.* Prentice Hall, 2004.

6 Appendix A

AC Subroutine

```
;AC
        bsf         ADCON0,1

ADLoop
        btfsc       ADCON0,1
        bra         ADLoop
```

7 Appendix B

Linear Memory Buffer Subroutine

```
;Memory
        movff           V2H,V3H
        movff           V2L,V3L
        movff           V1H,V2H
        movff           V1L,V2L
        movff           V0H,V1H
        movff           V0L,V1L
        movff           ADRESH,V0H
        movff           ADRESL,V0L
```

8 Appendix C

Adder Subroutine

```
;Adder
        movff           V0L,SumTotalL
        movff           V0H,SumTotalH

        movf            V1L,W
        addwf           SumTotalL,F
        movf            V1H,W
        addwfc          SumTotalH,F

        movf            V2L,W
        addwf           SumTotalL,F
        movf            V2H,W
        addwfc          SumTotalH,F

        movf            V3L,W
        addwf           SumTotalL,F
        movf            V3H,W
        addwfc          SumTotalH,F
```

9 Appendix D

Divider Subroutine

```
;Divider
        movf            SumTotalL,W
        andlw           B'11110000'
        movwf           SumTotalL
        swapf           SumTotalL,F

        movf            SumTotalH,W
        andlw           B'00001111'
        movwf           SumTotalH
        swapf           SumTotalH,F

        movf            SumTotalL,W
        addwf           SumTotalH,W
        movwf           Total
```

10 Appendix E

DC Subroutine

```
;DCHigh
        bcf         PORTC,RC0
        bcf         PIR1,SSPIF
        MOVLF       0x21,SSPBUF

DCLoop1
        btfss       PIR1,SSPIF
        bra         DCLoop1
        bcf         PIR1,SSPIF
        movff       Total,SSPBUF

DCLoop2
        btfss       PIR1,SSPIF
        bra         DCLoop2
        bsf         PORTC,RC0
```