

# FIR Digital Filter

Robert Bara

[Robert.Bara@temple.edu](mailto:Robert.Bara@temple.edu)

## Summary

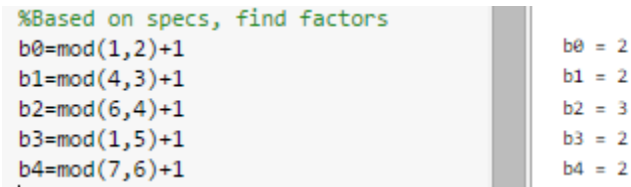
Lab 8 implements a finite impulse response (FIR) digital filter using the Zybo Board, in conjunction with the PmodAD1 ADC and PmodAD2 DAC. The filter is designed using C programming and based upon my TUID. Additionally, LEDs display verification of saturation and switches control a gain factor to be applied to the output.

## Introduction

From the transfer function of an FIR digital filter:

$$H(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4}$$

The coefficients are unique to the last five digits of my TUID:14617, and were calculated using modulo operations with addition of 1. This can be calculated through MATLAB:



```
%Based on specs, find factors
b0=mod(1,2)+1
b1=mod(4,3)+1
b2=mod(6,4)+1
b3=mod(1,5)+1
b4=mod(7,6)+1
```

b0 =	2
b1 =	2
b2 =	3
b3 =	2
b4 =	2

Figure 1. MATLAB coefficients calculations

Therefore, the discrete equation can be represented as:

$$y(m) = 2x(m) + 2x(m-1) + 3x(m-2) + 2x(m-3) + 2x(m-4)$$

From this the PmodAD1 unipolar signal should be modified as a bipolar signal by offsetting by -2048. The worst-case output should also be outputted to determine if any scaling is needed for the filter. This is done by calculating  $2048(b_0+b_1+b_2+b_3+b_4)$ , which in my case=22,528, which falls within the signed 16-bit integer range from -32,768 to 32,767, therefore no scaling is needed. After filtering, the 12-bit integer output for the PmodDA2 DAC gets offset by +2048 to accommodate the bipolar output voltage, creating a range of 0 to 4095 for the DAC. Switch inputs are then read and a percentage attenuation is based upon the worst-case scenario to produce the following conditionals:

Sw0=2047/worst-case scenario, creating 100% attenuation

Sw1=1024/worst-case scenario, creating 50% attenuation

Sw2=410/worst-case scenario, creating 20% attenuation

Sw3=205/worst-case scenario, creating 10% attenuation

When all switches are off, the gain is set to 0. LEDs will provide feedback for the FIIR by lighting up based upon the following saturation conditions:

LED0 turns on when positive saturation of the ADC input occurs ( $V_{in} \geq 2048$ )

LED1 turns on when negative saturation of the ADC input occurs ( $V_{in} \leq 2048$ )

LED2 turns on when positive saturation of the DAC output occurs ( $V_{out} \geq 2048$ )

LED3 turns on when negative saturation of the DAC output occurs ( $V_{out} \geq 2048$ )

The output signal should be compared to the input signal for verification, and the chip select can be analyzed to obtain the sampling rate. From the sampling rate, the poles, zeroes can be plotted based on the transfer function, and the frequency response should be plotted with the assistance of MATLAB.

## Discussion

### Hardware Design

The hardware design builds off of the previous lab's design but adds a GPIO to contain switches as an input and LEDs as an output. The ZYNQ processor block is added and upon running automation, the FCLK\_CLK1 is added with a frequency of 30 Hz, and FCLK\_CLK2 is added with a frequency of 50 Hz. The PmodAD1 is added and connected to CLK1 and PmodDA2 is added and connected to CLK2. Upon running automation, the rst block and ps7\_axi\_periph blocks are generated. The remaining input and outputs of the PmodAD1 and PmodDA2 should be made external, additionally the constraint file *AD1DA2JE.xdc* should be added. Finally, add a GPIO to select switches as an input in channel 1 and LEDs as an output in channel 2. Run automation, verify the design, create an HDL wrapper, and generate a bitstream. Upon completion, export to hardware, and launch SDK.

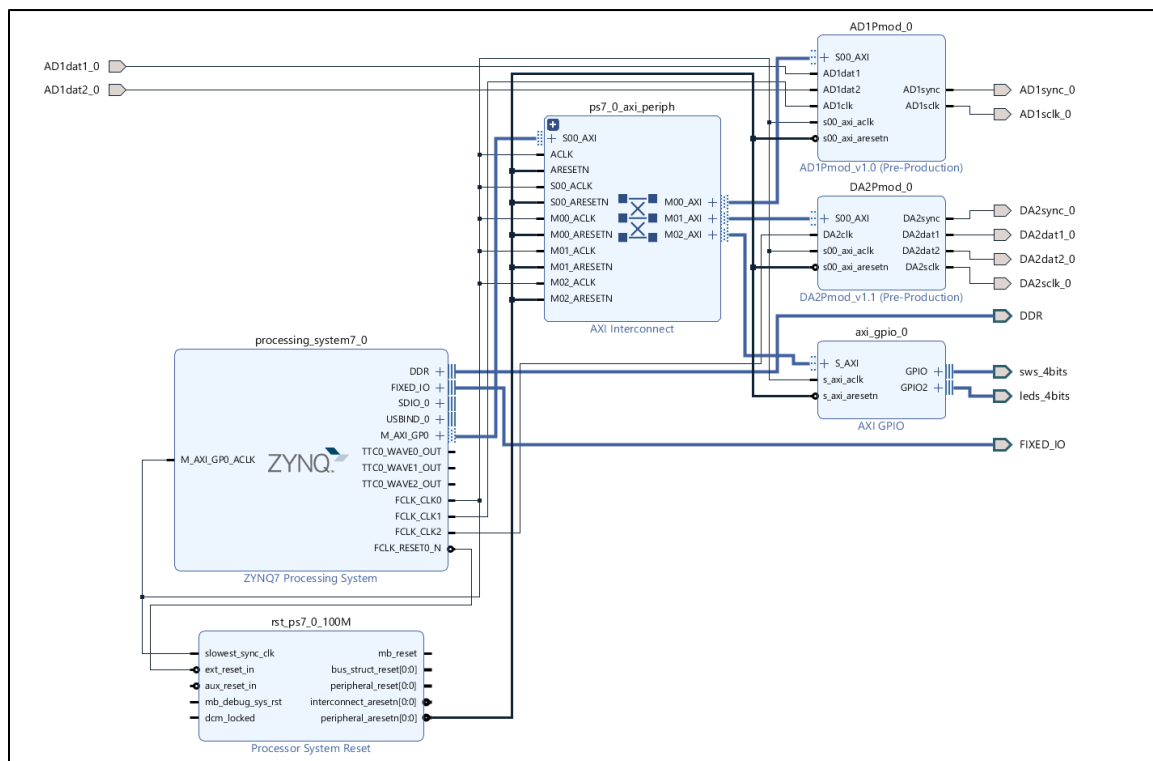


Figure 2. Block Diagram for FIR digital filter

### Software Design

The software design can be modified based off the template program *AD1DA2Pmod.c*. The math library should be added through SDK's settings C/C++ build settings, under the ARM v7 gcc linker, by adding the "m" library.

### Initialization

The Header files are added, addresses for the AD1, DA2, and GPIOs, obtained from Vivado's address editor. GPIOs should be mapped to the LEDs and Switches:

```

//FIR Digital Filter Lab8

#include "xparameters.h"
#include "xil_io.h"
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "sleep.h"
#include "math.h"
#include "xgpio.h"

//AD1Pmod from Address Editor in Vivado, first IP
#define AD1acq 0x43C00000 //AD1 acquisition - output
#define AD1dav 0x43C00004 //AD1 data available - input
#define AD1dat1 0x43C00008 //AD1 channel 1 data - input
#define AD1dat2 0x43C0000C //AD1 channel 2 data - input

//DAC2Pmod from Address Editor in Vivado, second IP
#define DA2acq 0x43C10000 //DA2 acquisition - output
#define DA2dav 0x43C10004 //DA2 data available - input
#define DA2dat1 0x43C10008 //DA2 channel 1 data - output
#define DA2dat2 0x43C1000C //DA2 channel 2 data - output

//GPIO address
#define SW_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID
#define LEDS_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID
//define printf xil_printf
XGpio LEDInst, SWInst;

```

Figure 3. Initializing headers and addressing

### Subroutines

Within main, variables are declared to map data into and out of the Pmods, the coefficients of the FIR filter are declared to carry the input signal and the modulo operation for each digit of my TUID calculate the scaler coefficients of the discrete equation as well as the worst case scenario to determine if scaling is necessary.

```

int main(void)
{
    int adcdav;      //ADC data available
    int adcddata1;   //ADC channel 1 data
    int adcddata2;   //ADC channel 2 data

    int dacdata1;    //DAC channel 1 data
    int dacdata2;    //DAC channel 2 data
    int dacdav;      //DAC data available

    int x1a;         //FIR   x(m)
    int x1b;         //      x(m-1)
    int x1c;         //      x(m-2)
    int x1d;         //      x(m-3)
    int x1e;         //      x(m-4)
    int y1a;         //      y(m)

    //From TuID: 915614617 L5SD->14617
    int b0=(1%2)+1;
    int b1=(4%3)+1;
    int b2=(6%4)+1;
    int b3=(1%5)+1;
    int b4=(7%6)+1;
    int wcs=2048*(b0+b1+b2+b3+b4);    //the worst case scenario

    int status,ytemp,sw=0,led=0;
    float gain;

```

Figure 4. Declaring the Filter variables and Pmod variables

Initialization of the peripherals and directions of the GPIOs are as follows. The ADC and DAC checks if data is available, and collects data within the next while loop:

```
//-----
// INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO
//-----
// Initialise LEDs
status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
if(status != XST_SUCCESS) return XST_FAILURE;
// Initialise Switches
status = XGpio_Initialize(&SWInst, SW_DEVICE_ID);
if(status != XST_SUCCESS) return XST_FAILURE;
// Set LEDs direction to outputs
XGpio_SetDataDirection(&LEDInst, 2, 0x00);
// Set all SW direction to inputs
XGpio_SetDataDirection(&SWInst, 1, 0xFF);
XGpio_DiscreteWrite(&LEDInst, 2, 0); //initialize LEDs as off

xil_printf("\n\rStarting AD1-DA2 Pmod FIR...\n");
Xil_Out32(AD1acq,0); //ADC stop acquire
adcdav=Xil_In32(AD1dav); //ADC available?
while(adcdav==1)
    adcdav=Xil_In32(AD1dav);
Xil_Out32(DA2acq,0); //DAC stop acquire
dacdav=Xil_In32(DA2dav); //DAC available?
while(dacdav==1)
    dacdav=Xil_In32(DA2dav);
```

Figure 5. Checking Initialization and Setting Direction of GPIOs

The filter is coded as follows after offsetting the input signal to create a bipolar signal. An if statement determines if scaling is needed and applies it or bypasses it. After that the gain attenuation is performed by reading the switch values, typecasting to perform calculations within a switch statement. After the gain value is properly determined, a temporary variable multiplies this to the filtered signal and typecasts back to integer. Finally, the offset is corrected to be a unipolar signal for the DAC to output. To compare the bypassed input signal to the filtered signal, DAC's 2<sup>nd</sup> channel reads the analog signal from the ADC channel 1.

```
//ADC ch1 -> DAC ch1 FIR
x1e=x1d;    //implementing the discrete equation
x1d=x1c;    //by passing the Analog signal through each
x1c=x1b;    //coefficient of the filter
x1b=x1a;
adcddata1=adcddata1-2048;    //offset for ADC
x1a=adcddata1;
y1a=b0*x1a+b1*x1b+b2*x1c+b3*x1d+b4*x1e; // FIR

/*if scaling was needed, scale by 100. No scaling will be needed because
 * my TUID, computing WCS=2040(b0+b1+b2+b3+b4)=22,528
 * which is less than the 16 signed int max of -32,768 to 32,767. So no scaler is required
 * This is left in for safety though */
if(y1a>32767) ytemp=y1a/100;
else ytemp=y1a;

//Reading Sw Input to determine attenuation
sw = XGpio_DiscreteRead(&SWInst, 1);
switch(sw)
{
    case 0b0001:    gain=(float)2047/(float)wcs; break; //attenuating by 100%
    case 0b0010:    gain=(float)1024/(float)wcs; break; //attenuating by 50%
    case 0b0100:    gain=(float)410/(float)wcs; break; //attenuating by 20%
    case 0b1000:    gain=(float)205/(float)wcs; break; //attenuating by 10%
    default:        gain=0; break; //otherwise, set the signal to 0
}
ytemp=(float)gain*(float)y1a;

//outputting
dacdata1=ytemp+2048;    //ADC ch1 with filter and attenuation -> DAC ch2
dacdata2=adcddata1+2048;    //ADC ch1 -> DAC ch2 straight through
```

Figure 6. FIR Filter and Gain attenuation

The LED specifications are then as follows and outputs to the DAC.

```
//outputting
dacdata1=ytemp+2048; //ADC ch1 with filter and attenuation -> DAC ch2
dacdata2=adcdata1+2048; //ADC ch1 -> DAC ch2 straight through

//Based on the Input and Outputs, turn on or off LEDs
XGpio_DiscreteWrite(&LEDInst, 2, 0); //initialize LEDs as off
if(adcdata1>=0){ led=0; led=1; XGpio_DiscreteWrite(&LEDInst, 2, led);} //vin>v/2 turn on LED 0, vout>v/2 turn on LED 2
if(adcdata1<=0){ led=0; led=2; XGpio_DiscreteWrite(&LEDInst, 2, led);} //vin<v/2 turn on LED 1, vout>v/2 turn on LED 3
if(dacdata1>=2048){led=0; led=4; XGpio_DiscreteWrite(&LEDInst, 2, led);} //vout>v/2 turn on LED 2
if(dacdata1<=2048){led=0; led=8; XGpio_DiscreteWrite(&LEDInst, 2, led);} //vout>v/2 turn on LED 3

//DAC
Xil_Out32(DA2dat1, dacdata1); //output DAC data
Xil_Out32(DA2dat2, dacdata2);
Xil_Out32(DA2acq,1); //DAC acquire
while (dacdav==0) //DAC data output?
    dacdav=Xil_In32(DA2dav);
Xil_Out32(DA2acq,0); //stop DAC acquire
while(dacdav==1) //wait for reset
    dacdav=Xil_In32(DA2dav);
}
```

Figure 7. Lighting up LEDs and Outputting to DAC

## Verification

### Waveforms

For consistency, I ran the following sinewave waveform:

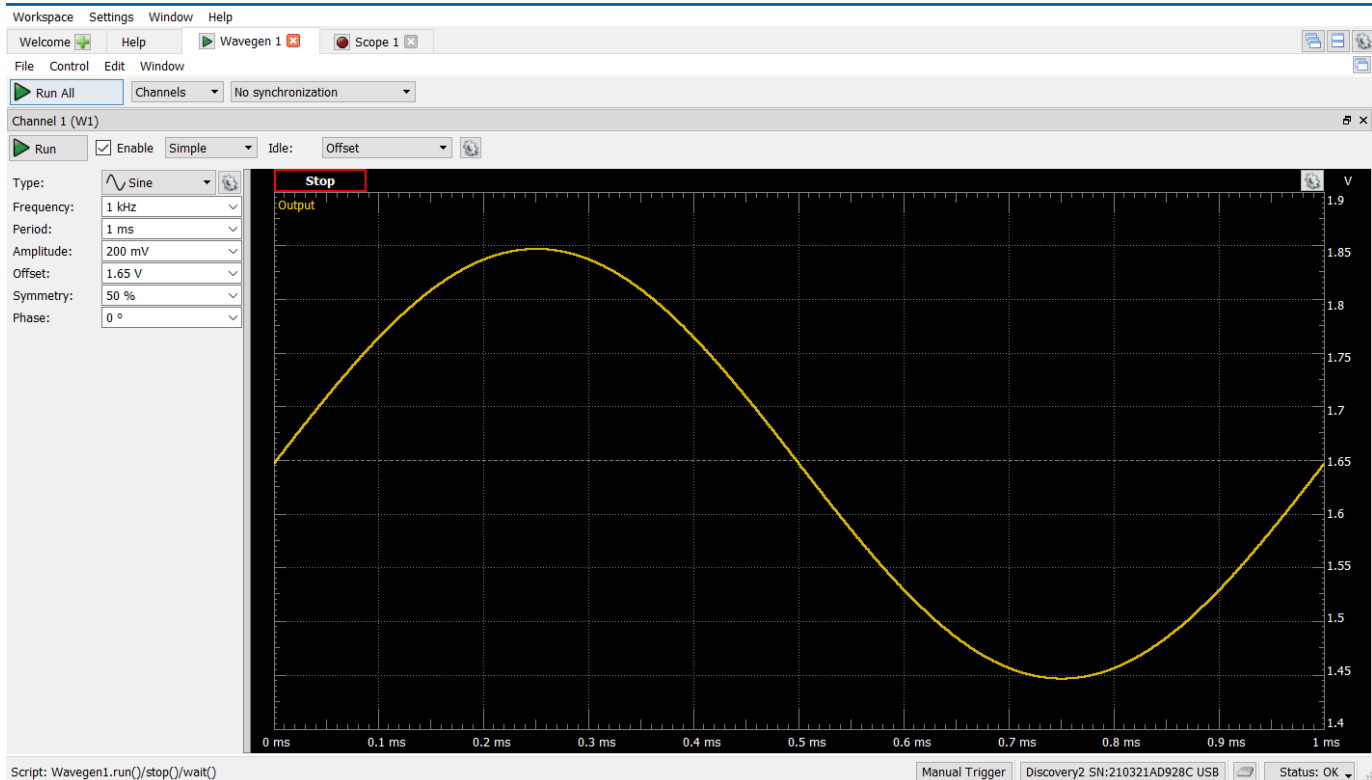


Figure 8. Input Wave



### Filtered output vs Input Reading

For Sw0 On: Gain is 100% C1 is the Filtered output, C2 is the input:

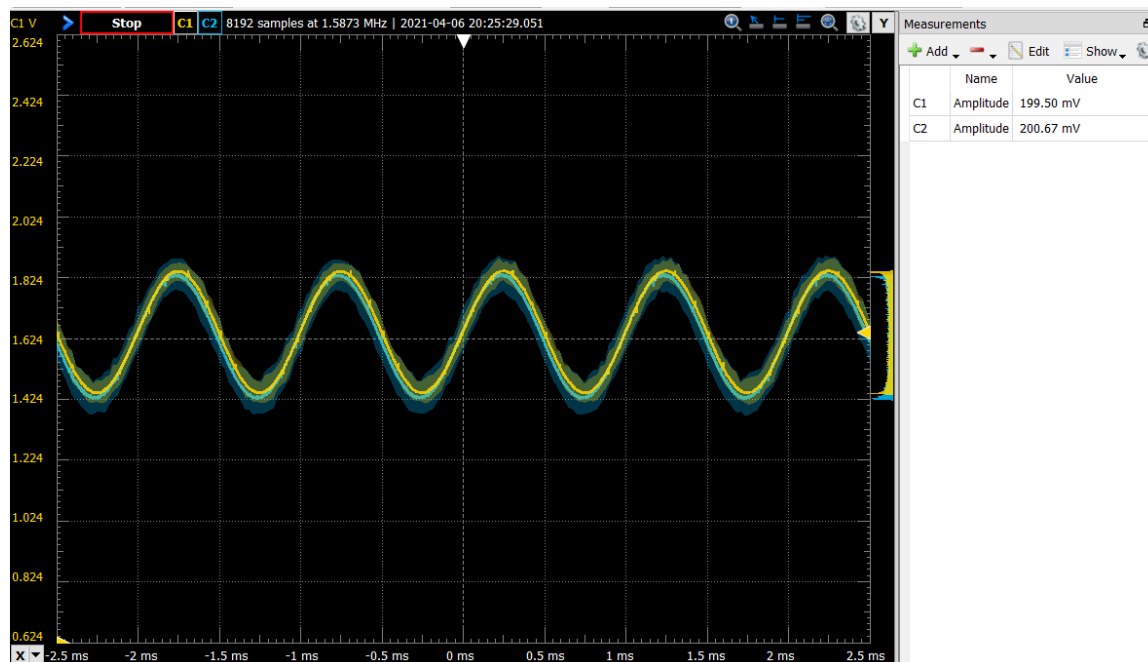


Figure 9. When SW0 is on

For Sw1 On: Gain is 50% C1 is the Filtered output, C2 is the input:

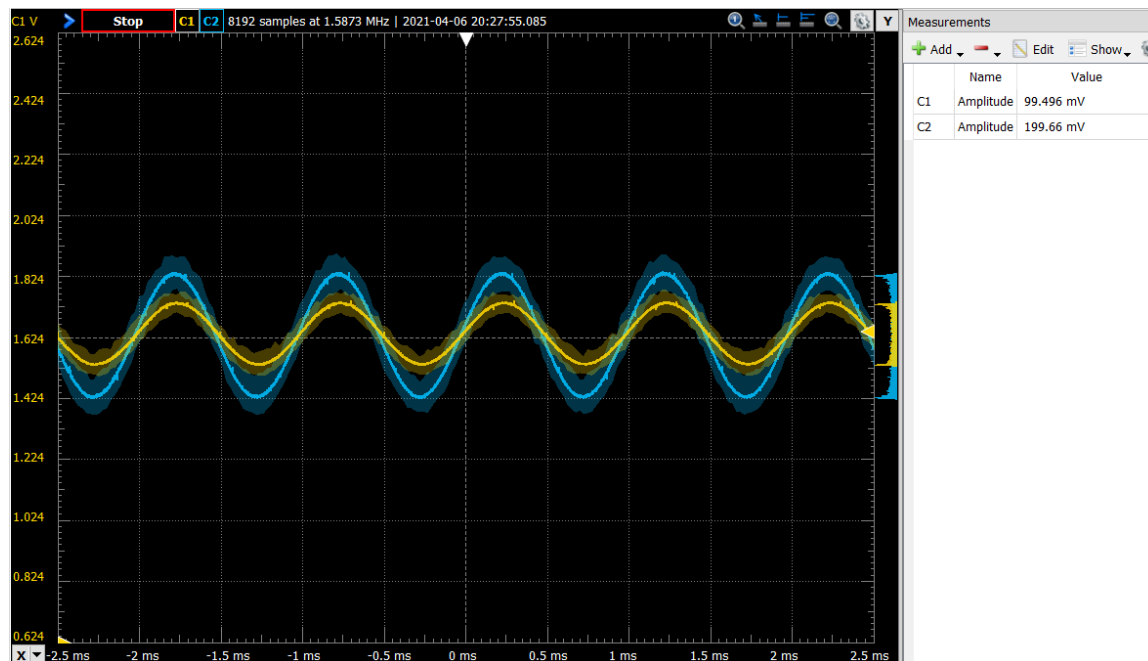


Figure 10. When SW1 is on

For Sw2 On: Gain is 20% C1 is the Filtered output, C2 is the input:

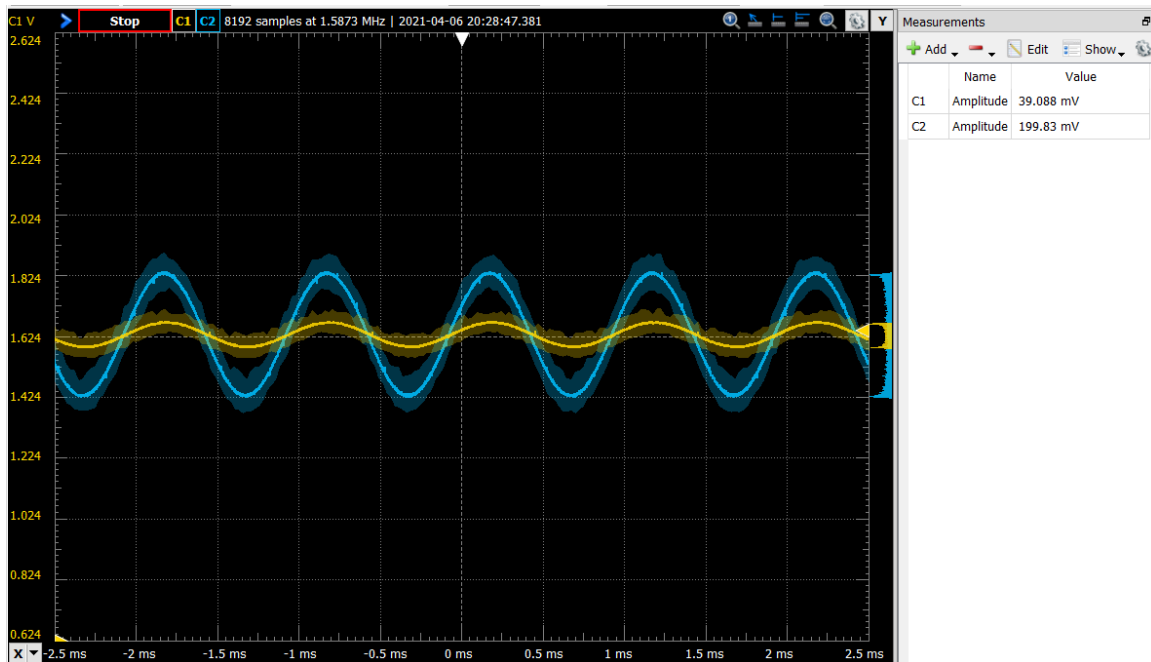


Figure 11. When SW2 is on

For Sw3 On: Gain is 10% C1 is the Filtered output, C2 is the input:

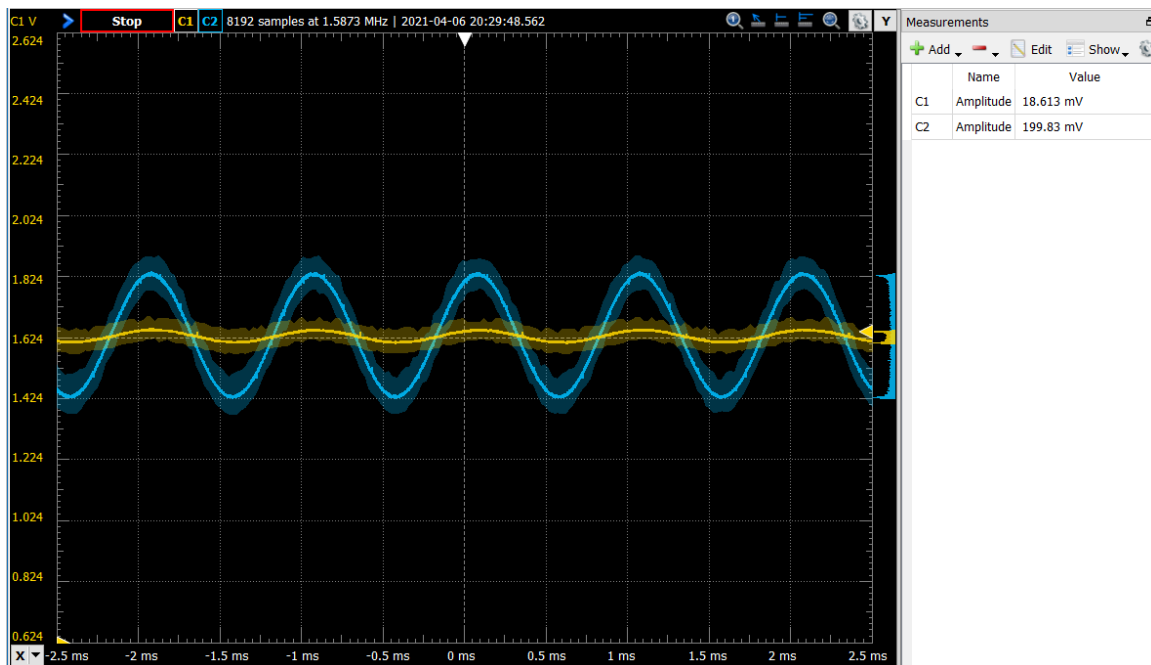


Figure 12. When SW3 is on

Otherwise, Gain of the filter is 0:

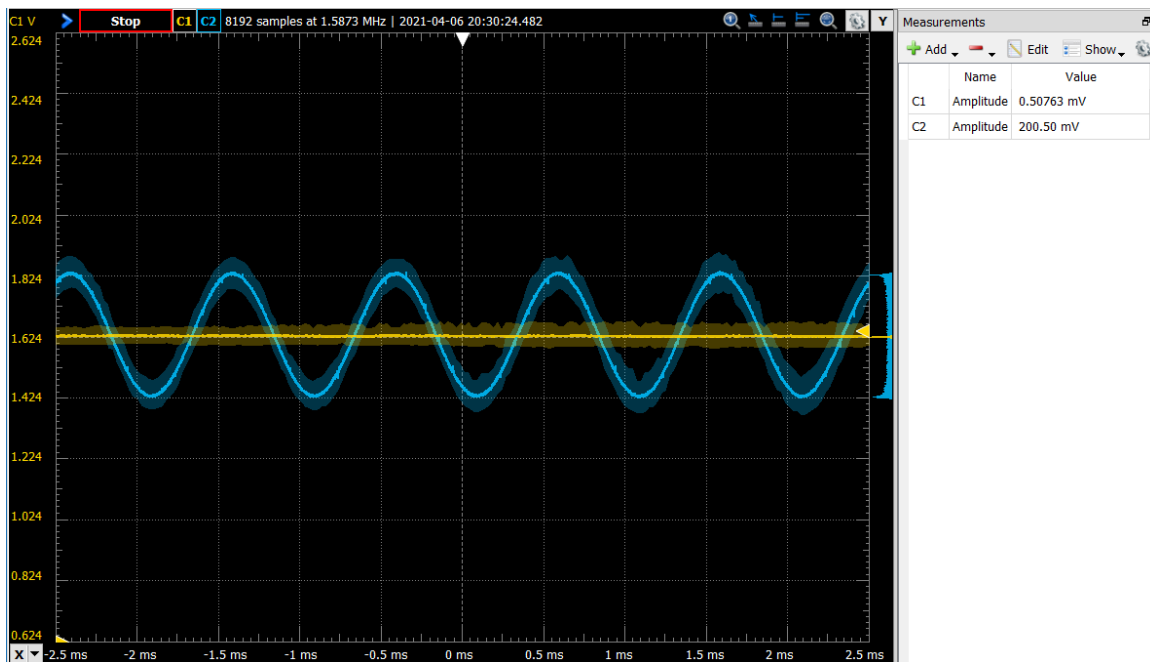


Figure 13. When SW's are not a valid input

#### Chip Select Reading to find Sample Rate

Below is the chip select reading where the sample frequency is approximately 178kHz, so the Nyquist rate is approximately 89 kHz



Figure 14. Chip Select Reading

#### MATLAB Analysis

##### Poles and Zeros

Using the following code, the plot of poles and zeros for my transfer function are as follows:

## Poles & Zeros Diagram

```
%Based on specs, find factors
b0=mod(1,2)+1
b1=mod(4,3)+1
b2=mod(6,4)+1
b3=mod(1,5)+1
b4=mod(7,6)+1

%determine scaling?
scaling=2048*(b0+b1+b2+b3+b4)
%16 bit signed integer ranges from -32,768 to 32,767
if(scaling<32767) x=true
else x=false
end

G=1; %gain of 1
coefficients=G*[b0,b1,b2,b3,b4];%z^4,z^3,z^2,z^1,z^0
zeros=roots(coefficients);%zeros
poles=[]; %poles

figure(1)
zplane(zeros,poles)
```

Yielding the following Pole/Zero plot:

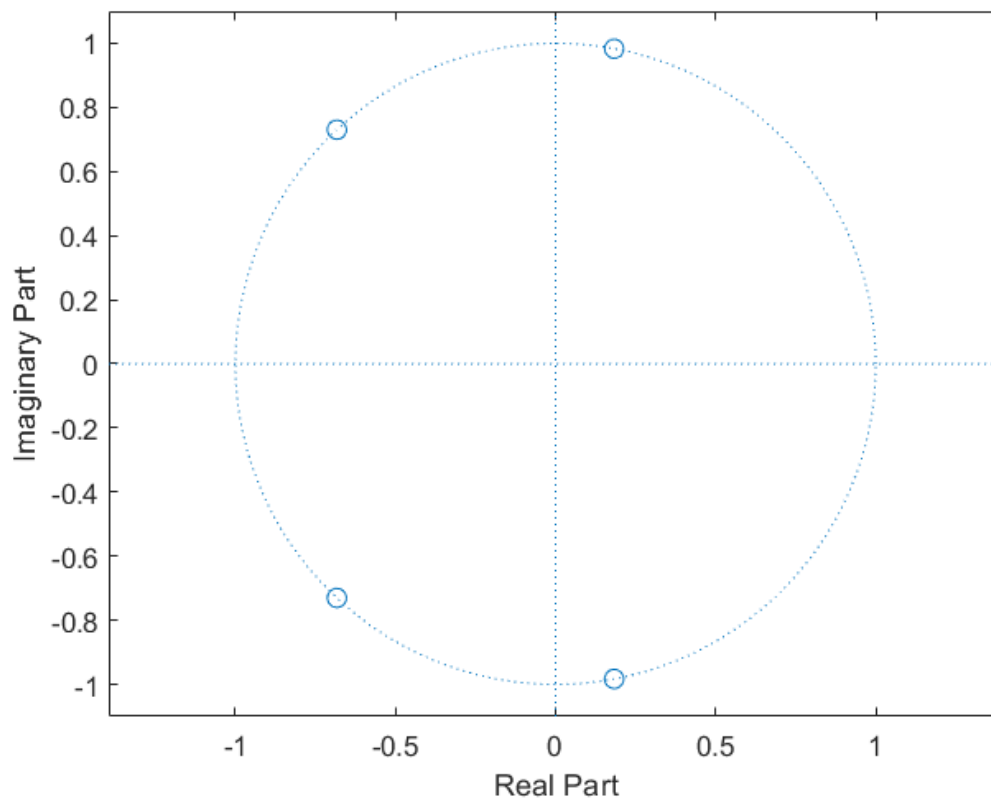


Figure 15. Poles and Zeros plot based on TUID transfer function

### Frequency Response

Using MATLAB and the sampling frequency from the chip select, the frequency response was plotted as follows, notice the yline at -3dB or 0.707 to determine if the filter experimentally matches:

### Frequency Response plot

```
fs=178e3;    %sampling freq
ts=1/fs;    %sampling percent
[H,W]=freqz(coefficients,[1],fs);
W_0=(linspace(0,fs/2, length(W)))';

%plot freq response
figure(2)
plot(W_0,abs(H));
set(gca,'xlim',[0 fs/2]); grid on;
yline(.707);
```

```
xlabel('Hz');
```

```
ylabel('Magnitude');
```

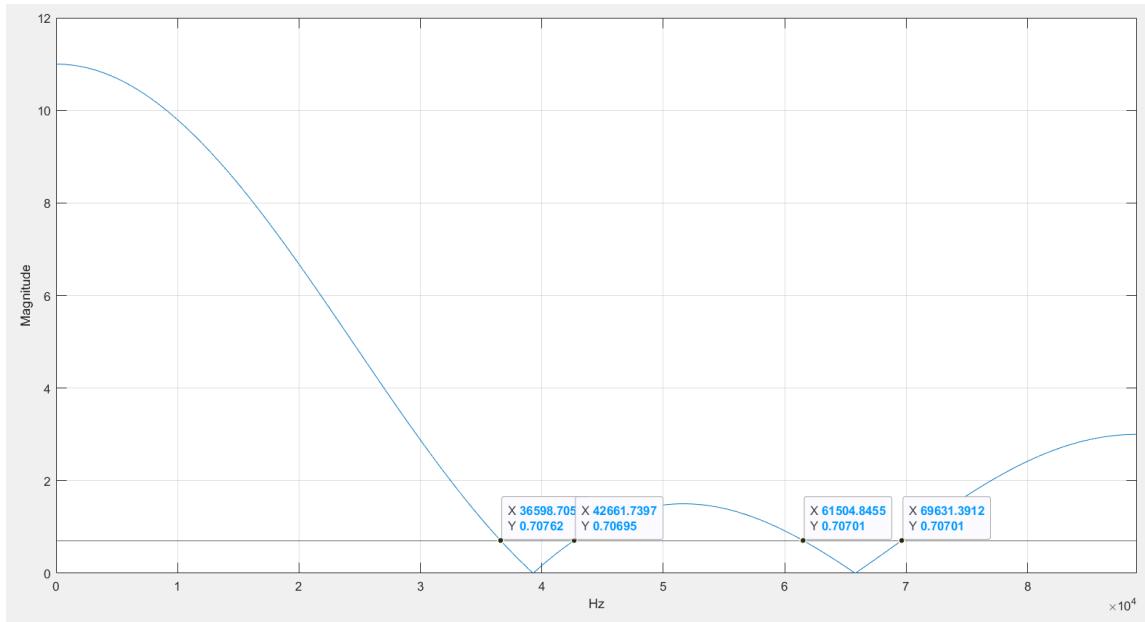


Figure 16. Simulated Frequency Response

Measuring and plotting the experimental Frequency Response exceeds the simulations x-range with different y-values but this could be because I used an amplitude of 200mV. Nonetheless it holds a similar shape:

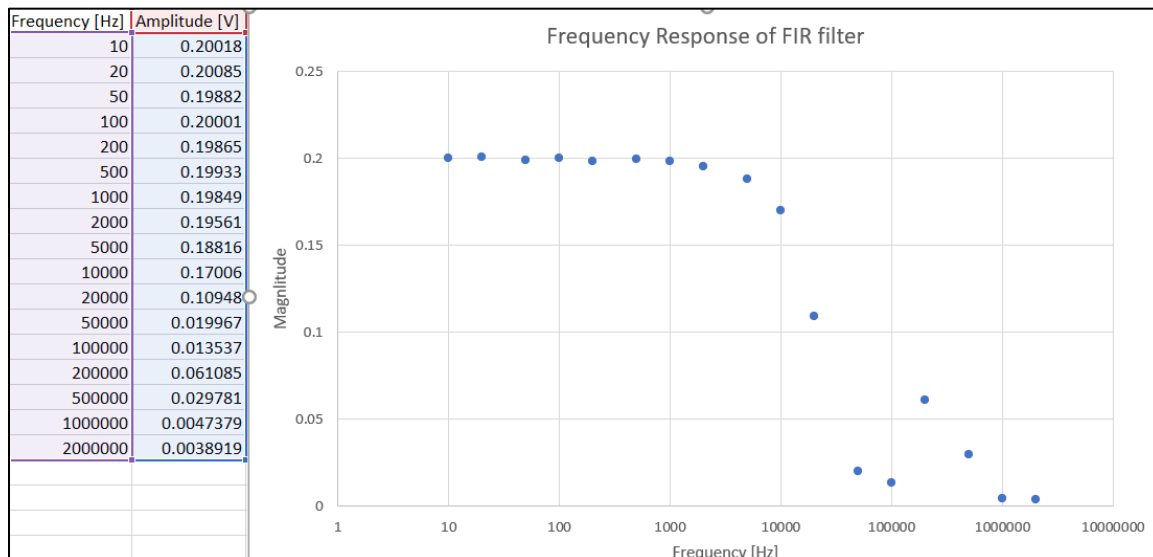


Figure 17. Excel analysis of Frequency Response

Observing the frequency rolls off right before the filter hits approximately 1000 Hz which matches the MATLAB simulation, and this is happening most likely when the frequency exceeds the Nyquist frequency=85kHz. Upon trying to examine when my filter

hits -3dB or 0.707, I put a horizontal line within the MATLAB simulation, in which this value should be reached at approximately 36.5kHz, 42.7kHz, 61.5kHz, and 69.6kHz. Trying to achieve this experimentally I changed my amplitude to 1V and examined the maximum values of my sinewave at these frequencies. My filter based upon my TU ID never reaches -3dB because it filters beyond that, and this is with sw0 high, so it is all of the signal. Subtracting the offset of 1.65V from the maximum values does yield similar amplitudes, however this is always within the range of approximately -22 to -23 dB.

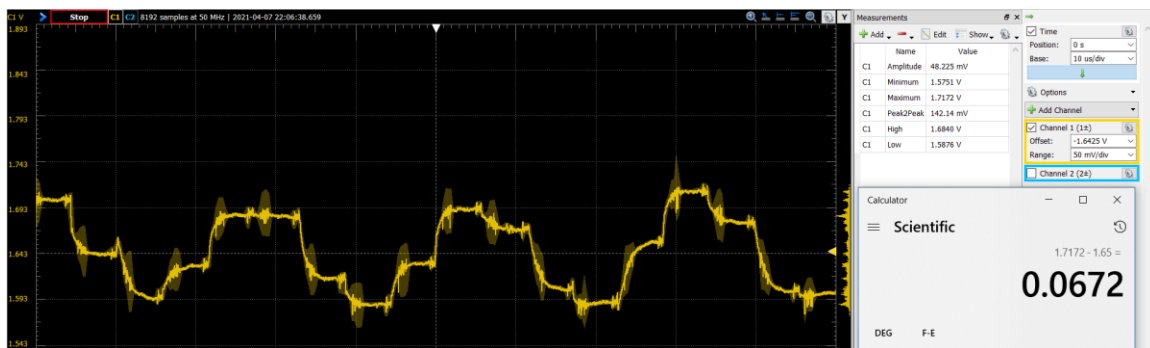


Figure 18. At 36.5 kHz

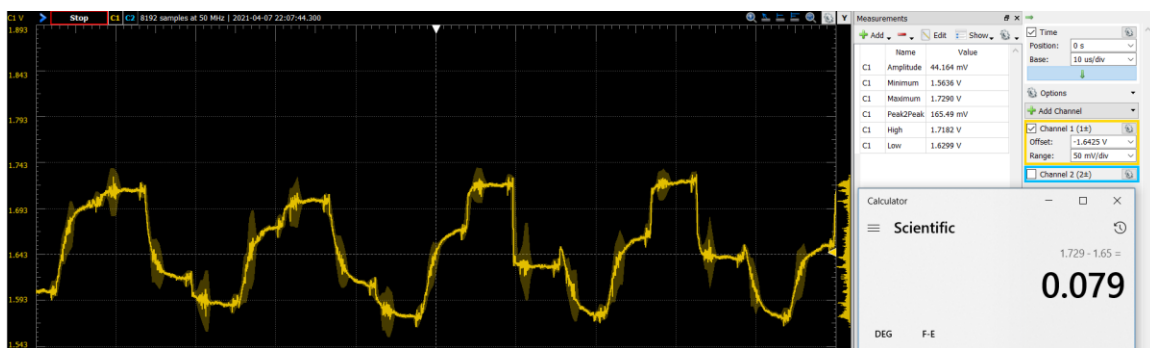


Figure 19. 42.7 kHz

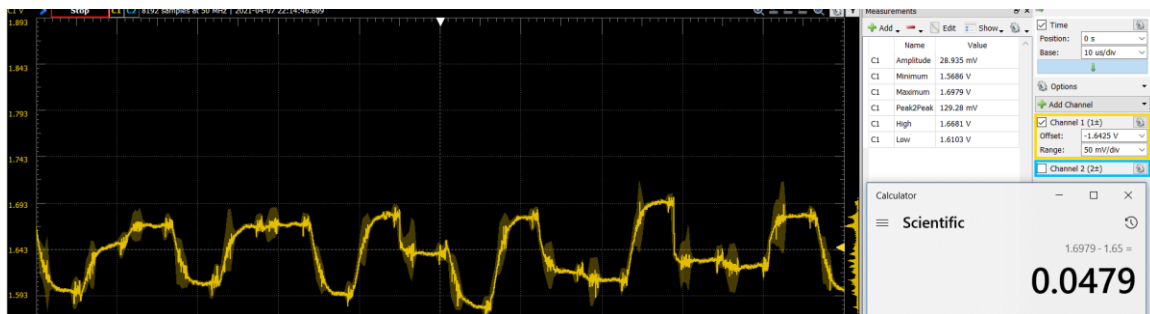


Figure 20. 61.5 kHz

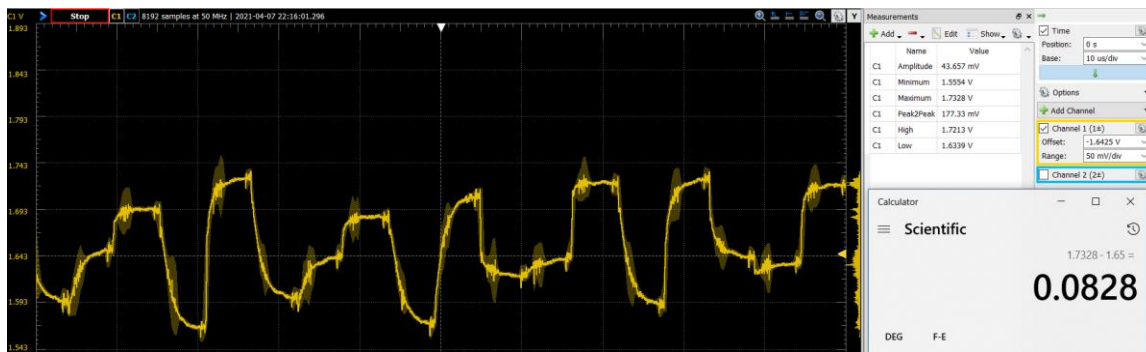


Figure 21. 69.6 kHz

#### Video Link Verification

This will showcase the LEDs blinking.

<https://youtu.be/Sa2kHcYoUn8>

#### LEDs



Figure 22a&b. SW0 high, LEDs blinking

#### Conclusion

The FIR digital filter tackles previous concepts such as SPI interfacing for the hardware and DSP by using ADC and DAC Pmods to create a digital filter specified to our TU IDs. Upon completion of the lab, I was able to calculate the discrete equation from my TU ID, perform gain attenuation from utilizing the Zybo board's switches, and verify through LEDs, MATLAB analysis, and Waveform/excel analysis. Overall, this lab serves as an introduction to real-world applications of DSP, since filtering can be across the board in terms of Electrical Engineering from audio to radar or machine learning. Though I am a student, it is possible some of my analysis could be wrong, especially when trying to find my experimental values for when the filter is -3dB, but I think this lab was helpful as someone who is a musician with an interest in audio processing and plans to dig deeper by taking Dr. Ahmad's DSP class within the next semester.



## Appendix

### C Code

//FIR Digital Filter Lab8

//ECE3622 Robert Bara

```
#include "xparameters.h"
```

```
#include "xil_io.h"
```

```
#include <stdio.h>
```

```
#include "platform.h"
```

```
#include "xil_printf.h"
```

```
#include "sleep.h"
```

```
#include "math.h"
```

```
#include "xgpio.h"
```

//AD1Pmod from Address Editor in Vivado, first IP

```
#define AD1acq      0x43C00000 //AD1 acquisition - output
```

```
#define AD1dav      0x43C00004 //AD1 data available - input
```

```
#define AD1dat1     0x43C00008 //AD1 channel 1 data - input
```

```
#define AD1dat2     0x43C0000C //AD1 channel 2 data - input
```

//DAC2Pmod from Address Editor in Vivado, second IP

```
#define DA2acq      0x43C10000 //DA2 acquisition - output
```

```
#define DA2dav      0x43C10004 //DA2 data available - input
```

```
#define DA2dat1     0x43C10008 //DA2 channel 1 data - output
```

```
#define DA2dat2     0x43C1000C //DA2 channel 2 data - output
```

//GPIO address

```
#define SW_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID
```

```
#define LEDS_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID
```

```

//#define printf                xil_printf
XGpio LEDInst, SWInst;

int main(void)
{
    int adcdav;                //ADC data available
    int adcdata1;              //ADC channel 1 data
    int adcdata2;              //ADC channel 2 data

    int dacdata1;              //DAC channel 1 data
    int dacdata2;              //DAC channel 2 data
    int dacdav;                //DAC data available

    int x1a;                   //FIR  x(m)
    int x1b;                   //    x(m-1)
    int x1c;                   //    x(m-2)
    int x1d;                   //    x(m-3)
    int x1e;                   //    x(m-4)
    int y1a;                   //    y(m)

    //From TuID: 915614617 L5SD->14617
    int b0=(1%2)+1;
    int b1=(4%3)+1;
    int b2=(6%4)+1;
    int b3=(1%5)+1;
    int b4=(7%6)+1;
    int wcs=2048*(b0+b1+b2+b3+b4);        //the worst case scenario

```

```

int status,ytemp,sw=0,led=0;

float gain;

//-----

// INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO
//-----

// Initialise LEDs
status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
if(status != XST_SUCCESS) return XST_FAILURE;

// Initialise Switches
status = XGpio_Initialize(&SWInst, SW_DEVICE_ID);
if(status != XST_SUCCESS) return XST_FAILURE;

// Set LEDs direction to outputs
XGpio_SetDataDirection(&LEDInst, 2, 0x00);

// Set all SW direction to inputs
XGpio_SetDataDirection(&SWInst, 1, 0xFF);

XGpio_DiscreteWrite(&LEDInst, 2, 0); //initialize LEDs as off


xil_printf("\n\rStarting AD1-DA2 Pmod FIR...\n");
Xil_Out32(AD1acq,0); //ADC stop acquire
adcdav=Xil_In32(AD1dav); //ADC available?
while(adcdav==1)
    adcdav=Xil_In32(AD1dav);
Xil_Out32(DA2acq,0); //DAC stop acquire
dacdav=Xil_In32(DA2dav); //DAC available?
while(dacdav==1)
    dacdav=Xil_In32(DA2dav);

```

```

while (1)
{

    //ADC
    Xil_Out32(AD1acq,1);          //ADC acquire
    while (adcdav==0)              //ADC data available?
        adcdav=Xil_In32(AD1dav);
    Xil_Out32(AD1acq,0);          //ADC stop acquire
    adcddata1=Xil_In32(AD1dat1);   //input ADC data
    adcddata2=Xil_In32(AD1dat2);
    while (adcdav==1)              //wait for reset
        adcdav=Xil_In32(AD1dav);

    //ADC ch1 -> DAC ch1 FIR
    x1e=x1d;    //implementing the discrete equation
    x1d=x1c;    //by passing the Analog signal through each
    x1c=x1b;    //coefficient of the filter
    x1b=x1a;
    adcddata1=adcddata1-2048;      //offset for ADC
    x1a=adcddata1;
    y1a=b0*x1a+b1*x1b+b2*x1c+b3*x1d+b4*x1e; // FIR

    /*if scaling was needed, scale by 100. No scaling will be needed because
    * my TUID, computing WCS=2040(b0+b1+b2+b3+b4)=22,528
    * which is less than the 16 signed int max of -32,768 to 32,767. So no
    scaler is required

    * This is left in for safety though */
    if(y1a>32767) ytemp=y1a/100;

```

```

else ytemp=y1a;

//Reading Sw Input to determine attenuation
sw = XGpio_DiscreteRead(&SWInst, 1);
switch(sw)
{
    case 0b0001: gain=(float)2047/(float)wcs; break; //attenuating
by 100%
    case 0b0010: gain=(float)1024/(float)wcs; break; //attenuating
by 50%
    case 0b0100: gain=(float)410/(float)wcs; break; //attenuating
by 20%
    case 0b1000: gain=(float)205/(float)wcs; break; //attenuating
by 10%
    default: gain=0; break;
//otherwise, set the signal to 0
}
ytemp=(float)gain*(float)y1a;

//outputting
dacdata1=ytemp+2048; //ADC ch1 with filter and attenuation
-> DAC ch2
dacdata2=adcdata1+2048; //ADC ch1 -> DAC ch2 straight
through

//Based on the Input and Outputs, turn on or off LEDs
XGpio_DiscreteWrite(&LEDInst, 2, 0); //initialize LEDs as off
if(adcdata1>=0){ led=0; led=1; XGpio_DiscreteWrite(&LEDInst, 2,
led);} // vin>v/2 turn on LED 0, vout>v/2 turn on LED 2
if(adcdata1<=0){ led=0; led=2; XGpio_DiscreteWrite(&LEDInst, 2, led);
} // vin<v/2 turn on LED 1, vout>v/2 turn on LED 3

```

```

        if(dacdata1>=2048){led=0; led=4; XGpio_DiscreteWrite(&LEDInst, 2,
led);} // vout>v/2 turn on LED 2

```

```

        if(dacdata1<=2048){led=0; led=8; XGpio_DiscreteWrite(&LEDInst, 2,
led);} // vout>v/2 turn on LED 3

```

```

        //DAC
        Xil_Out32(DA2dat1, dacdata1);    //output DAC data
        Xil_Out32(DA2dat2, dacdata2);
        Xil_Out32(DA2acq,1);             //DAC acquire
        while (dacdav==0)                 //DAC data output?
            dacdav=Xil_In32(DA2dav);
        Xil_Out32(DA2acq,0);             //stop DAC acquire
        while(dacdav==1)                  //wait for reset
            dacdav=Xil_In32(DA2dav);
    }
}

```

MATLAB Analysis Code

```

close all
clear all
clc
%915614617

```

## Poles & Zeros Diagram

%Based on specs, find factors

```

b0=mod(1,2)+1
b1=mod(4,3)+1
b2=mod(6,4)+1
b3=mod(1,5)+1
b4=mod(7,6)+1

```

```

%determine scaling?
scaling=2048*(b0+b1+b2+b3+b4)

%16 bit signed integer ranges from -32,768 to 32,767
if(scaling<32767) x=true
else x=false
end

G=1; %gain of 1
coefficients=G*[b0,b1,b2,b3,b4];%z^4,z^3,z^2,z^1,z^0

zeros=roots(coefficients);%zeros
poles=[]; %poles

figure(1)
zplane(zeros,poles)

```

## Frequency Response plot

```

fs=178e3;    %sampling freq
ts=1/fs;    %sampling percent

[H,W]=freqz(coefficients,[1],fs);
W_0=(linspace(0,fs/2,length(W)))';

%plot freq response
figure(2)
plot(W_0,abs(H));
set(gca,'xlim',[0 fs/2]); grid on;
yline(.707);
xlabel('Hz');
ylabel('Magnitude');

```