ECE3623 Embedded System Design Laboratory

Dennis Silage, PhD
silage@temple.edu

FIR Digital Filter

In this Lab 8 you are to implement a finite impulse response (FIR) digital filter in Xilinx Vivado and SDK, standalone OS (not FreeRTOS) and the Zynq Zybo Board with the PmodAD1 ADC and PmodDA2 DAC external peripherals and LEDs and switches. Lab 7 is a predecessor and can form the basis for Lab 8. The PmodAD1 ADC and PmodDA2 DAC program *AD1DA2Pmod.c* from Lab 7 is provided as a template on Canvas and shown below.

The project is to implement the FIR digital filter with constant coefficients and the transfer function H(z) given by:

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

The coefficients are unique: b_o is the least significant digit (LSD) of your Temple University ID *modulo* 2 *plus* 1, b_1 is the second LSD *modulo* 3 *plus* 1, b_2 is the third LSD *modulo* 4 *plus* 1, b_3 is the fourth LSD *modulo* 5 *plus* 1 and b_4 is the fifth LSD *modulo* 6 *plus* 1.

The term *plus* 1 is there so that if the *modulo* operation produces 0 the coefficient would be at least 1. For example, if the five least significant digits (left to right) are 68324 then $b_0 = 1$, $b_1 = 3$, $b_2 = 4$, $b_3 = 3$, $b_4 = 5$.

An example FIR digital filter transfer function H(z) and discrete output equation for y(m) in terms of the x(m)'s would be:

$$H(z) = 1 + 3 z^{-1} + 4 z^{-2} + 3 z^{-3} + 5 z^{-4}$$

$$y(m) = x(m) + 3\,x(m-1) + 4\,x(m-2) + 3\,x(m-3) + 5\,x(m-4)$$

The project task is to implement the FIR digital filter as a *standalone*, C language processing task in Xilinx SDK with hardware design blocks in Vivado for the PmodDA1 ADC, the PmodDA2 DAC and GPIOs for SWs and LEDs in Vivado.

Note that the start addresses for the PmodAD1, PmodDa2 and the GPIO(s) for the SWs and LEDs must be ascertained from the Address Editor in Vivado and included in your application program modified from the template program *AD1DA2Pmod.c.*

- The PmodAD1 is unipolar and must be offset by −2048 as in the template program *AD1DA2Pmod.c.* The worst-case output of the FIR filter should be assessed and is required fit as a signed 16-bit integer.

  The worst-case output would occur if the ADC input is always +2047 or −2048 and for the FIR example here is approximately 2048(1+3+4+3+5) = 32768. A signed 16-bit integer ranges from −32768 to 32767 and scaling of the ADC is required here, although your model may or may not require that. This determination must be included in the Laboratory Report.

- The constraint file *AD1DA2JE.xdc* is provided on Canvas. Download and add this file to your project. This file must be edited to enable the required SWs and LEDs for this Laboratory. Uncomment all the appropriate lines for the SWs and LEDs by removing the # comment mark and save the file in your project.

- The 12-bit integer output for the PmodDA2 DAC must be offset by +2048 to accommodate the bipolar output voltage signals as in the template program *AD1DA2Pmod.c*. The unipolar DAC has a range of 0 to 4095. The slide switches (SW) are to be used to set the gain ranges for the DAC output.

  The SWs when ON are to have a percentage attenuation as follows for the example output as a percentage of worst-case. Your model would require different gains for the same percentage attenuations.

SW0 = 2047/26625	≈ 0.07688	100%
SW1 = 1024/26625	≈ 0.03846	50%
SW2 = 410/26625	≈ 0.01540	20%
SW3 = 205/26625	≈ 0.00770	10%

  Note that the scale factor must be a floating-point calculation, then converted back to integer for DAC output. However, the FIR digital filter output can be accomplished as signed integers. The ADC may be required to be scaled as described above. The template program *AD1DA2Pmod.c* does not include ADC scaling or SW operation for attenuation. Note also that the *m* library for *math.h* should be included for floating point math.

  All SWs OFF should be a gain of 0 for a zero output (2048 −> V/2 ≈ 1.65 V). A *lock-out* should be provided so that only the first slide switch is read and processed. This complete procedure must be included in the Laboratory Report.

- The LEDs should provide feedback for the operation of the FIR digital filter as follows:

  LED0 Positive saturation of the ADC input has occurred ($V_{in} \geq V/2$)
  LED1 Negative saturation of the ADC input has occurred ($V_{in} \leq V/2$)
  LED2 Positive saturation of the DAC output has occurred ($V_{out} \geq V/2$)
  LED3 Negative saturation of the DAC output has occurred ($V_{out} \leq V/2$)

  A small input overload at 100% attenuation (SW0) can be used to test and demonstrate that these specification features work. Note that the template program *AD1DA2Pmod.c* also does not include LED operation.

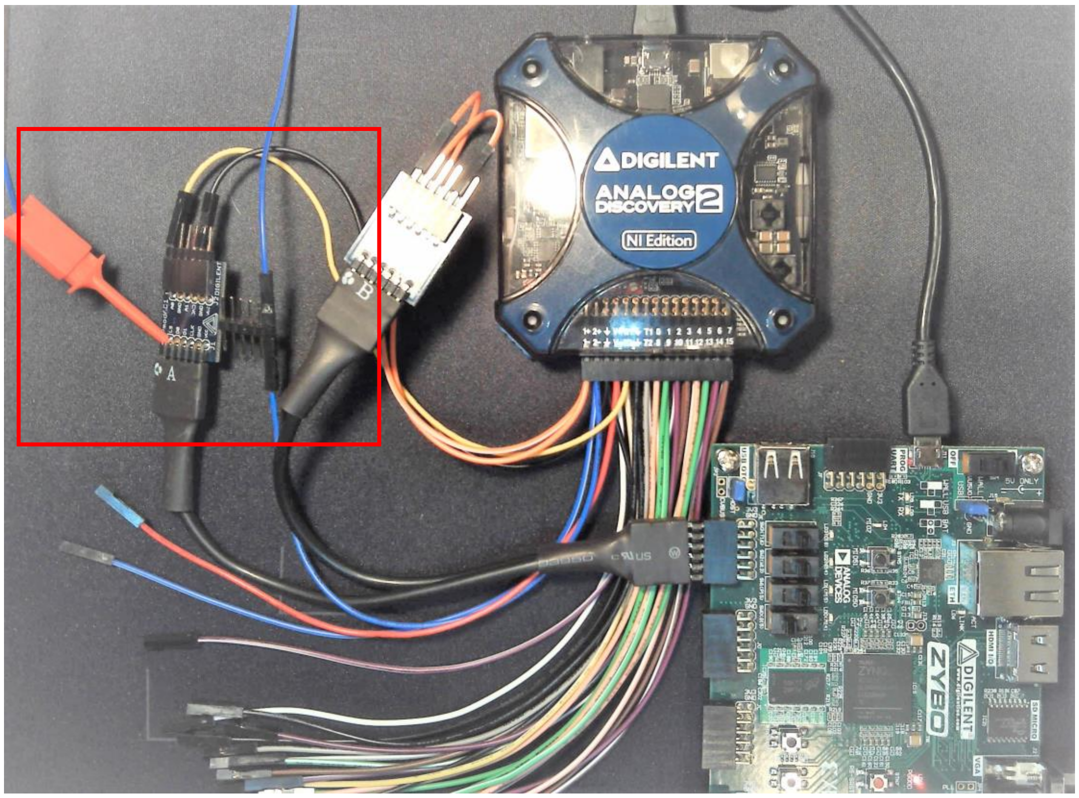
- The sampling rate f_s of the complete Vivado and SDK hardware and software design must be measured before the actual frequency response of the FIR digital filter can be assessed. You can monitor the PmodAD1 CS SPI bus signal on the PmodAD1 pin 1 for an indication of the initial sampling rate f_s.

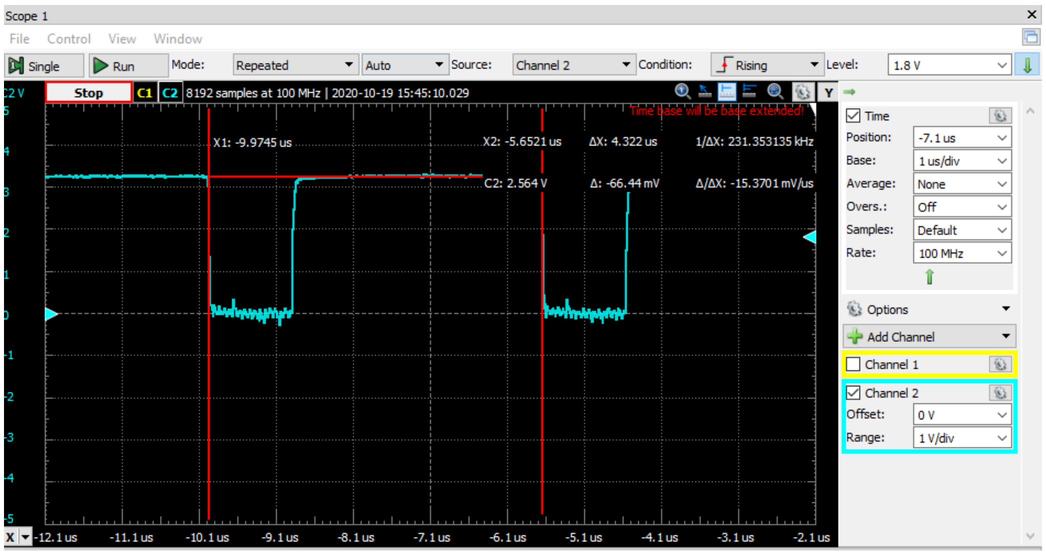
  Shown below is the measurement configuration using the Waveform channel 2. A single blue cable for channel 2 is sufficient since ground is already connected with the black wire for the ADC input. The yellow cable is the Wavegen output of the Discovery 2.

  Once measured insert *software delays* to *round-off* the sampling rate to a practical whole number. Here perhaps 225 kHz would be reasonable. Measure, record and utilize this operational sampling rate as f_s and it must be delineated in the Laboratory Report.

- To test the response of the FIR digital filter, you are to first calculate the magnitude of the frequency response using the sampling rate f_s utilized. You could use EDA tools such as MATLAB to facilitate the process. Plot the Z-plane pole-zero configuration for the Laboratory Report. Can you infer the magnitude of the frequency response from this plot?

  Shown below is the measured active low pulse interval of the CS pulse at pin 1 of the PmodAD1 as 4.322 usec or a sampling rate $f_s \approx 231.35$ kHz. This implies that the Nyquist frequency to avoid aliasing here is $f_s/2 \approx 115.68$ kHz. Your model may have another value of f_s. This measurement must be delineated in the Laboratory Report.





Mode: Repeated Auto Source: Channel 2 Condition: Rising Level: 1.8 V

8192 samples at 100 MHz | 2020-10-19 15:45:10.029

Time base will be base extended!

X1: -9.9745 us X2: -5.6521 us ΔX: 4.322 us 1/ΔX: 231.353135 kHz

C2: 2.564 V Δ: -66.44 mV Δ/ΔX: -15.3701 mV/us

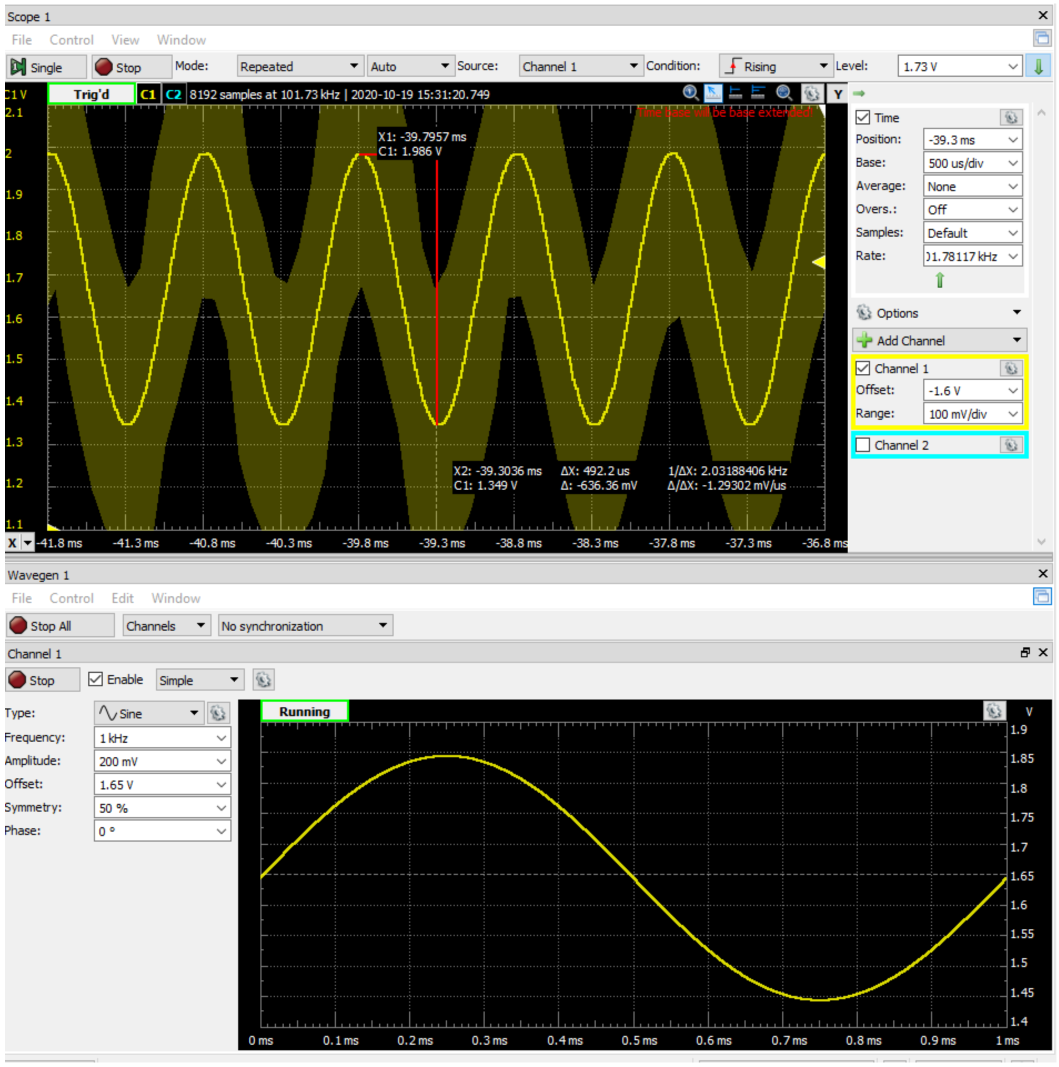
Time
Position: -7.1 us
Base: 1 us/div
Average: None
Overs.: Off
Samples: Default
Rate: 100 MHz

Channel 1

Channel 2
Offset: 0 V
Range: 1 V/div

- With the Discovery 2 as a sinusoidal frequency generator with the yellow and black cables and oscilloscope channel 1 with the orange and orange-white cables as shown above, measure the peak-to-peak amplitude of the actual response from the DAC at several frequencies. Obtain its ratio to the input peak-to-peak amplitude to verify that it nominally agrees with the calculated response at several frequencies. This is a FIR filter gain as a function of frequency | H(z) |.

Shown below is the Waveforms display of the channel 1 Scope and the Wavegen at nominally 1 kHz and a peak-to-peak input amplitude of 200 mV. The peak-to-peak output amplitude is measured with cursors to be (1.986−1.394) ≈ 0.592 V (592 mV). The measured magnitude of the frequency response then at 1 kHz is 592/200 = 2.96.



Mode: Repeated | Auto | Source: Channel 1 | Condition: Rising | Level: 1.73 V

Trig'd | C1 | C2 | 8192 samples at 101.73 kHz | 2020-10-19 15:31:20.749

X1: -39.7957 ms
C1: 1.986 V

X2: -39.3036 ms
C1: 1.349 V

ΔX: 492.2 us
Δ: -636.36 mV

1/ΔX: 2.03188406 kHz
Δ/ΔX: -1.29302 mV/us

Time
Position: -39.3 ms
Base: 500 us/div
Average: None
Overs.: Off
Samples: Default
Rate:)1.78117 kHz

Channel 1
Offset: -1.6 V
Range: 100 mV/div

Channel 2

No synchronization

Channel 1
Enable | Simple

Type: Sine
Frequency: 1 kHz
Amplitude: 200 mV
Offset: 1.65 V
Symmetry: 50 %
Phase: 0 °

Running

- Tabulate or plot the actual response and the predicted response as a function of frequency. Such frequency response plots are performed at standard multiples of 1, 2, 5, 10 (for example 10, 20, 50, 100, 200, 500, 1000…) Hz. You are to measure and plot the low frequency gain and *rolloff* of the actual frequency response of the FIR digital filter in the Laboratory Report.

Ensure that the input sinusoidal signal is properly biased (centered near to V/2 ≈ 1.65 V) and ranged so that you do not overdrive the input signal to the ADC and cause saturation or possible damage. Frequency rolloff is assessed by noting the frequency for which the gain is –3 dB (0.707) of the low frequency voltage response. If a gain of –3 dB cannot be achieved for your particular FIR filter at some frequency then another threshold (for example, –1 dB) can be used. The frequency rolloff point and the dB threshold used is to be described in the Laboratory report.

- What happens to a sinusoidal output as observed with the DAC if the frequency exceeds the Nyquist frequency $f_S/2$?

The completed Laboratories should be archived on your laptop and will form the basis of SNAP Quizzes and Exams. You are to use the *Project Report Format* posted on *Canvas*. You are to upload your *Report* to Canvas for time and date stamping to avoid a late penalty.

This is a two-week Lab starting March 23rd and March 30th and due no later than April 6th 11:59 PM with upload to *Canvas*.

```
//AD1DA2Pmod.c  PmodAD1 PmodDA2 ADC- FIR Digital Filter Example
//ECE3622    c2020 Dennis Silage

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

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)

        xil_printf("\n\rStarting AD1-DA2 Pmod FIR example...\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
                adcdata1=Xil_In32(AD1dat1);      //input ADC data
                adcdata2=Xil_In32(AD1dat2);
                while (adcdav==1)                //wait for reset
                        adcdav=Xil_In32(AD1dav);

                x1d=x1c;                         //ADC ch1 -> DAC ch1 FIR
                x1c=x1b;
                x1b=x1a;
                adcdata1=adcdata1-2048;          //offset for ADC
                x1a=adcdata1/8;                  //efficient shift right scaling
                y1a=x1a+3*x1b+4*x1c+3*x1d+5*x1e; //example FIR
                dacdata1=y1a+2048;               //offset to restore DAC

                dacdata2=adcdata2           //ADC ch2 -> DAC ch2 straight through

                //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);
        }
}
```