SAN DIEGO STATE UNIVERSITY

Parametric IIR Filtering on an FPGA

Department of Electrical and Computer Engineering

Signal Processing Research laboratory

David Walker-Howell and Ashkan Ashrafi

Table of Contents

Introduction

This paper presents a tutorial for a real-time parametric IIR filter implementation on the Xilinx Virtex-5 FPGA ML506 Evaluation Platform. The FPGA implements a biquad IIR filter architecture with the ability to dynamically load new filter coefficients via serial UART communication. A host PC running a Python-based graphical user interface (GUI) allows real-time control of the filter's parameters such as gain, center-frequency, and bandwidth. In this tutorial, the peaking/notch type filter is used as demonstration for its practical application in audio equalization. However, this design can be generalized for other filter types like lowpass, highpass, bandpass, etc.

Requirements

The requirements specified are necessary to be able to follow this tutorial step-by-step. However, most of the software components are generalized and modular, and thus could be extended for use with other hardware platforms.

Hardware Requirements

- Xilinx ML506 Evaluation Platform (features Virtex-5 FPGA).
- Xilinx Platform Cable USB II
- Digilent Pmod DA2 digital-to-analog converter (DAC).
- Digilent Pmod AD1 analog-to-digital converter (ADC).
- USB to 3.3V UART breakout board.
- Waveform function generator.
- Oscilloscope (2+ channels).

Software Requirements

- ISE Design Suite 14.7 (Version used in the making of this tutorial. Other versions may be possible to use.)
- Python 3.7.1 (with the following library dependencies installed).
  - Numpy
  - PyQt5
  - pyqtgraph
  - PySerial

Theory

IIR filters are advantageous for digital filtering applications requiring variety of frequency response selectability. In general, the filters magnitude response can be met more efficiently compared to the Finite Impulse Response (FIR) filters [1]. Due to the IIR filters design process, which derives from continuous-time filter designs, the generation of filter coefficients can be computed using closed form equations. This leads to fast, non-iterative computer programs for generating the filter coefficients, which is desired for adjustments of the frequency response in real-time.

A useful application for a parametrically adjustable IIR filter is an audio equalizer. Audio equalizers, often used in music applications, boost or attenuate specific frequency bands. These equalizers require filter banks of parametrically adjustable filters to be able to control gain, center frequency, and bandwidth. The filters in an equalizer have a peaking or notch type filter form to allow for unity gain passing of the input signal at the frequency bands not being adjusted **Figure 1**. The architecture and hardware implementation of the parametric IIR filter presented in this tutorial demonstrates the concepts of a digital hardware-based equalizer.

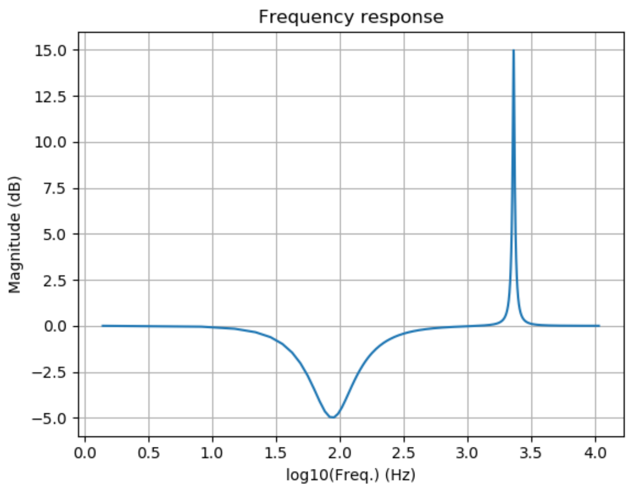


Figure 1*: Example of parametric equalizer frequency response using peaking/notch filter.*

The biquad filter is used in this project because it is a second order filter that allows for a quality magnitude response with only two poles and two zeros. A basic form of the filter, called Direct Form I, is given as

Frequency Response $$H(z) = \frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}}$$

Impulse Response $$y[k] = b_0x[k] + b_1x[k-1] + b_2x[k-2] - a_1y[k-1] - a_2y[k-2]$$

The five coefficients are computed using a set of closed form equations which are given in [2]. Note, depending on the type of filter used, there are different sets of closed form equations that can be derived.

Since the IIR filter is implemented in the digital hardware of the FPGA, the coefficients cannot be infinite precision floating-point. The IIR filter implementation used in this project requires the coefficients to be fixed-point two's complement binary. The conversion from floating-point to fixed-point introduces round-off error that can make the filter response significantly differ from the original design or even cause the filter to become unstable [3]. As an example, if the coefficients of the poles are too close to the unit circle, then the conversion to fixed-point might cause the poles to go outside the unit circle, resulting in an unstable filter. There are a few ways to help the errors be less significant.

1. Use as many bits as possible for fixed-point representation.
2. Make the number of fractional bits only 1-2 less than the total number of bits. For example, if 16-bits are used for representation, make 14-bits the fractional part.
3. Normalize the coefficients and signals for filtering to mitigate overflow.

In this project, the coefficients and signals are represented with a total width of 16-bits and a fractional width of 14-bits.

System Design

The architecture of the FPGA implementation is broken up into multiple modules. See **Figure 2**.

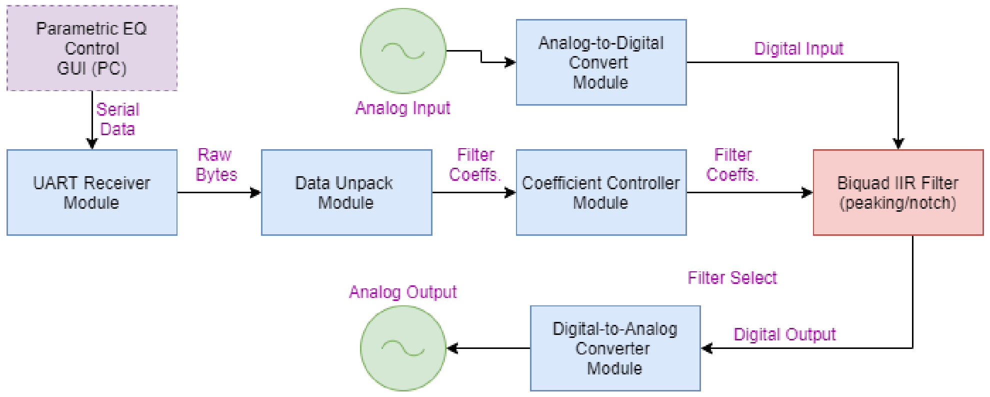
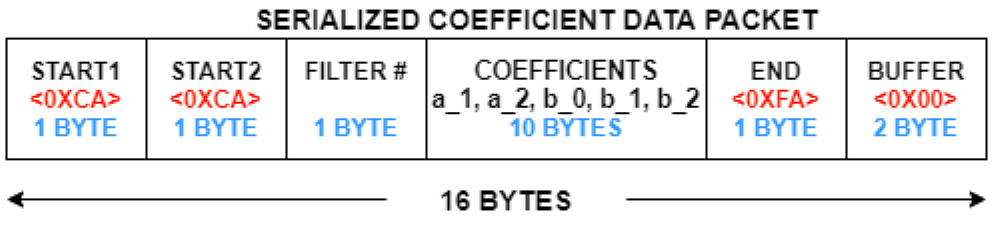

Parametric EQ Control GUI (PC)
Serial Data
UART Receiver Module
Raw Bytes
Data Unpack Module
Filter Coeffs.
Coefficient Controller Module
Filter Coeffs.
Biquad IIR Filter (peaking/notch)
Analog Input
Analog-to-Digital Convert Module
Digital Input
Filter Select
Digital Output
Digital-to-Analog Converter Module
Analog Output

Figure 2: *System block diagram.*

First, the desired filter response is designed via the Python GUI on the host PC. The filter coefficients are generated and serialized to be sent to the FPGA via UART serial communication. The Verilog UART Receiver Module reads in 10-bit (1 start bit, 8 data bits, 1 stop bit) data packets and extracts the data. This data is then sent to the Data Unpack Module to check which coefficient data the individual bytes correspond to. **Figure 3** shows what a valid coefficient data packet looks like.



SERIALIZED COEFFICIENT DATA PACKET

START1	START2	FILTER #	COEFFICIENTS	END	BUFFER
<0XCA>	<0XCA>		a_1, a_2, b_0, b_1, b_2	<0XFA>	<0X00>
1 BYTE	1 BYTE	1 BYTE	10 BYTES	1 BYTE	2 BYTE

16 BYTES

Figure 3: *Coefficient data packet that is unpacked and validated by the Data Unpack Module.*

Once the coefficients are successfully unpacked, the coefficients are moved into a simple RAM block in the Coefficient Controller Module. The Coefficient Controller Module is responsible for timing when to change the coefficients of filter to ensure the change in frequency response is fast and stable. Coefficients are changed in the filter when the filter module has finished its accumulation step and before the next input is sampled. Finally, the Biquad IIR Filter Module samples the inputs from the ADC at every sample clock, performs a multiplication and accumulation sequence, then sends the output to the DAC. **Figure 4** shows the clocking of the IIR filter computations with respect to the sample clock and FPGA's system clock.

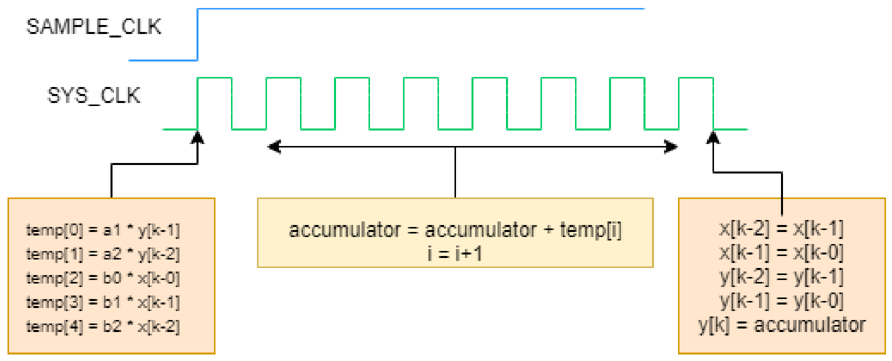


Figure 4: *Verilog IIR filtering computation stages.*

Hardware Setup

USB UART Serial Breakout

UART communication is used to transmit the filter coefficients from the host PC to the FPGA. A USB to UART converter breaks out the USB data lines to I/O UART TX/RX pins that can be connected to the FPGA via GPIO. In this project, the SparkFun USB UART Serial Breakout - CY7C65213 [4] was used. Any USB to UART breakout board will work fine for this project. Make sure the logic voltages match; for the ML506 Evaluation Board, the single-ended GPIO voltage logic is 3.3V.

1. Connect the host PC to the converter with an appropriate USB cable.
2. Connect the ground pin on the converter to the ground on the FPGA.
3. Connect the transmit pin (TX) on the converters to pin number 2 (HDR1_2) on the FPGA's singled-ended GPIO.

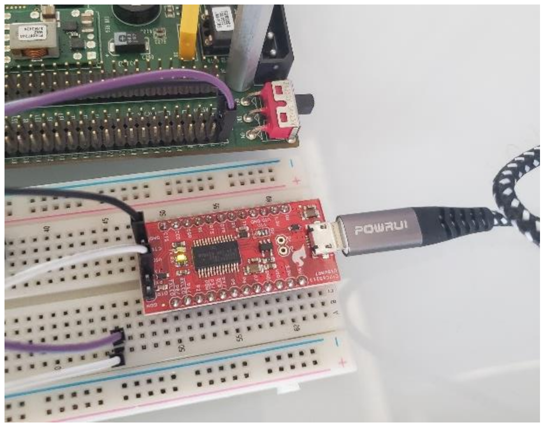


Figure 5: *USB to UART breakout board connect to the Virtex-5 single-ended GPIO.*

FPGA Pin Name	USB/UART Converter Pin Name
HDR1_2	TX
GND	GND

Table 1: *Virtex-5 FPGA GPIO connection with the USB to UART breakout board.*

Digilent Pmod DA2 DAC

The Digilent Pmod DA2 12-bit, 2 channel DAC [5] converts the digital output signal from the IIR filter to an analog signal that can be observed with an oscilloscope. The Pmod DA2 communicates with the FPGA logic through the single-ended GPIO pins. The communication protocol is similar to SPI, where the 12-bit resolution digital signal is synchronously sent from the FPGA to the DAC.

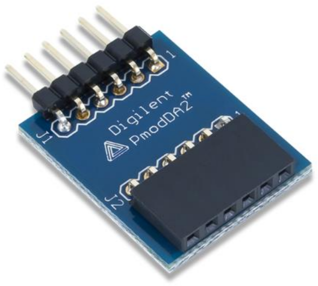


Figure 6: *The Digilent Pmod DA2 12-bit, 2 channel digital-to-analog converter. Source: [5].*

Figure 7 shows the GPIO to DAC connections on the FPGA. The right-most column of pins is the single-ended GPIO pins. The middle column are all ground pins.

1. Connect the DA2 pins (1, 2, 3, 4) to FPGA'S GPIO pins (4, 6, 8, 10) respectively.
2. Connect the DA2 VCC pin to a 3.3V power pin on the FPGA.
3. Connect the DA2 GND pin to ground on the FPGA.

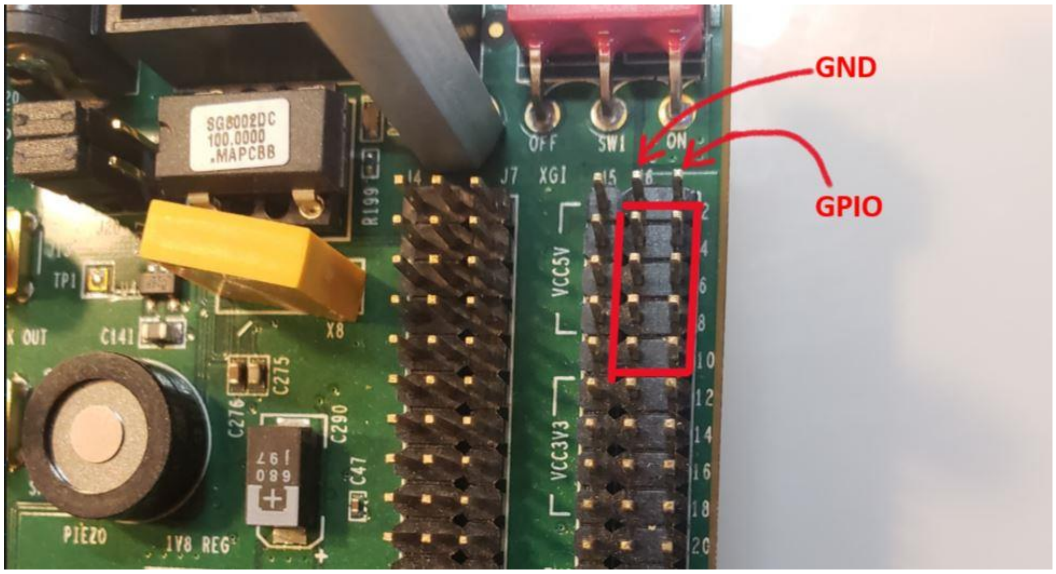


***Figure 7**: The highlighted GPIO pins to connect the Digilent Pmod DA2.*

FPGA Pin Name	Pmod DA2 Pin Name	Pmod DA2 Pin Number
HDR1_4	~SYNC	1
HDR1_6	DINA	2
HDR1_8	DINB	3
HDR1_10	SCLK	4
GND	GND	5
VCC3V3	VCC	6

***Table 2**: Virtex-5 FPGA GPIO pin to PMOD DA2 pin connections.*

The connection order for the DAC is shown in **Table 2**. Male-to-female jumper wires are suggested.

Digilent Pmod AD1 ADC

The Digilent Pmod AD1 12-bit, 2 channel ADC [6] also provides an SPI-like communication protocol with up to 1MSa of 12-bit conversion. It's operation and connection type are similar to the Pmod DA2 DAC.

1. Connect the AD1 pins (1, 2, 3, 4) to FPGA'S GPIO pins (12, 14, 16, 18) respectively.
2. Connect the AD1 VCC pin to a 3.3V power pin on the FPGA.
3. Connect the AD1 GND pin to ground on the FPGA.

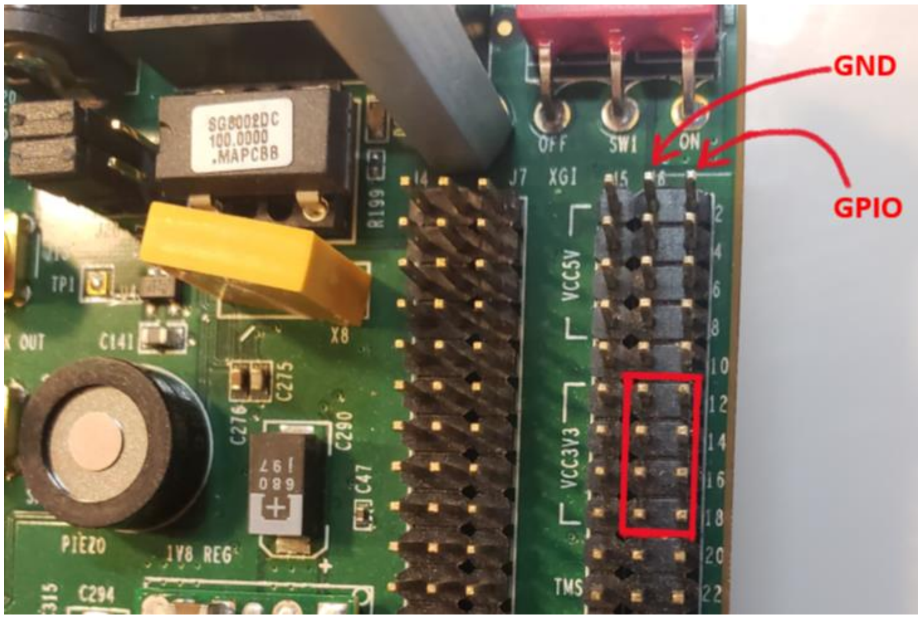


Figure 8: *The highlighted GPIO pins to connect the Digilent Pmod AD1.*

FPGA Pin Name	Pmod AD1 Pin Name	Pmod AD1 Pin Number
HDR1_12	CS	1
HDR1_14	D0	2
HDR1_16	D1	3
HDR1_18	SCK	4
GND	GND	5
VCC3V3	VCC	6

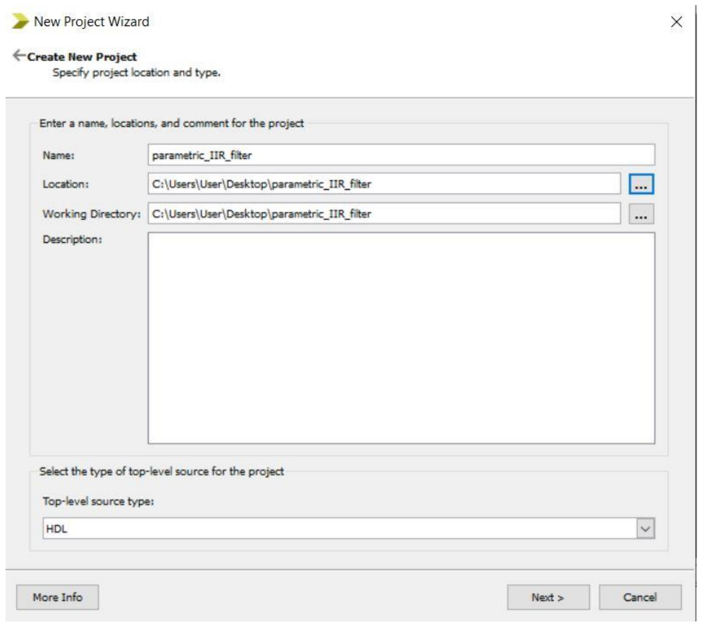
Table 3: *Virtex-5 FPGA GPIO pin to PMOD DA2 pin connections.*

The connection order for the ADC is shown in **Table 3**. Male-to-female jumper wires are suggested.

Implementation

To implement the digital system on the FPGA, Xilinx ISE Design Suite 14.7 is used.

1. In a desired file location to store the project files. Open Xilinx ISE Design Suite. Under **File**, click **New Project**. The project wizard in **Figure 9** will appear. Give the project a name and save it in any desired location. Make sure the **Top-level source type** is HDL. Finally click **Next**.

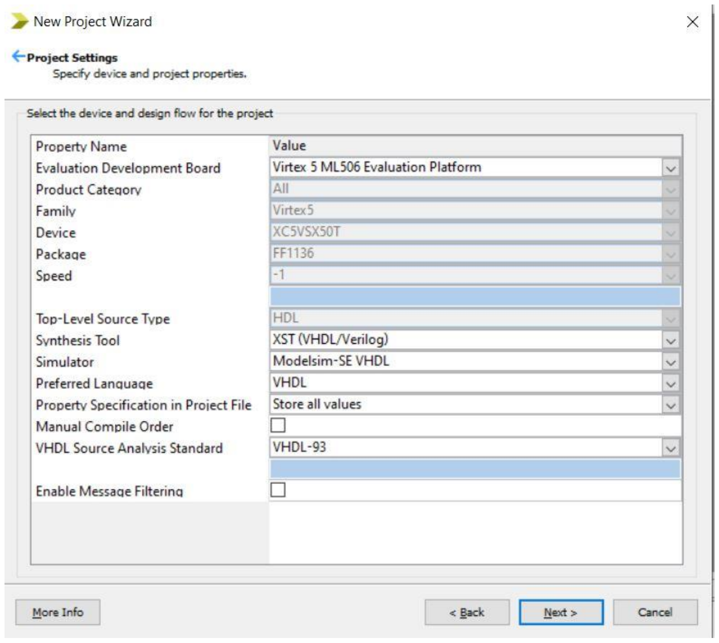

New Project Wizard
Create New Project
Specify project location and type.
Enter a name, locations, and comment for the project

Name: parametric_IIR_filter
Location: C:\Users\User\Desktop\parametric_IIR_filter
Working Directory: C:\Users\User\Desktop\parametric_IIR_filter
Description:

Select the type of top-level source for the project
Top-level source type: HDL

***Figure 9**: New Project Wizard.*

2. Next, the project settings, shown in **Figure 10** will be set to ensure the Xilinx software configures to the correct FPGA hardware. In the **Evaluation Development Board** dropdown select **Virtex 5 ML506 Evaluation Platform**; this will automatically fill-in the correct properties of **Product Category**, **Family**, **Device**, **Package**, and **Speed**. The rest of the settings should match **Figure 10**. Click **Next**, then **Finish**.


New Project Wizard
Project Settings
Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	Virtex 5 ML506 Evaluation Platform
Product Category	All
Family	Virtex5
Device	XC5VSX50T
Package	FF1136
Speed	-1
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE VHDL
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

***Figure 10**: Project property settings.*

3. The ZIP folder **Parametric_IIR_Filter_FPGA.zip** contains the Verilog and Constraints files for this project. Unzip this folder to a suitable location. As shown in **Figure 11**, select the **Project** tab, then click **Add Copy of Source** to add all the files to the project. Select all the unzipped files shown in **Figure 12**. Click **Open**, then **Next**.

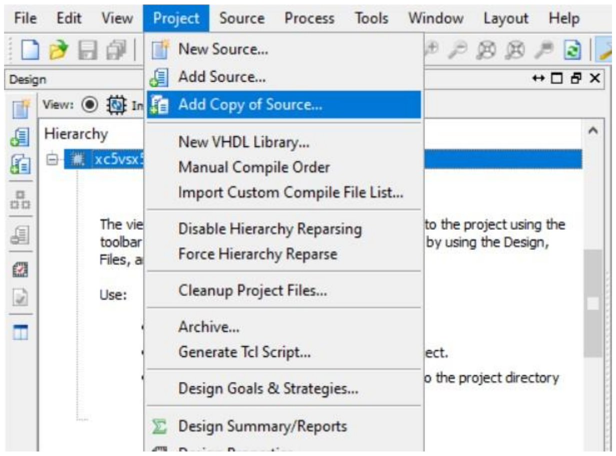
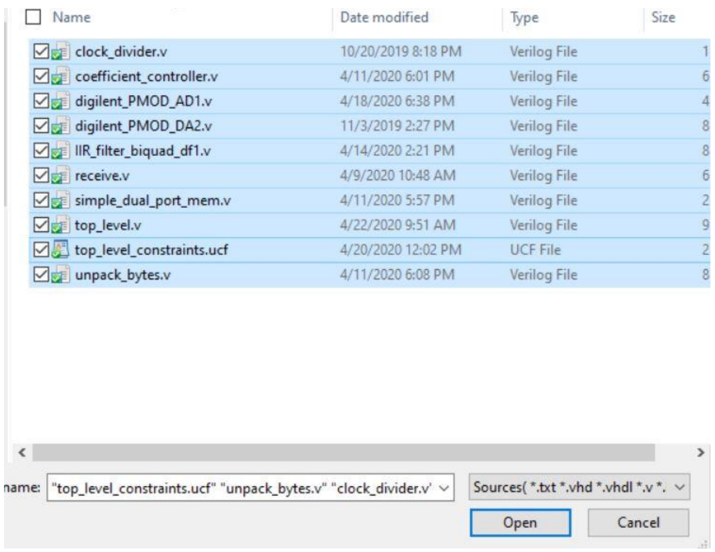


***Figure 11**: Add copy of source files to project.*



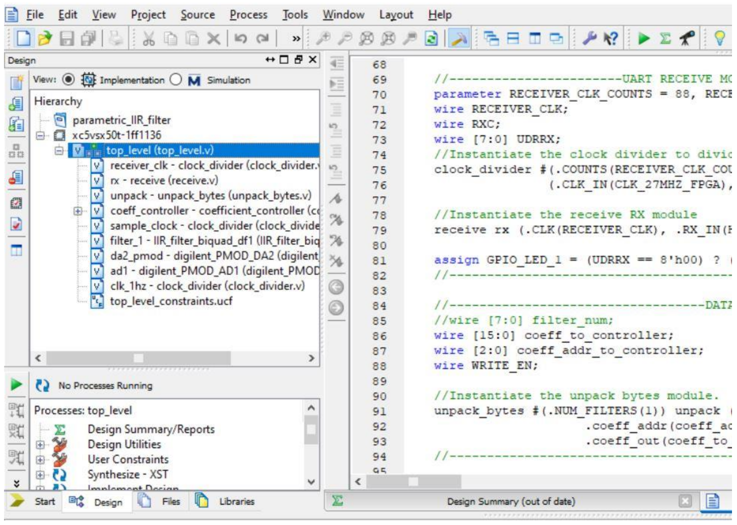
Name	Date modified	Type	Size
clock_divider.v	10/20/2019 8:18 PM	Verilog File	1
coefficient_controller.v	4/11/2020 6:01 PM	Verilog File	6
digilent_PMOD_AD1.v	4/18/2020 6:38 PM	Verilog File	4
digilent_PMOD_DA2.v	11/3/2019 2:27 PM	Verilog File	8
IIR_filter_biquad_df1.v	4/14/2020 2:21 PM	Verilog File	8
receive.v	4/9/2020 10:48 AM	Verilog File	6
simple_dual_port_mem.v	4/11/2020 5:57 PM	Verilog File	2
top_level.v	4/22/2020 9:51 AM	Verilog File	9
top_level_constraints.ucf	4/20/2020 12:02 PM	UCF File	2
unpack_bytes.v	4/11/2020 6:08 PM	Verilog File	8

name: "top_level_constraints.ucf" "unpack_bytes.v" "clock_divider.v"

Sources(*.txt *.vhd *.vhdl *.v *.

***Figure 12**: All the necessary source files for implementation.*

4. Make sure the module **top_level.v** is set as the top level module. A top level module is signified by 3 squares with one highlighted green and have all of the other modules nested underneath it. See **Figure 13**.



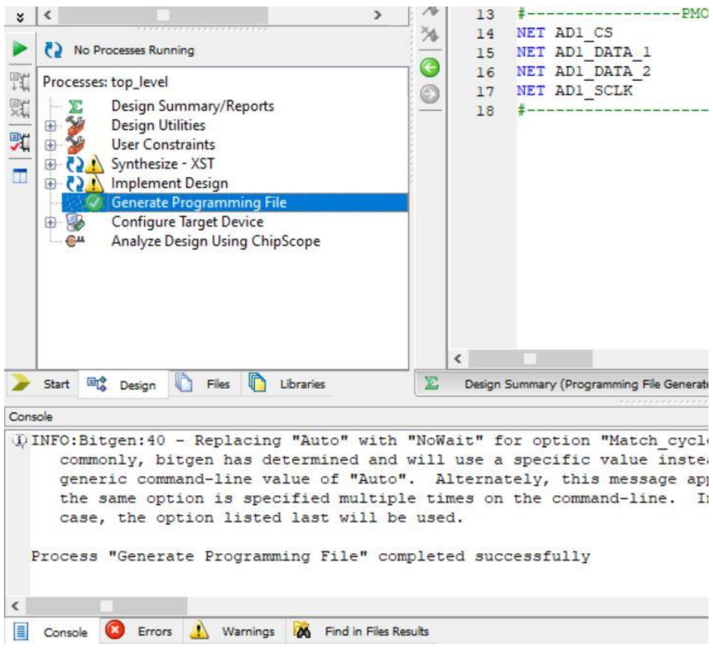
Hierarchy
- parametric_IIR_filter
- xc5vsx50t-1ff1136
  - top_level (top_level.v)
    - receiver_clk - clock_divider (clock_divider.
    - rx - receive (receive.v)
    - unpack - unpack_bytes (unpack_bytes.v)
    - coeff_controller - coefficient_controller (c
    - sample_clock - clock_divider (clock_divide
    - filter_1 - IIR_filter_biquad_df1 (IIR_filter_biq
    - da2_pmod - digilent_PMOD_DA2 (digilent
    - ad1 - digilent_PMOD_AD1 (digilent_PMOD
    - clk_1hz - clock_divider (clock_divider.v)
    - top_level_constraints.ucf

Processes: top_level
- Design Summary/Reports
- Design Utilities
- User Constraints
- Synthesize - XST

```

//-----------------------UART RECEIVE MO
parameter RECEIVER_CLK_COUNTS = 88, RECE
wire RECEIVER_CLK;
wire RXC;
wire [7:0] UDRRX;
//Instantiate the clock divider to divic
clock_divider #(.COUNTS(RECEIVER_CLK_COU
                (.CLK_IN(CLK_27MHZ_FPGA),

//Instantiate the receive RX module
receive rx (.CLK(RECEIVER_CLK), .RX_IN(F

assign GPIO_LED_1 = (UDRRX == 8'h00) ? (
//-------------------------------------

//---------------------------------DATA
//wire [7:0] filter_num;
wire [15:0] coeff_to_controller;
wire [2:0] coeff_addr_to_controller;
wire WRITE_EN;

//Instantiate the unpack bytes module.
unpack_bytes #(.NUM_FILTERS(1)) unpack (
                    .coeff_addr(coeff_ac
                    .coeff_out(coeff_to_
//-------------------------------------
```

***Figure 13:** How to project looks after importing source files.*

5. Lastly, the design is synthesized for the FPGA hardware and a programming file (bitstream) is generated to load the design onto the FPGA. On the bottom-left hand-side in the **Processes** task bar double-click **Synthesize – XST**. It will be noticed that there are numerous warning messages after the synthesis completes; this is okay and should not result in any problems. Next, double-click **Implement Design**. Finally, after the implantation stage is successful, double-click **Generate Programming File**. The result should look similar to **Figure 14**.



```
#----------------PMO
NET AD1_CS
NET AD1_DATA_1
NET AD1_DATA_2
NET AD1_SCLK
#-------------------
```

```
INFO:Bitgen:40 - Replacing "Auto" with "NoWait" for option "Match_cycl
   commonly, bitgen has determined and will use a specific value inste
   generic command-line value of "Auto".  Alternately, this message ap
   the same option is specified multiple times on the command-line.  I
   case, the option listed last will be used.

Process "Generate Programming File" completed successfully
```

Figure 14*: Post successful Synthesis, Implementation, and Bitstream.*

6. Connect the Xilinx Platform Cable USB II to the FPGA and PC. This device is needed to program the bitstream from the PC to the FPGA.

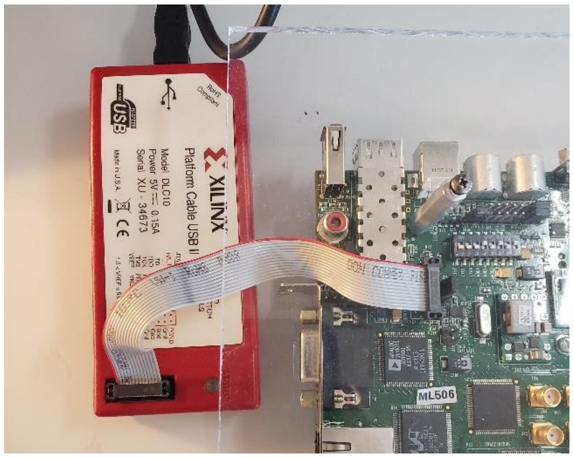


Figure 15*: Xilinx Platform Cable USB II connected.*

7. In Xilinx ISE, select **iMPACT** from the **Tool** dropdown. This will open the iMPACT editor, the tool to load the bitstream to the FPGA. See **Figure 16**. In the iMPACT tool, double-click **Boundary Scan**, follow instructions to **Right click to Add Device**, and click **Initialize Chain.** See **Figure 17**.

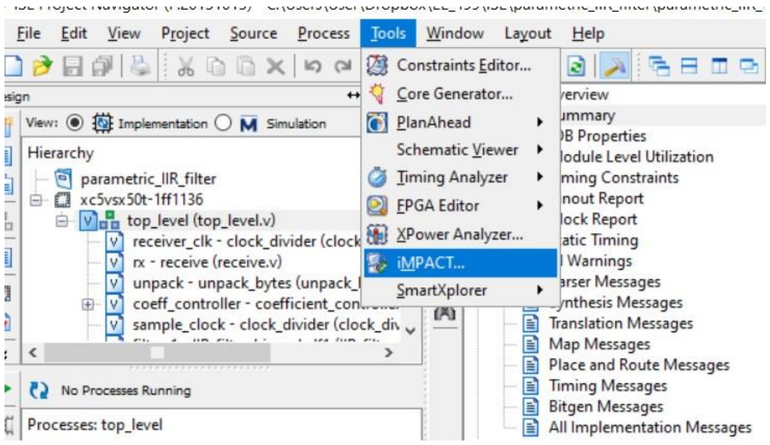


***Figure 16**: Opening ISE iMPACT.*

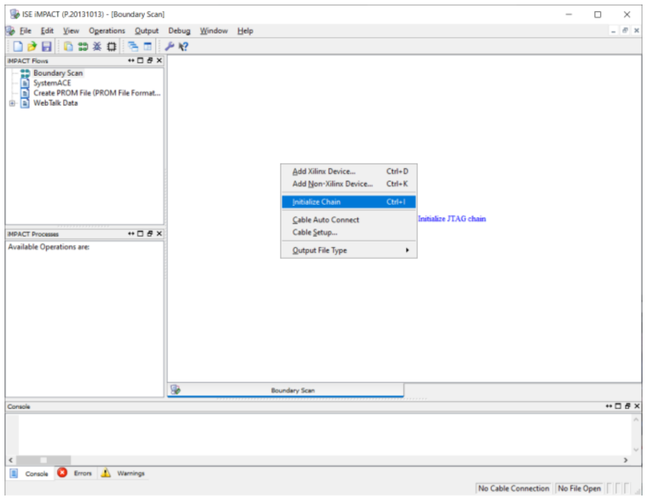
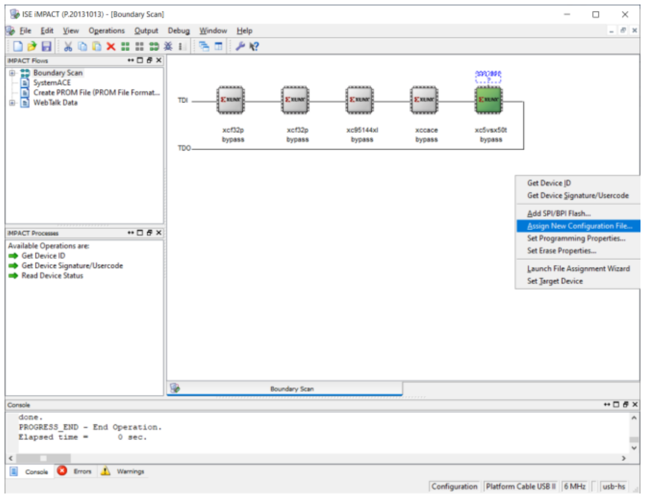


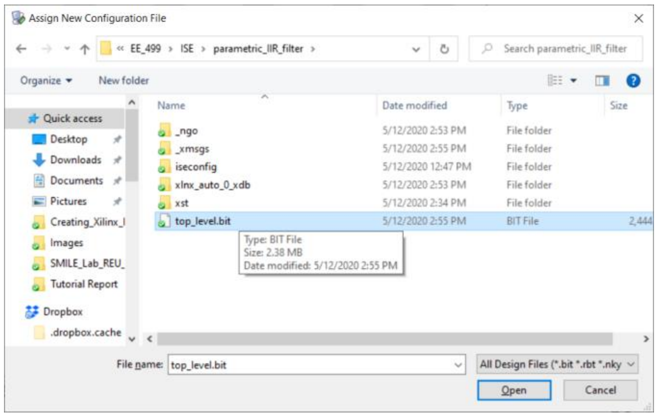
***Figure 17**: Initialize chain to program FPGA.*

8. The last step is to set the Generated Programming File to be the configuration file for the FPGA. Make sure the targeted FPGA device in the tool chain, labeled **xc5vsx50t**, is highlighted green as shown in **Figure 18**. Right click on the device, then choose **Assign New Configuration File.** From the file explore that pops-up, navigate to the main ISE project folder and select **top_level.bit**. See **Figure 19**.



```
done.
PROGRESS_END - End Operation.
Elapsed time =       0 sec.
```

***Figure 18**: Assign New Configuration File.*



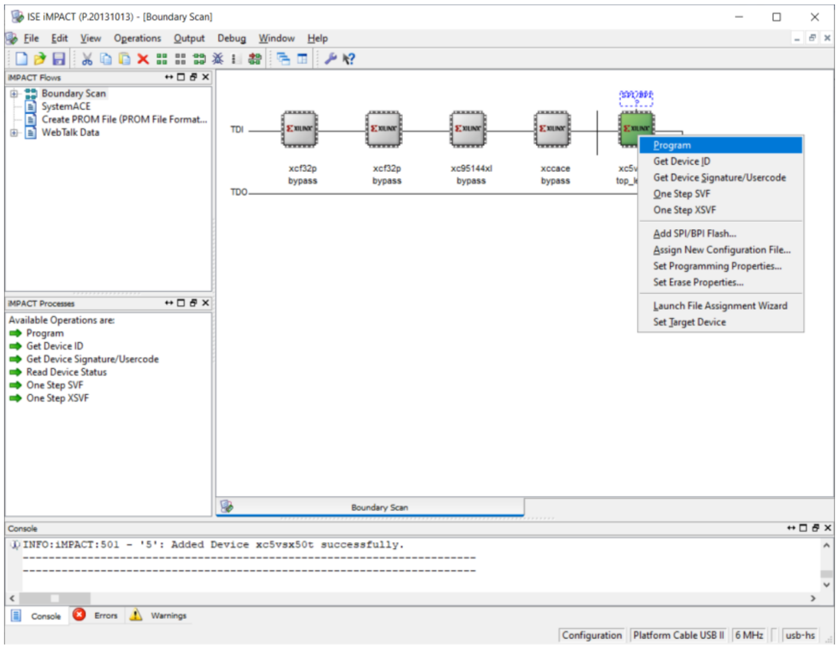
Name	Date modified	Type	Size
_ngo	5/12/2020 2:53 PM	File folder	
_xmsgs	5/12/2020 2:55 PM	File folder	
iseconfig	5/12/2020 12:47 PM	File folder	
xlnx_auto_0_xdb	5/12/2020 2:53 PM	File folder	
xst	5/12/2020 2:34 PM	File folder	
top_level.bit	5/12/2020 2:55 PM	BIT File	2,444

Type: BIT File
Size: 2.38 MB
Date modified: 5/12/2020 2:55 PM

File name: top_level.bit

***Figure 19**: Selecting the generate programming file (.bit) to program the FPGA.*

9. Right click on the device **xc5vsx50t** and click **Program**. See **Figure 20**. Now the FPGA is programmed.

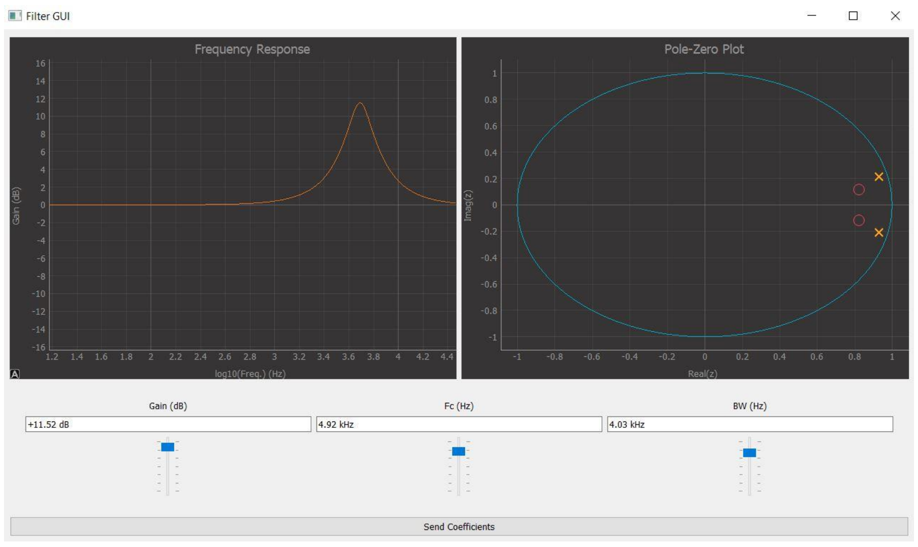


```
INFO:iMPACT:501 - '5': Added Device xc5vsx50t successfully.
----------------------------------------------------------------------
----------------------------------------------------------------------
```

***Figure 20**: Program the generated programming file to the FPGA.*

Graphical User Interface

The graphical user interface (GUI) provides a visual guide to design the filter response, generate the fixed-point coefficients, and communicate those coefficients to the FPGA via serial communication. This GUI is built in Python 3 using the PyQt5 graphics library. Custom class and functions were developed for the fixed-point filter design of the biquad IIR filter.

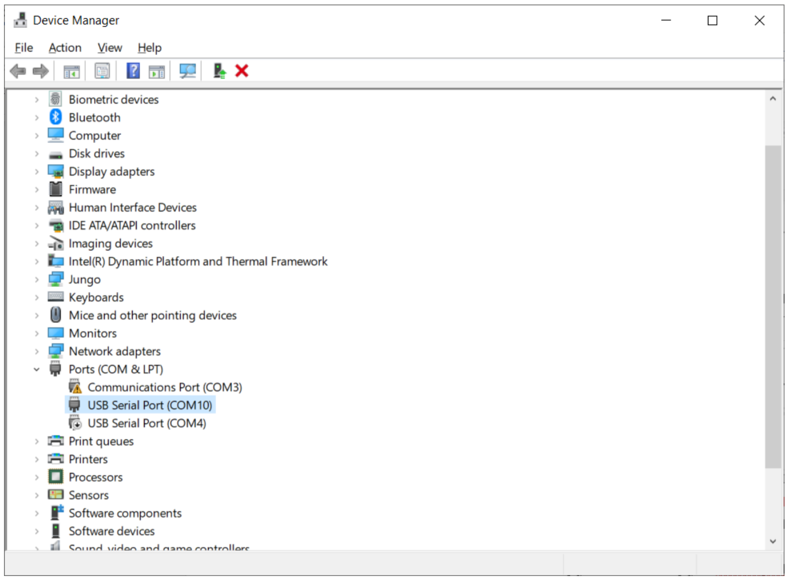


Gain (dB): +11.52 dB
Fc (Hz): 4.92 kHz
BW (Hz): 4.03 kHz

***Figure 21**: Python based GUI.*

Using the GUI

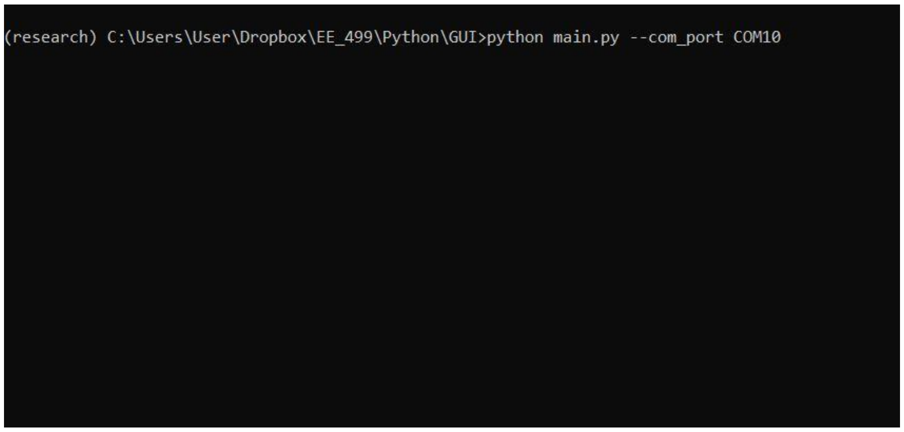
1. In a preferred directory unzip the file **Parametric_IIR_Filter_GUI.zip** to get the GUI source code. This folder contains to subfolders called **PySP** and **GUI**. These two directories must be in the same parent directory. The command line (or terminal) will be used to launch the GUI within the Python environment.
2. Open a command line terminal of choice that can be used to run Python 3 applications with the required libraries (i.e. this may be a virtual environment for some users). Navigate into the **GUI** directory using the "cd" command.
3. Before running the command to start the GUI, it is necessary to know the COM port that the USB UART breakout board is communicating over. For example, if Windows is being used, then the COM port can be found in the **Device Manager** control panel. **Figure 22** shows that the breakout board is communicating on "COM10." For other operating systems finding the communication port will differ.



```
> Biometric devices
> Bluetooth
> Computer
> Disk drives
> Display adapters
> Firmware
> Human Interface Devices
> IDE ATA/ATAPI controllers
> Imaging devices
> Intel(R) Dynamic Platform and Thermal Framework
> Jungo
> Keyboards
> Mice and other pointing devices
> Monitors
> Network adapters
v Ports (COM & LPT)
    Communications Port (COM3)
    USB Serial Port (COM10)
    USB Serial Port (COM4)
> Print queues
> Printers
> Processors
> Sensors
> Software components
> Software devices
> Sound, video and game controllers
```

***Figure 22**: Windows Device Manager to find COM port for USB to UART breakout.*

4. In the terminal run the command **python main.py --com_port [*COM port name*]**, where ***COM port name*** is the name of the communication port discussed in **step 3**. See **Figure 23**.



```
(research) C:\Users\User\Dropbox\EE_499\Python\GUI>python main.py --com_port COM10
```

Figure 23*: Example of command to start GUI.*

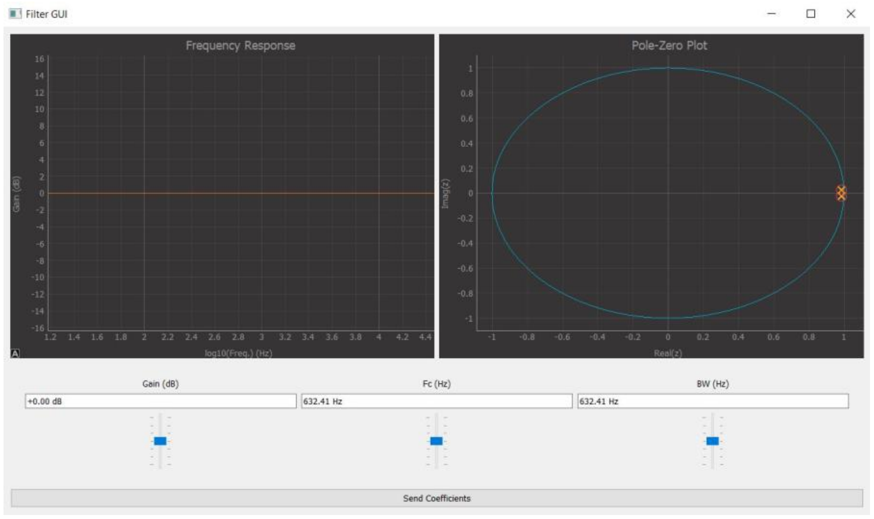


Figure 24*: How the GUI looks when opened.*

5. Adjust the gain, center-frequency, and bandwidth as desired. As the sliders are adjusted the frequency response and pole-zero plot will reflect the expected response of the filter. Once the desired filter response is obtained, click the **Send Coefficients** button to transmit the filter coefficients to the FPGA.

Testing and Verification

The system is ready to be tested. A function generator is used to supply the input signal to the ADC. The Pmod AD1 ADC allows an input voltage range of 0V - 3.3V, so have the function generator output be within this range. Hook the function generators probe to the ADC's analog input on channel 1 (A1). Next, the two channels of the oscilloscope are attached to measure the original input signal and resulting filtered output signal. The comparison is useful to identify that the magnitude frequency response corresponds to the design. One oscilloscope probe is connected to the DAC output on channel 1 to measure the filtered signal and the second oscilloscope probe is connected on channel 2 of the DAC to measure to original signal sampled from the function generator.

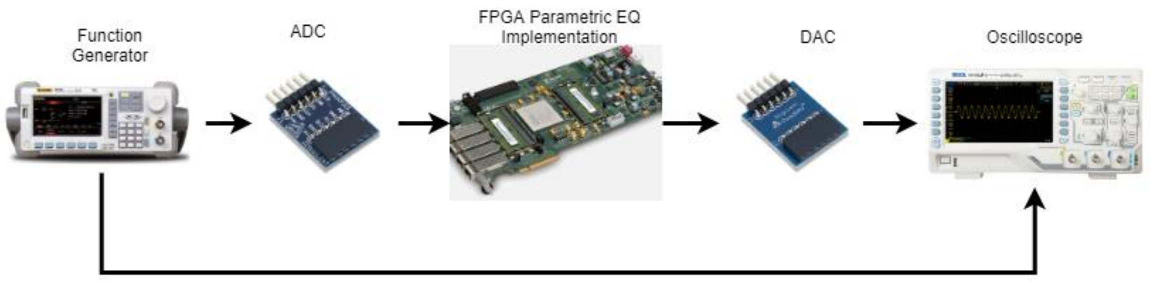


Figure 25: *The testing and verification setup diagram.*

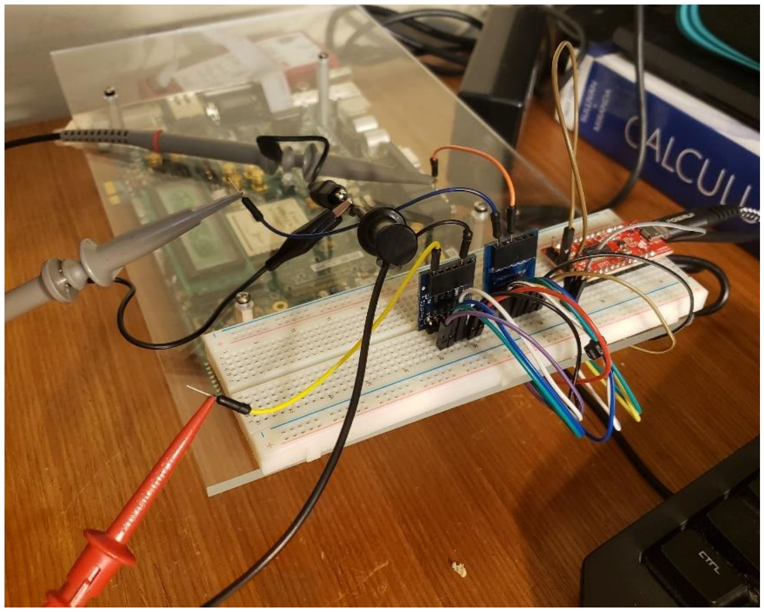
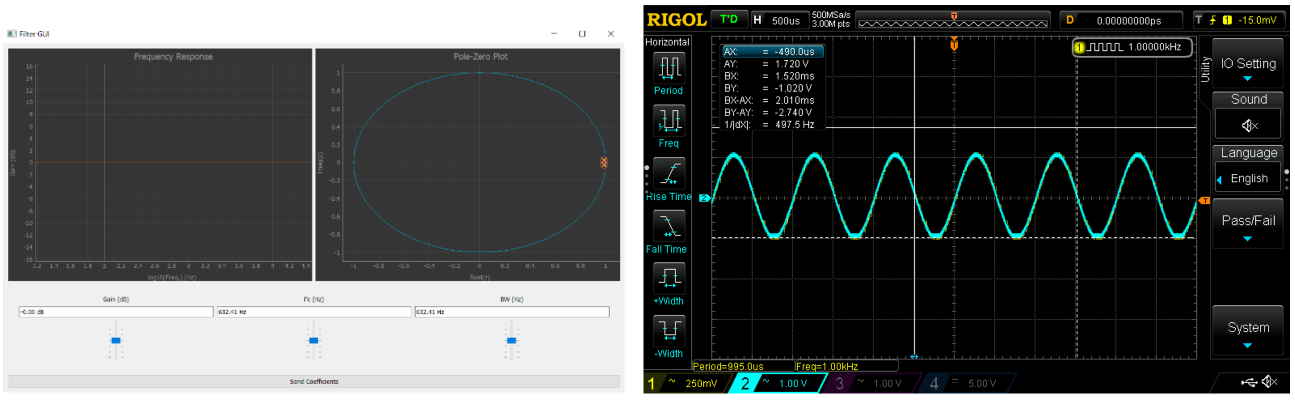


Figure 26: *Test and verification setup.*

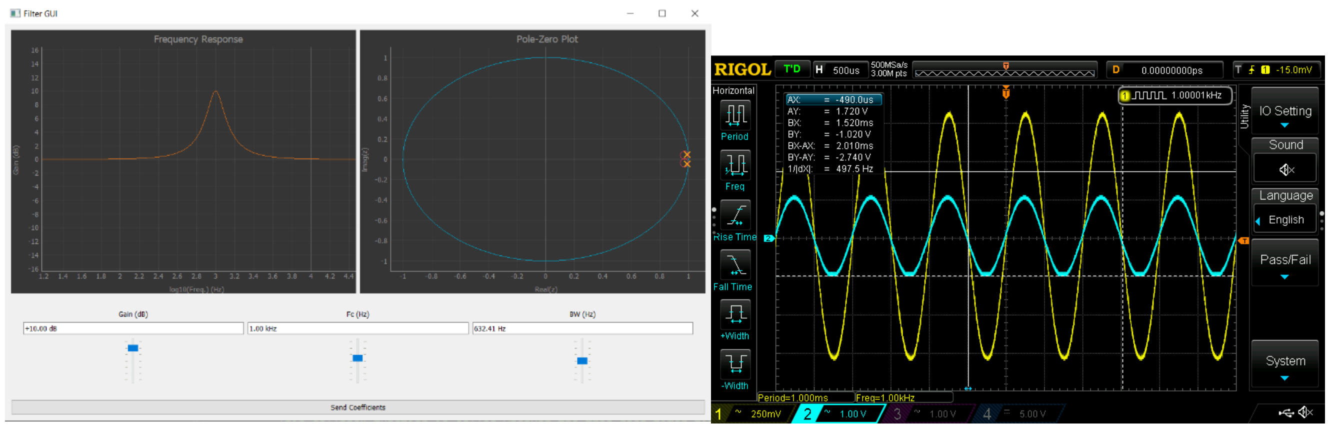
The output from the DAC of the filtered signal will have a magnitude scaled down by a factor of 4. This is ensuring the DAC will never be the source of saturation or encounter overflow that will corrupt the signal. Below are several test cases that verify the desired operation of parametric IIR implementation.



Gain (dB): -0.00 dB
Fc (Hz): 632.41 Hz
BW (Hz): 632.41 Hz

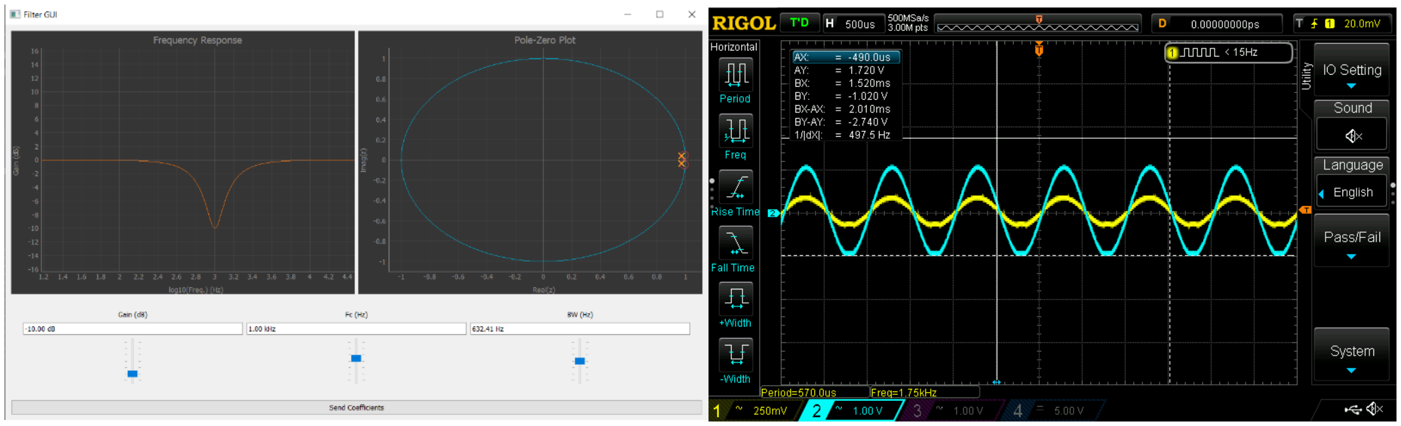
```
T'D  H 500us  500MSa/s 3.00M pts   D 0.00000000ps   T 1 -15.0mV
AX:      = -490.0us
AY:      = 1.720 V
BX:      = 1.520ms
BY:      = -1.020 V
BX-AX:   = 2.010ms
BY-AY:   = -2.740 V
1/|dX|:  = 497.5 Hz
1 1.00000kHz
Period=995.0us   Freq=1.00kHz
1 250mV   2 1.00 V   3 1.00 V   4 5.00 V
```

Figure 27: *All pass, unity gain of signal. The input signal is given in the color blue and the output signal is the color yellow. However, since the filter is unity gain across all frequencies, the signals in the oscilloscope output are nearly on-top of one another.*


Gain (dB) +10.00 dB
Fc (Hz) 1.00 kHz
BW (Hz) 632.41 Hz

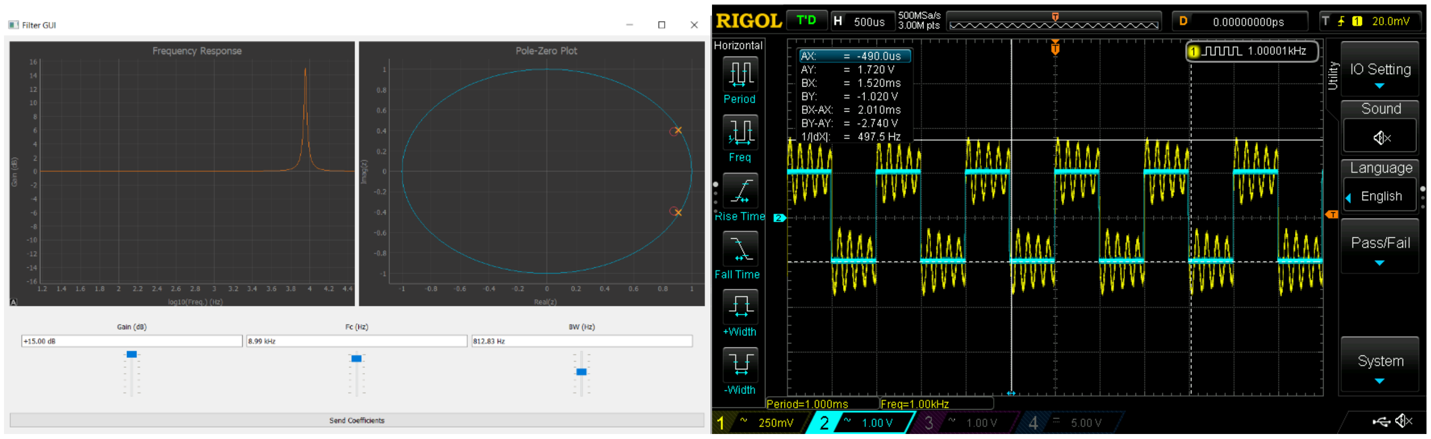
```
AX:     = -490.0us
AY:     = 1.720 V
BX:     = 1.520ms
BY:     = -1.020 V
BX-AX:  = 2.010ms
BY-AY:  = -2.740 V
1/|dX|: = 497.5 Hz
Period=1.000ms   Freq=1.00kHz
```

Figure 28: *The parametric EQ parameters specified were G=+10.0dB, F_c = 1000Hz, BW=632 Hz. The input signal (blue) is a 1000 Hz sine. The output signal (yellow) is the 10dB amplified.*


Gain (dB) -10.00 dB
Fc (Hz) 1.00 kHz
BW (Hz) 632.41 Hz

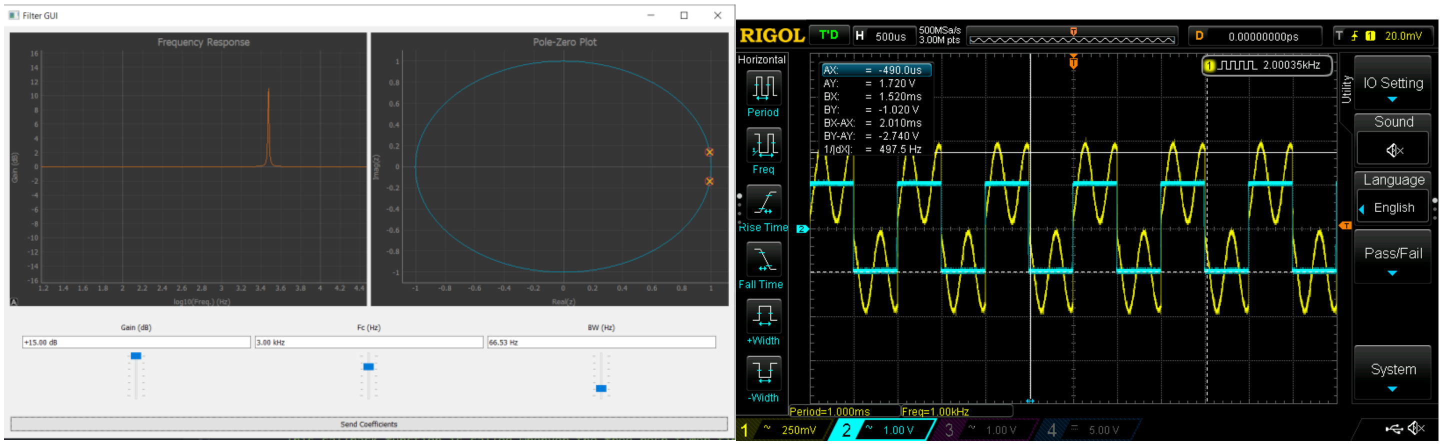
```
AX:     = -490.0us
AY:     = 1.720 V
BX:     = 1.520ms
BY:     = -1.020 V
BX-AX:  = 2.010ms
BY-AY:  = -2.740 V
1/|dX|: = 497.5 Hz
Period=570.0us   Freq=1.75kHz
```

Figure 29: *The parametric EQ parameters specified were G=-10.0dB, F_c = 1000Hz, BW=632 Hz. The input signal (blue) is a 1000 Hz sine. The output signal (yellow) is the 10dB attenuated.*


Gain (dB) +15.00 dB
Fc (Hz) 8.99 kHz
BW (Hz) 812.83 Hz

```
AX:     = -490.0us
AY:     = 1.720 V
BX:     = 1.520ms
BY:     = -1.020 V
BX-AX:  = 2.010ms
BY-AY:  = -2.740 V
1/|dX|: = 497.5 Hz
Period=1.000ms   Freq=1.00kHz
```

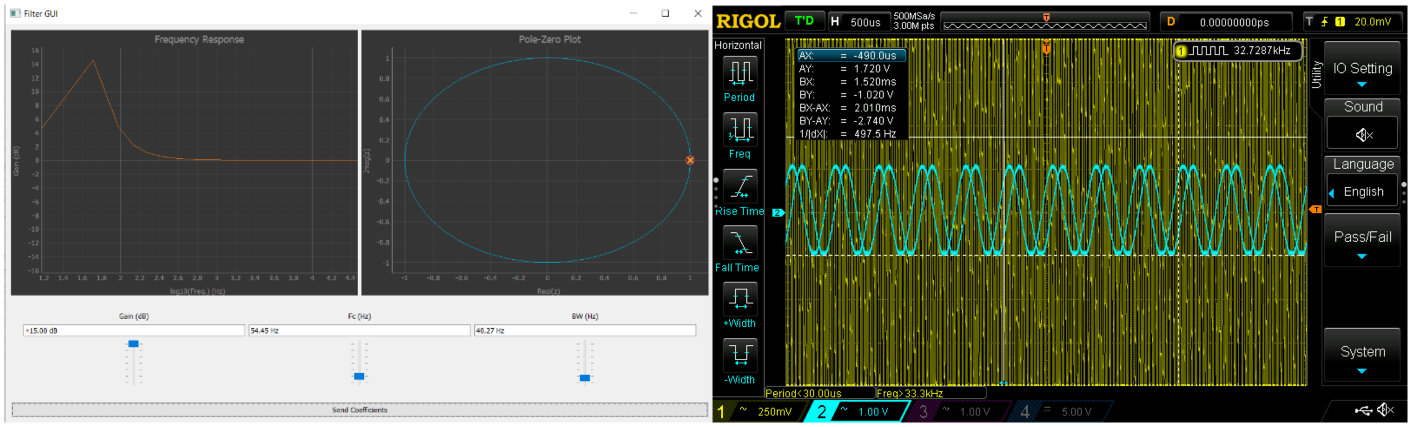
Figure 30: *The parametric EQ parameters specified were G=+15.0dB, F_c = 9000Hz, BW=812 Hz. The input signal (blue) is a 1000 Hz square wave. Notice, the output signal (yellow) has the 9th harmonic amplified (9000Hz) by about 15dB.*



```
Gain (dB)      Fc (Hz)      BW (Hz)
+15.00 dB      3.00 kHz     66.53 Hz
```

```
H 500us   500MSa/s 3.00M pts   D 0.00000000ps   T 20.0mV
AX:    = -490.0us
AY:    = 1.720 V
BX:    = 1.520ms
BY:    = -1.020 V
BX-AX: = 2.010ms
BY-AY: = -2.740 V
1/|dX|: = 497.5 Hz
2.00035kHz
Period=1.000ms   Freq=1.00kHz
1 250mV   2 1.00 V   3 1.00 V   4 5.00 V
```

Figure 31: *The parametric EQ parameters specified were G=+15.0dB, F_c = 3000Hz, BW=66 Hz. The input signal (blue) is a 1000 Hz square wave. Notice, the output signal (yellow) has the 3*rd *harmonic amplified (3000Hz) by about 15dB.*



```
Gain (dB)      Fc (Hz)      BW (Hz)
+15.00 dB      54.45 Hz     40.27 Hz
```

```
H 500us   500MSa/s 3.00M pts   D 0.00000000ps   T 20.0mV
AX:    = -490.0us
AY:    = 1.720 V
BX:    = 1.520ms
BY:    = -1.020 V
BX-AX: = 2.010ms
BY-AY: = -2.740 V
1/|dX|: = 497.5 Hz
32.7287kHz
Period<30.00us   Freq>33.3kHz
1 250mV   2 1.00 V   3 1.00 V   4 5.00 V
```

Figure 32: *The parametric EQ parameters specified were G=+15.0dB, F_c = 55Hz, BW=40 Hz. The input signal (blue) is a 1000 Hz sine wave. Notice, the output signal (yellow) is extremely noisy and corrupted. This is due to the fixed-point poles being outside the unit circle. This is an example of the difficulties of converting an infinite precision filter design to a fixed-point design.*

References

[1] A. V. Oppenheim and R. W. Schafer, *Discrete-time signal processing*. Harlow: Pearson, 2014.

[2] R. Bristow-Johnson, “RBJ Audio-EQ-Cookbook¶,” *RBJ Audio-EQ-Cookbook - Musicdsp.org documentation*, 04-May-2005. [Online]. Available: https://www.musicdsp.org/en/latest/Filters/197-rbj-audio-eq-cookbook.html. [Accessed: Feb 2020].

[3] D. G. Manolakis and V. K. Ingle, *Applied digital signal processing: theory and practice.* Cambridge, U.K.: Cambridge University Press, 2011. ch. 15, pp. 902-967.

[4] SparkFun, *SparkFun USB UART Serial Breakout - CY7C65*, Accessed on: May 12, 2020. [Online]. Available: *https://www.sparkfun.com/products/13830*

[5] “Pmod DA2: Two 12-bit D/A Outputs,” *Digilent*. [Online]. Available: https://store.digilentinc.com/pmod-da2-two-12-bit-d-a-outputs/. [Accessed: Apr-2020].

[6] “Pmod AD1: Two 12-bit A/D Inputs,” *Digilent*. [Online]. Available: https://store.digilentinc.com/pmod-ad1-two-12-bit-a-d-inputs/. [Accessed: Apr-2020].

[7] Xilinx, *ML505/ML506/ML507 Evaluation Platform User Guide*, UG347 datasheet. May. 2011.