GENERAL-PURPOSE DIGITAL FILTER PLATFORM

by

Michael Cheng

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

June 2017

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

ABSTRACT

This senior project provides a platform for high-speed, general-purpose digital filter implementation. EE 459 currently implements digital filters using reprogrammable digital signal processor boards. These aging digital signal processors serially calculate each difference equation term. Operating at 1 Mega-sample per second, the new general-purpose platform simultaneously processes at least ten digital filtering difference equation coefficients. The platform also features an audio jack input and BNC connectors for viewing input and output signals. The filter digitizes single channel audio signals at 44.1 kHz sampling rate with 16-bit precision or 1 MHz sampling at 8-bit precision. The new reprogrammable platform includes a data acquisition computer interface. This general-purpose digital filter supports a more modern approach to DSP development than the current platform.

I. INTRODUCTION

Cal Poly's Digital Signal Processing laboratory, EE 459, currently uses a platform that leverages the slower serial digital signal processor. This general-purpose, reprogrammable processor supports the implementation of customized filters, but the platform's computation speed severely limits filter performance. Additionally, the platform uses an obsolete DSP device that is not used in new designs. Therefore, students are not exposed to devices they might encounter in their future careers. This translates to inoperability as time progresses and generates the need for a modern solution, leading me to begin a project developing a new digital filter platform.

The General-Purpose Digital Filter Platform modernizes the EE 459 laboratory experience by providing an industry-oriented filtering solution [1, 2]. This project provides a new platform that exploits FPGA execution parallelism to provide high speed digital filtering. It exposes students to the latest industry design practices, increasing effectiveness of Cal Poly students in signal processing positions [1 - 4]. It is high speed, reprogrammable, compact, fully enclosed, and energy efficient for use in laboratory environments. Additionally, it has standard input and output ports, which improves on the previous platform that had standard ports hanging by wires due to post-design additions. While FPGA based signal processing solutions significantly increase cost and development time, the higher performance derived from inherent parallelism drastically outweighs the disadvantages. This platform offers another learning tool in addition to the older platform to teach newer signal processing design methodologies.

II. PRODUCT DESIGN ENGINEERING REQUIREMENTS

Table 1 details the requirements and specifications for this project. Choosing an appropriate FPGA for this project proves difficult. Some FPGAs come integrated onto a platform, whereas others only come as single chips. Integrating a preexisting product limits the overall platform design, where the single chip method allows for greater control over platform design. However, designing a board with only the FPGA increases board layout difficulty through the numerous pins requiring testing. Parts selection also proved difficult through balancing the need for high performance parts with low costs. Additionally, the added Vivado System Generator requirement limits FPGA selection. Determining customer needs required consulting various people impacted by the project. Since this project primarily benefits EE 459, the project consulted Dr. Wayne Pilkington, who dictates the platform's marketing requirements and engineering specifications. Additionally, multiple EE 460 students provided additional input to refine Table 1's contents. At its core, the platform supports the simultaneous processing of multiple difference equation coefficients through an FPGA's parallel architecture [1, 2, 4, 5]. Platform requirements also stemmed from consulting multiple design resources in the forms of patents and IEEE database articles [6 – 9]. The system must function in a standard Cal Poly electrical engineering laboratory environment and provide superior performance when compared to the previous platform.

TABLE I
GENERAL-PURPOSE DIGITAL FILTER PLATFORM REQUIREMENTS AND SPECIFICATIONS

Marketing Requirements	**Engineering Specifications**	**Justification**
3	The platform runs on 120 V, 60 Hz AC power supplied by an IEC type A outlet.	The United States supports IEC type A and B as the standard for electrical outlets [10]. American utility companies supply electricity for general usage at 120 V, 60 Hz [11].
2, 8	The platform interfaces with a computer through at least USB 2.0 for reprogramming.	Existing commercial FPGA platforms integrate USB 2.0 functionality for computer reprogramming [12].
1, 4	The FPGA simultaneously processes at least ten difference equation terms.	Customer required this specification to improve upon the previous platform.
5	The platform has at least four onboard user-configurable debugging SPST toggle switches and green LEDs for diagnostic purposes.	Customer required this specification to incorporate features found in the previous platform.
6	The system exists on at least a 2-layer printed circuit board occupying no more than 5.5”x8.5”x3.5”.	Customer required the platform exist on a final printed circuit board. The sizing requirements satisfy the enclosure specifications.
6, 7, 10	The final product fits in a space measuring no more than 6”x9”x4”.	The Cal Poly electrical engineering standard lab bench accommodates a device measuring 2’x2’x2’, but the form factor should not exceed the current solution’s footprint. An electrically insulated enclosure encapsulates the entire platform to prevent unwanted shocking.
7	The enclosure must protect against electricity measuring up to 120 V.	The power delivered by the electrical output serves as the highest voltage the platform sees.
6	The platform’s maximum power draw consumes no more than 85 W.	High performance FPGA platforms recommend 60 W power supplies [13].
7	The product operates normally between 40 and 100 degrees Fahrenheit.	The Cal Poly electrical engineering standard lab environment ranges anywhere from 40 to 100 degrees Fahrenheit.
8	The system uses at least two 3.5mm audio jacks and at least two BNC connectors for connecting input and output signals, where it samples from either input and produces the output at either connector.	Customer required this specification for standard input and output signal ports. Popular electronic devices support transmission and reception of audio signals through 3.5mm audio jacks [14]. Most electrical engineering lab bench instruments support BNC connections.
9	The platform samples any single channel audio signal at 44.1 kHz with at least 16-bit precision or 1 MHz with at least 8-bit precision.	Customer required this specification to support audio signal processing and high speed signal processing.
2	The FPGA and computer interface supports the Vivado System Generator tool.	Customer required this specification to streamline filter design. Converting designs from Simulink to FPGA implementations finds wide use in research [15 – 17].

Marketing Requirements
1. High-speed Processing
2. Reprogrammable
3. Wall Powered
4. FPGA-based
5. Onboard Diagnostic Tools
6. Compact and Energy Efficient
7. Suitable for Lab Environments
8. Standard Input and Output Signal Interfaces
9. High-Quality Digitization
10. Dedicated Enclosure

The requirements and specifications table format derives from [18], Chapter 3.

Table 2 describes input and output functionality of the General-Purpose Digital Filter Platform block diagram. It provides electrical specifications for every input and output, while providing how signals propagate through the system.

TABLE II
GENERAL-PURPOSE DIGITAL FILTER PLATFORM LEVEL 0 SIGNAL ANALYSIS

Module	**General-Purpose Digital Filter Platform**
Inputs	3.5 mm Audio Jack Input Signal: 2 V peak, 20 Hz - 20 kHz BNC Input Signal: 2 V peak, 20 Hz - 20 kHz Power: 5.9 V DC, 23.6 W User-Programmable Filter Design: HDL filter profile
Outputs	3.5 mm Audio Jack Output Signal: 2 V peak, 20 Hz - 20 kHz BNC Output Signal: 2 V peak, 20 Hz - 20 kHz
Functionality	The system filters the input signal from either the 3.5mm or BNC input. The user determines the filter design through programming the platform with customized hardware description language code. The output signal appears on either the 3.5mm or BNC port.

Figure 1 describes the completed system from a level 0 or high-level view. It shows that the platform takes inputs from either the 3.5 mm audio jack or BNC connector. The filter processes the digitized inputs and sends the result to either the 3.5 mm audio jack or BNC connector output.

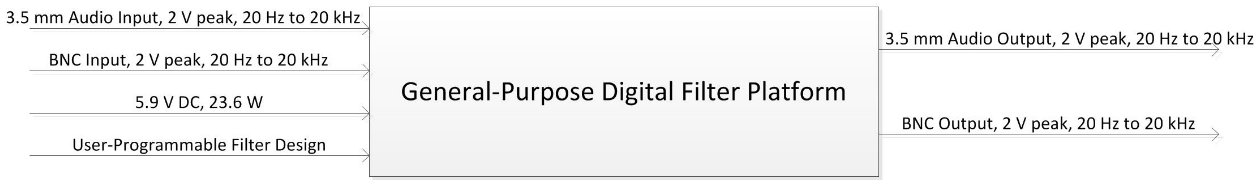

3.5 mm Audio Input, 2 V peak, 20 Hz to 20 kHz
BNC Input, 2 V peak, 20 Hz to 20 kHz
5.9 V DC, 23.6 W
User-Programmable Filter Design
General-Purpose Digital Filter Platform
3.5 mm Audio Output, 2 V peak, 20 Hz to 20 kHz
BNC Output, 2 V peak, 20 Hz to 20 kHz

FIGURE I
GENERAL-PURPOSE DIGITAL FILTER PLATFORM LEVEL 0 BLOCK DIAGRAM

III. BACKGROUND

The advent of the digital age sparked the need to digitize analog signals for processing on computers. The rise of faster digital computing creates a desirable option to shift certain aspects of analog design into a digital form. Analog filtering often requires many design formulas and significant time to develop a desired product, inhibiting future expandability and customization [19]. It also produces results that vary with component tolerances and drift [20].

Digital filtering provides a solution that features expansion and customization capabilities. Through reprogramming, filter functions can change without necessitating additional hardware development costs [20]. Traditional digital filters employ discrete digital signal processors to quickly process difference equation terms. These processors sequentially execute instructions. Serial instruction execution establishes an upper data throughput limit, which dampens future efforts to create faster and more complex filters [1,2].

To combat digital signal processor limitations, field programmable gate arrays allow for parallel processing through multiple digital signal processing slices. Modern FPGAs integrate DSP slices into the final product, reducing the need for traditional interfacing between microcontrollers and digital signal processors [3]. This significant benefit causes industry applications to shift from discrete digital signal processors to FPGAs [1, 4].

IV. SYSTEM DESIGN – FUNCTIONAL DECOMPOSITION

Table 3 shows the ADC signal specifications. It also describes the ADC's role in the platform. The ADC samples an analog waveform and converts it into digital data for processing on the FPGA.

TABLE III
GENERAL-PURPOSE DIGITAL FILTER PLATFORM LEVEL 1 ADC ANALYSIS

Module	**ADC**
Inputs	3.5 mm Audio Jack Input Signal: 2 V peak, 20 Hz - 20 kHz BNC Input Signal: 2 V peak, 20 Hz - 20 kHz DC Voltage: 5 V, DC
Outputs	Digital Input: 8-bit or 16-bit digital signal, 0 Hz – 1 MHz
Functionality	The ADC samples the analog input signal from either the 3.5 mm audio input or the BNC input. It digitizes the input signal with up to 16-bit precision for processing on the FPGA. DC voltage powers the ADC.

Table 4 shows the FPGA signal specifications. It describes the FPGA's role in the platform. The FPGA applies a filtering profile specified by the user. It outputs this filtered digital signal to a DAC for conversion into an analog waveform.

TABLE IV
GENERAL-PURPOSE DIGITAL FILTER PLATFORM LEVEL 1 FPGA ANALYSIS

Module	**FPGA**
Inputs	Digital Input: 8-bit or 16-bit digital signal, 0 Hz – 1 MHz DC Voltage: 5 V, DC User-Programmable Filter Design: Filter design written in HDL code
Outputs	Digital Output: 8-bit or 16-bit digital signal, 0 Hz – 1 MHz
Functionality	The FPGA processes the digital input signal and applies the user-provided filter design. It outputs the resulting data in a digital output signal. DC voltage powers the FPGA.

Table 5 shows the DAC signal specifications. It describes the DAC's role in the platform. The DAC converts the FPGA's digital output into an analog output. This output signal appears at one of the output ports.

TABLE V
GENERAL-PURPOSE DIGITAL FILTER PLATFORM LEVEL 1 DAC ANALYSIS

Module	**DAC**
Inputs	Digital Output: 8-bit or 16-bit digital signal, 0 Hz – 1 MHz DC Voltage: 5 V, DC
Outputs	3.5 mm Audio Jack Output Signal: 2 V peak, 20 Hz - 20 kHz BNC Output Signal: 2 V peak, 20 Hz - 20 kHz
Functionality	The DAC converts the digital output from the FPGA into an analog output that appears on both the 3.5 mm audio jack and BNC connector. DC voltage powers the DAC.

Table 6 shows the voltage regulator signal specifications. It describes the voltage regulator's role of converting higher voltage into a lower one. This regulated voltage is used by all devices inside the platform.

TABLE VI

GENERAL-PURPOSE DIGITAL FILTER PLATFORM LEVEL 1 VOLTAGE REGULATOR ANALYSIS

Module	**Voltage Regulator**
Inputs	DC Voltage: 5.9 V, DC, 23.6 W
Outputs	DC Voltage: 5 V, DC
Functionality	The voltage regulator converts the 5.9 V input DC voltage into a 5 V DC voltage.

Figure 2 provides further detail when compared to Figure 1. It explores the basic structure of the entire platform. The analog-to-digital converter samples the input signal. The FPGA filters this digital signal using user-defined filter settings, where the digital-to-analog converter produces the resulting waveform at the output connectors. Electricity provided from the utility company powers the platform.

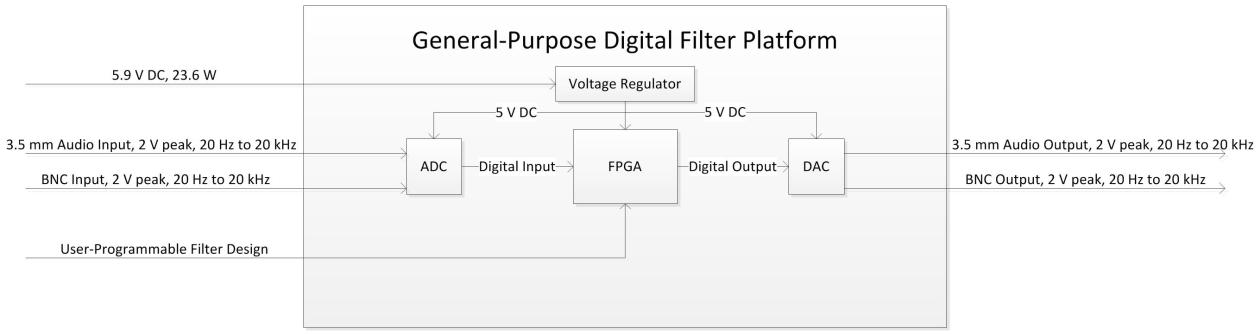

General-Purpose Digital Filter Platform
5.9 V DC, 23.6 W
Voltage Regulator
5 V DC
5 V DC
3.5 mm Audio Input, 2 V peak, 20 Hz to 20 kHz
BNC Input, 2 V peak, 20 Hz to 20 kHz
ADC
Digital Input
FPGA
Digital Output
DAC
3.5 mm Audio Output, 2 V peak, 20 Hz to 20 kHz
BNC Output, 2 V peak, 20 Hz to 20 kHz
User-Programmable Filter Design

FIGURE II

GENERAL-PURPOSE DIGITAL FILTER PLATFORM LEVEL 1 BLOCK DIAGRAM

V. TECHNOLOGY CHOICES AND DESIGN APPROACH ALTERNATIVES CONSIDERED

To meet the sampling specifications listed in table #, there was possibility of needing two ADCs and two DACs to satisfy both sampling rate requirements. Delta-sigma ADC architectures are prevalent in the high precision data conversion area, but the drawback is that the maximum sampling rate typically falls under 200 kHz [21]. Initially, this issue was addressed by finding two ADCs: one that gave high precision and another that gave high sampling rate. Attempting to integrate four mixed-signal IC's proved challenging through the significant additional cost. Eventually, one ADC and DAC combination was found that satisfied all sampling and precision requirements. The ADS8681 ADC had a successive-approximation architecture supporting both sampling rates up to 16-bit precision [22]. The DAC8811 had a 0.5 µs settling time with 16-bit precision but was only available in current output form [23]. While a trans-impedance amplifier would have converted the current output into voltage, the DAC8830 provided a voltage output 16-bit precision DAC with 1 µs settling time, providing an easier implementation [24].

However, the voltage output from the DAC was unbuffered, which necessitated an op amp buffer to perform impedance transformations. The ADS8681 also provides an onboard 4.096 V bandgap reference that could be used for the DAC reference [22]. This reference is unbuffered as well. Finally, an input buffer was required to add a 2.5 V DC offset to the AC input signal for single supply operation. The OPA4354 provided four op amps for general use with a high slew rate of 150 V/µs [25]. The required slew rate for a 5 V peak input signal at 500 kHz is approximately 16 V/µs [26]. Since three op amps are required for buffering in the platform, the quad array provides a single chip solution, which reduces costs. Furthermore, the op amps are CMOS, which eliminates the possibility of loading the unbuffered sources. The use of op amp buffers also allowed for easy inclusion of low-order active filters.

For the FPGA selection, the original thought was to create a single board platform. This proved challenging for development, as FPGA chips are typically packaged as ball grid arrays. The single board solution was abandoned due to development time constraints. A Xilinx FPGA was also required to use Vivado System Generator, which translated Simulink code to VHDL. Since the platform directly benefits EE 459, the Basys 3 FPGA development board was chosen to reduce complexity and development time associated with the entire platform. The Basys 3 is used in other Cal Poly EE department courses, mainly in CPE 133 and 233. The development board is packaged with an FPGA with 90 DSP slices [12]. This satisfies the requirement that the platform must process at least ten coefficients, since the worst case is using one DSP slice per coefficient term. Additionally, the platform comes with diagnostic oriented peripherals like LEDs, switches, and buttons integrated into the development board. Instead of designing the filtering platform as a single board, the new concept is to design a peripheral board around a preexisting FPGA platform used in other classes. This significantly enhances cost efficiency of the product, as well as the student learning experience.

To see if the platform could work theoretically, a prototype platform system implemented using the Basys 3, MCP3208, and MCP4921. This system sampled an input waveform with a 12-bit SAR ADC, filtered the data through the Basys 3 DSP slices, and output a waveform using a 12-bit DAC [12, 27, 28]. Figure 3 below shows the complete schematic of the prototype test circuit.

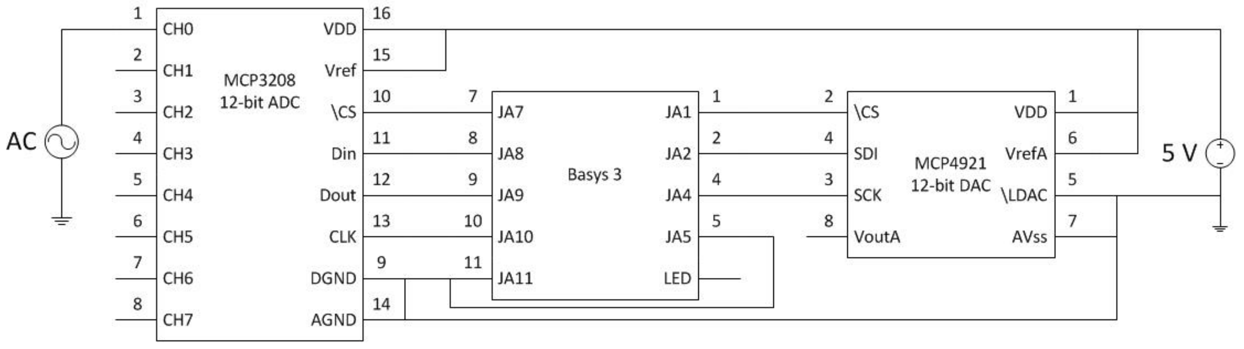


FIGURE III
PROTOTYPE FILTER PERIPHERAL BOARD SCHEMATIC

Xilinx produces an intellectual property core that allows easy production of FIR filters. This core was customized with the frequency response profile shown below in figure 4. The sampling rate of the entire system is 50 kHz with the master and serial control clocks at 100 MHz. The Basys 3 controls the ADC and DAC through SPI protocols. Figure 5 shows that the filtering operation with the Xilinx intellectual property core successfully occurs. The 1 kHz input sinusoid was 2 V peak to peak, and the output waveform is 599 mV peak to peak. This attenuation factor is approximately 0.3, and the predicted attenuation was 0.3466. This verifies that the Basys 3 can use DSP slices to perform digital filtering.

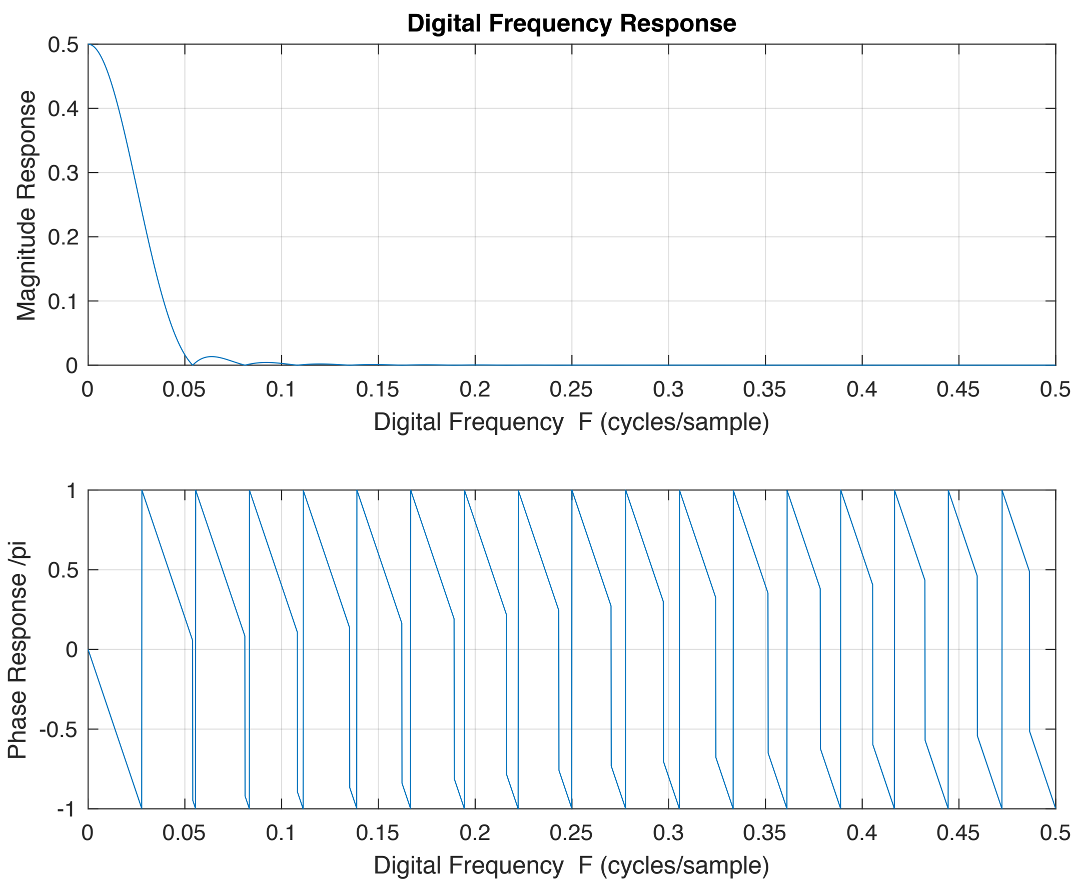
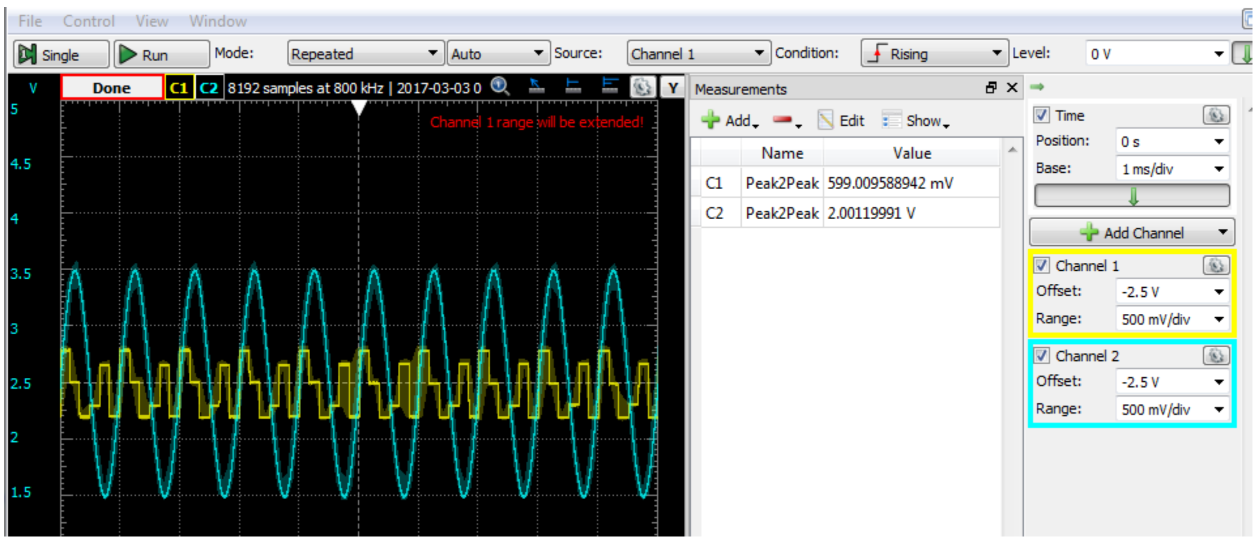


FIGURE IV
PROTOTYPE FILTER PERIPHERAL BOARD DIGITAL FREQUENCY RESPONSE



Mode: Repeated Auto Source: Channel 1 Condition: Rising Level: 0 V

Done 8192 samples at 800 kHz | 2017-03-03 0

Channel 1 range will be extended!

Measurements

	Name	Value
C1	Peak2Peak	599.009588942 mV
C2	Peak2Peak	2.00119991 V

Time
Position: 0 s
Base: 1 ms/div

Channel 1
Offset: -2.5 V
Range: 500 mV/div

Channel 2
Offset: -2.5 V
Range: 500 mV/div

FIGURE V
PROTOTYPE FILTER 1 KHZ SINUSOID TEST

VI. PROJECT DESIGN DESCRIPTION

Figure 6 details the level 2 block diagram for the entire platform. It includes all major stages of the filtering process. The inputs from either the 3.5mm or BNC jack are passed to a physical switch, where the user decides which signal to filter. The input signal is buffered and level shifter to a single supply range, since the input signal is AC coupled. The ADC samples the shifted input signal, where the FPGA performs the FIR filtering operation. The ADC comes with an internal 4.096 V bandgap reference that is fed to the DAC [22]. The DAC reproduces the signal from the data clocked in from the FPGA over SPI. The new analog signal is passed through an active low pass filter with unity gain to remove any digital distortion introduced by the DAC. The user then determines where to send the output using the second switch. The system takes in 5.9 V DC provided from an AC adapter, where voltage regulators step down the voltage to 5 V for the system's ICs [29].

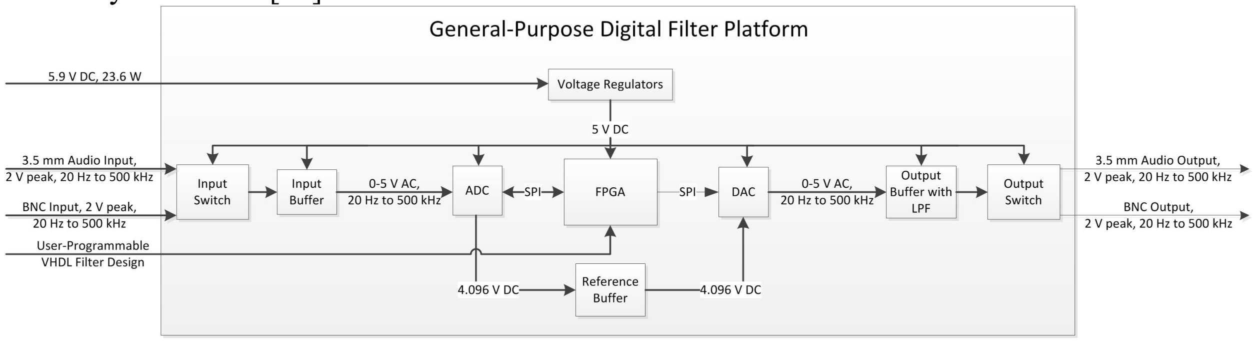

General-Purpose Digital Filter Platform
5.9 V DC, 23.6 W
Voltage Regulators
5 V DC
3.5 mm Audio Input, 2 V peak, 20 Hz to 500 kHz
BNC Input, 2 V peak, 20 Hz to 500 kHz
User-Programmable VHDL Filter Design
Input Switch
Input Buffer
0-5 V AC, 20 Hz to 500 kHz
ADC
SPI
FPGA
SPI
DAC
0-5 V AC, 20 Hz to 500 kHz
Output Buffer with LPF
Output Switch
3.5 mm Audio Output, 2 V peak, 20 Hz to 500 kHz
BNC Output, 2 V peak, 20 Hz to 500 kHz
4.096 V DC
Reference Buffer
4.096 V DC

FIGURE VI
GENERAL-PURPOSE DIGITAL FILTER PLATFORM LEVEL 2 BLOCK DIAGRAM

The ADS8681 ADC has 16-bit resolution at up to 1 MSPS [22]. This satisfies the sampling rate and precision specifications listed in the table above. The DAC8830 has a 1 μs settling time, which complements the ADC's 1 MSPS sampling rate [22, 24]. All buffers have unity gain to prevent clipping at the supply rails, where the input buffer introduces a 2.5 V DC offset to allow for single supply operation. The output buffer has a small feedback capacitor to implement a low-order low pass filter. This smooths the digital step discontinuities introduced by the DAC. The sampling rate associated with the system is determined by the FPGA, but the maximum FPGA system clock is 100 MHz. There are two LT1529 linear voltage regulators for the entire platform. The first is dedicated to running the digital hardware present in the platform. This digital regulator is responsible for providing 5 V to the Basys 3, as well as the digital hardware present in the ADC [29]. The second regulator is to provide a clean analog voltage source and ground for the analog hardware in the platform.

The FPGA block in Figure 6 contains the reconfigurable digital filter module. This module contains all necessary support logic to take the ADC data received over SPI and filter the data for output on the DAC. Figure 7 shows the logic contained within the FPGA block of Figure 6. The Gateway In block is where the ADC data is received. This 16-bit value is scaled by 1.25 due to the analog front end built into the ADC. The ADS8681's input range for this configuration is from 0 to 5.12 V, which differs from a conventional ADC [22]. This means that even though the internal ADC reference is 4.096 V, an input signal of 4.096 V will be quantized as 4.096 V/ 5.12 V * 65536. Since this ADC does not quantize over the reference full scale range, the CMult block restores the full 16-bit range to represent 0 to 4.096 V. The peripheral board's single supply limitation forced a DC offset to be added to the input signal at the input buffer of Figure 6. A 2.048 V DC offset was chosen to maximize the swing of an AC signal. However, the filtering step needs to subtract this DC offset, which is why the AddSub block subtracts 32768 from the input value. Half of the reference, which is 2.048 V, corresponds to an ADC value of 32768. The input buffer stage referenced in Figure 6 uses a unity gain inverting amplifier to buffer the input AC signal. The Negate block eliminates the polarity inversion. The conditioned value is then sent to the Digital FIR Filter block, which uses coefficients generated by the FDATool. The FDATool opens MATLAB's Filter Designer application, so a user can specify filter characteristics. This block can also take a row coefficient vector. The FPGA in the Basys 3 only has 90 DSP slices, which corresponds to 90 multipliers [12]. The whole system requires a minimum of 4 DSP slices for signal conditioning, leaving 86 slices for filtering. These 86 slices allow for a 171 tap symmetric filter, since symmetric coefficients only require one multiplier for two taps. After filtering, the DC offset is added back for output to the DAC. The BitBasher block only takes the integer bits from the conditioned value for output. All the intermediate blocks prior to the BitBasher utilize fixed point math to maintain calculation accuracy. The sampling rate of the entire system is controlled through the System Generator block and Gateway In block UI control windows. Additionally, the System Generator block contains the FPGA target information and means to generate necessary VHDL code to translate the blocks between Gateway In and Gateway Out to something Vivado can use. Figure 7's sampling rate is 1 MSPS. In Figure 8, Down Sample and Up Sample blocks were introduced to satisfy the 44.1 kHz requirement. This effectively creates a system sampling rate of 44.1 kHz without changing the surrounding VHDL system support modules.

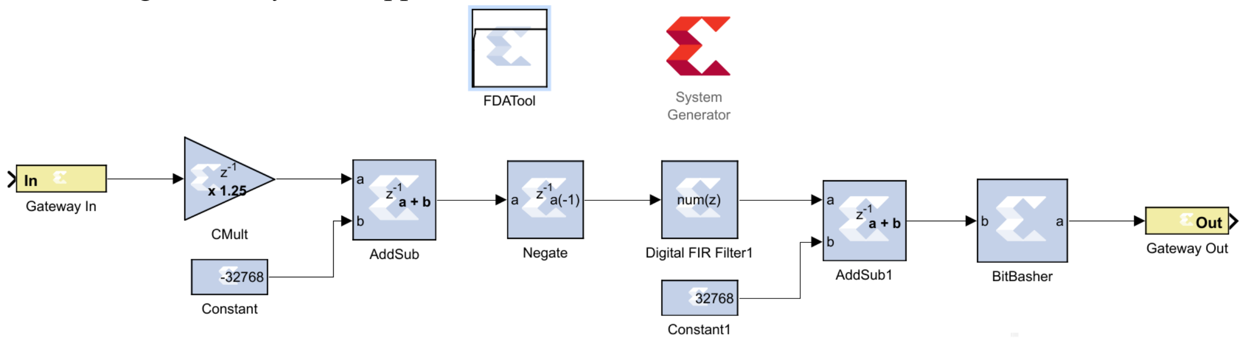


FIGURE VII
FPGA BLOCK'S 1MSPS INTERNAL LOGIC

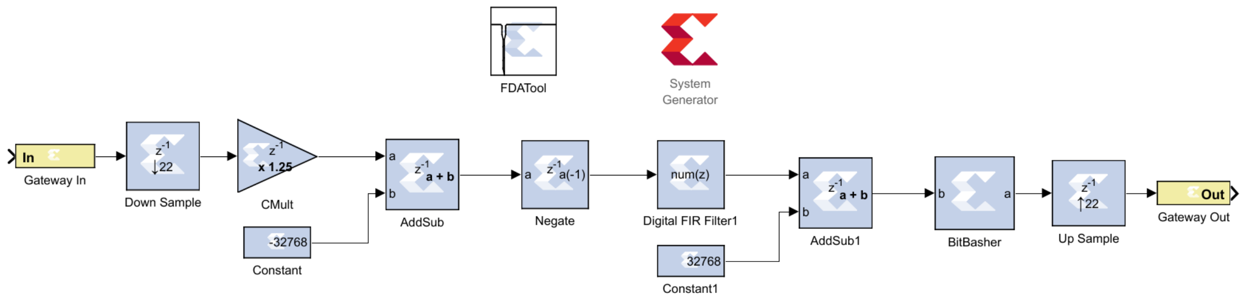


FIGURE VIII
FPGA BLOCK'S 44.1KSPS INTERNAL LOGIC

VII. PHYSICAL CONSTRUCTION AND INTEGRATION

Figure 9 details the complete circuit schematic for the platform's peripheral board. It integrates the ADC, DAC, op-amp array, various passive components, connectors, and jacks. The schematic shows the entire circuit for the platform described in figure 6. The schematic separates the analog and digital ground planes to prevent excessive digital noise contaminating the analog ground plane. This is particularly important for the ADC and DAC, where conversion and reproductions take place using references. If the analog ground plane is moving too much, data conversion errors occur. The analog and digital ground planes are joined through a small wire near the ADC for the connection to act like an inductor, which is inherently low-pass in nature. Analog and digital circuitry was isolated from each other to further prevent noise contamination. The linear voltage regulators use the PCB as a heatsink. Power to the regulators is supplied through a 5.9 V wall adapter [30]. All passive components were limited to 0805 and smaller for layout space constraints, and all ICs were bypassed with both ceramic and tantalum decoupling capacitors to also prevent noise contamination.

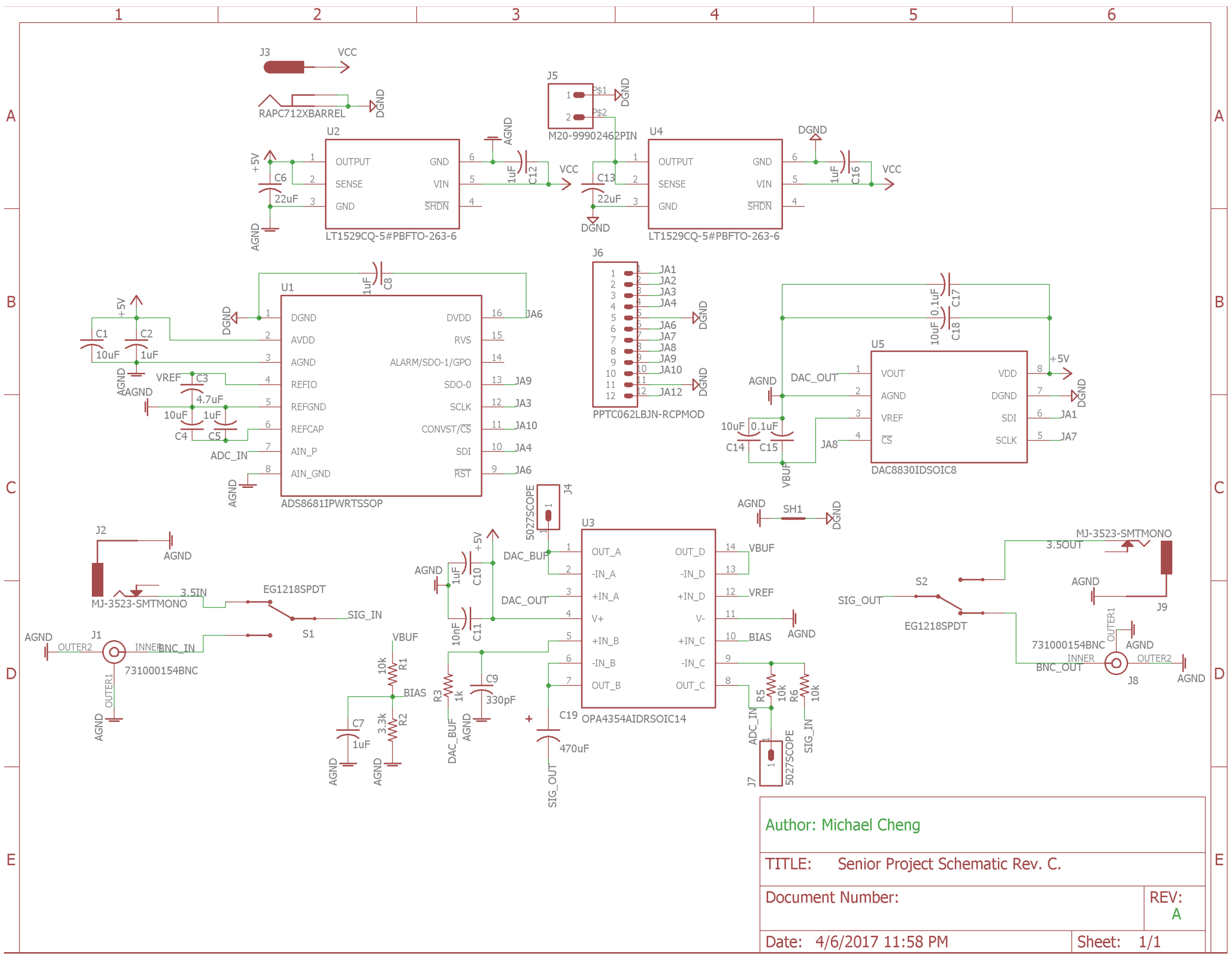


FIGURE IX

GENERAL-PURPOSE DIGITAL FILTER PLATFORM CIRCUIT SCHEMATIC

VIII. INTEGRATED SYSTEM TESTS AND RESULTS

For the following three filter design verification tests, the platform's BNC input was connected to an Agilent 33120A arbitrary waveform function generator, and the BNC output was connected to a Keysight MSOX2022A oscilloscope. Both channels of the oscilloscope were AC coupled, as all signals seen by the platform were AC waveforms. The input and output SPDT switches were both set to BNC. Since the requirements stated the platform needed to filter any frequencies between 20 Hz and 20 kHz, the function generator was set to output at 3.8 Vpp tone sweep from 1 Hz to 20 kHz with a sweep period of 100 ms. The Keysight MSOX2022A provided the time-domain data points for the input and output of the platform. The oscilloscope's trigger was connected to the synchronization output on the function generator to ensure proper oscilloscope waveform capture. Using MATLAB, these two waveforms were post processed to produce the frequency response shown in figures 10 through 12.

Using the FIR filter support logic from Figure 8, the FDATool was configured with a band pass filter. The filter assumed a sampling rate of 44.1 kHz, corresponding to the down sampled input data. The pass band allowed for 1 dB of ripple, and the stop band attenuation was 80 dB. Additionally, the first transition band was from 1 kHz to 3 kHz, and the second transition band was from 9 kHz to 11 kHz. The final filter order was 56, but the filter coefficients were symmetric. Therefore, only 29 multipliers were required for the filter block. In Figure 10, the band pass filter experimental and theoretical frequency responses are compared. The figure's x axis has been limited to 20 Hz to 20 kHz per requirements. The magnitude response shows that the pass band closely resembles the expected filter response. The response reveals that the platform's noise floor is -40 dB, which is lower than the desired 80 dB attenuation. The experimental phase response shows linear phase in the pass band, which verifies the FIR design. The phase response's y axis is scaled by π to clearly show points where the phase wraps. Figure 10's phase response differs through having a steeper experimental slope. The group delay directly translates into a sample delay for FIR filters, and the theoretical group delay is 28 samples. At a surface level, this might conflict with the experimental group delay of 38 samples, which was calculated as the change of phase with the change of radial frequency. The group delay formula used was $g = |\theta_2 - \theta_1| / ((f_2 - f_1) * \frac{2}{F_s})$. These extra 10 sample delays can be accounted for in the six internal filter support logic blocks like adding, constant multiplication, and inverting. The other four delays are found in the SPI modules, where there are two instances that communicate with the ADC and DAC. Each of these modules has two registers for buffering inputs and outputs. Altogether, this adds ten sample delays to the entire filtering process. Since the oscilloscope can only see the inputs and outputs of the platform and not only the filter, the sample delays include register-induced sample delays.

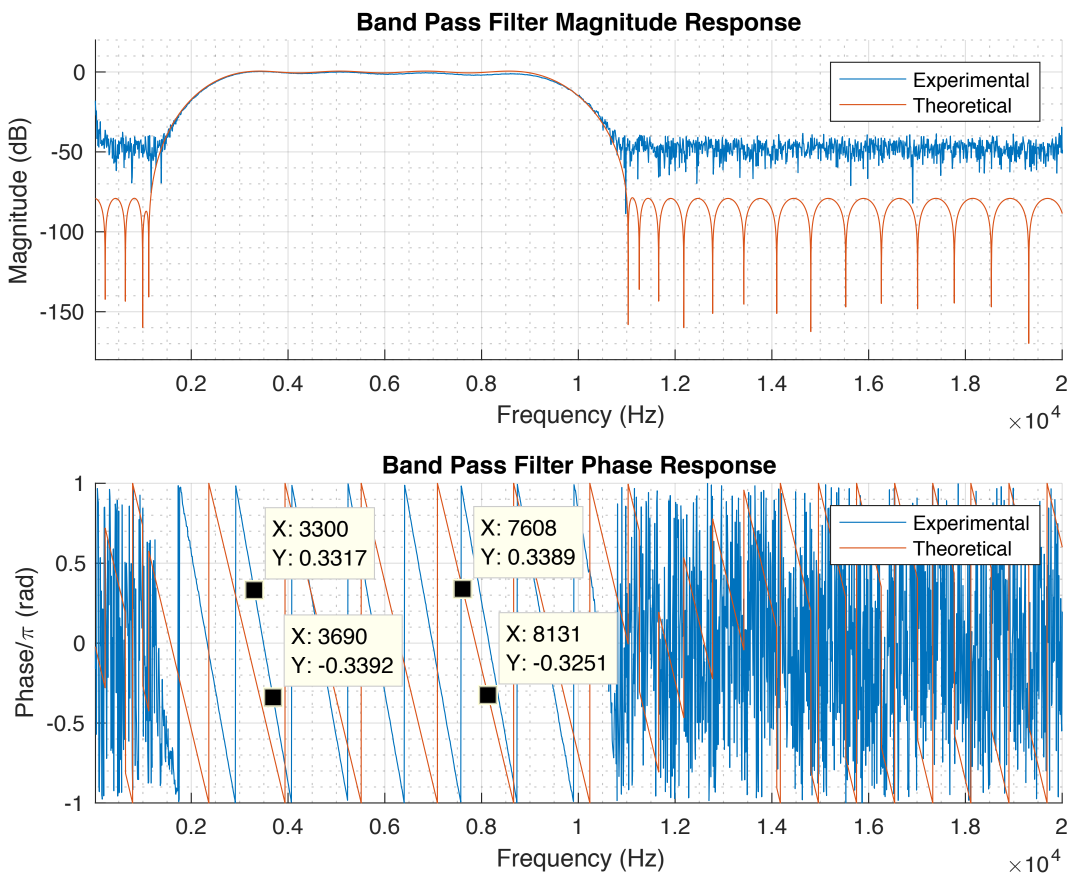


FIGURE X
BAND PASS FILTER FREQUENCY RESPONSE COMPARISON

Using the FIR filter support logic from Figure 8, the FDATool was configured with a band stop filter. The filter assumed a sampling rate of 44.1 kHz, corresponding to the down sampled input data. The pass band allowed for 1 dB of ripple, and the stop band attenuation was 80 dB. Additionally, the first transition band was from 3 kHz to 3.8 kHz, and the second transition band was from 4.2 kHz to 5 kHz. The final filter order was 152, but the filter coefficients were symmetric. Therefore, only 77 multipliers were required for the filter block. In Figure 11, the band stop filter experimental and theoretical frequency responses are compared. The figure's x axis has been limited to 20 Hz to 20 kHz per requirements. The magnitude response shows that the pass band closely resembles the expected filter response. The response reveals that the platform's noise floor is -40 dB, which is lower than the desired 80 dB attenuation. The experimental phase response shows linear phase in the pass band, which verifies the FIR design. The phase response's y axis is scaled by π to clearly show points where the phase wraps. Figure 11's phase response differs through having a steeper experimental slope. The group delay directly translates into a sample delay for FIR filters, and the theoretical group delay is 76 samples. At a surface level, this might conflict with the experimental group delay of 86 samples, which was calculated as the change of phase with the change of radial frequency. This verifies that the platform in Figure 8's configuration adds a ten-sample delay to the filter's theoretical group delay. At higher frequencies, the experimental pass band magnitude response seems to roll off at a constant rate. This is due to the integrated anti-aliasing filter on the ADC. The ADS8681 integrates a second order low pass filter after the programmable gain amplifier. These both combine to form the ADS8681's special integrated analog front end [22]. This low pass filter has a nominal -3 dB frequency of 15 kHz, and the experimental curve shows that the -3 dB frequency is 15 kHz.

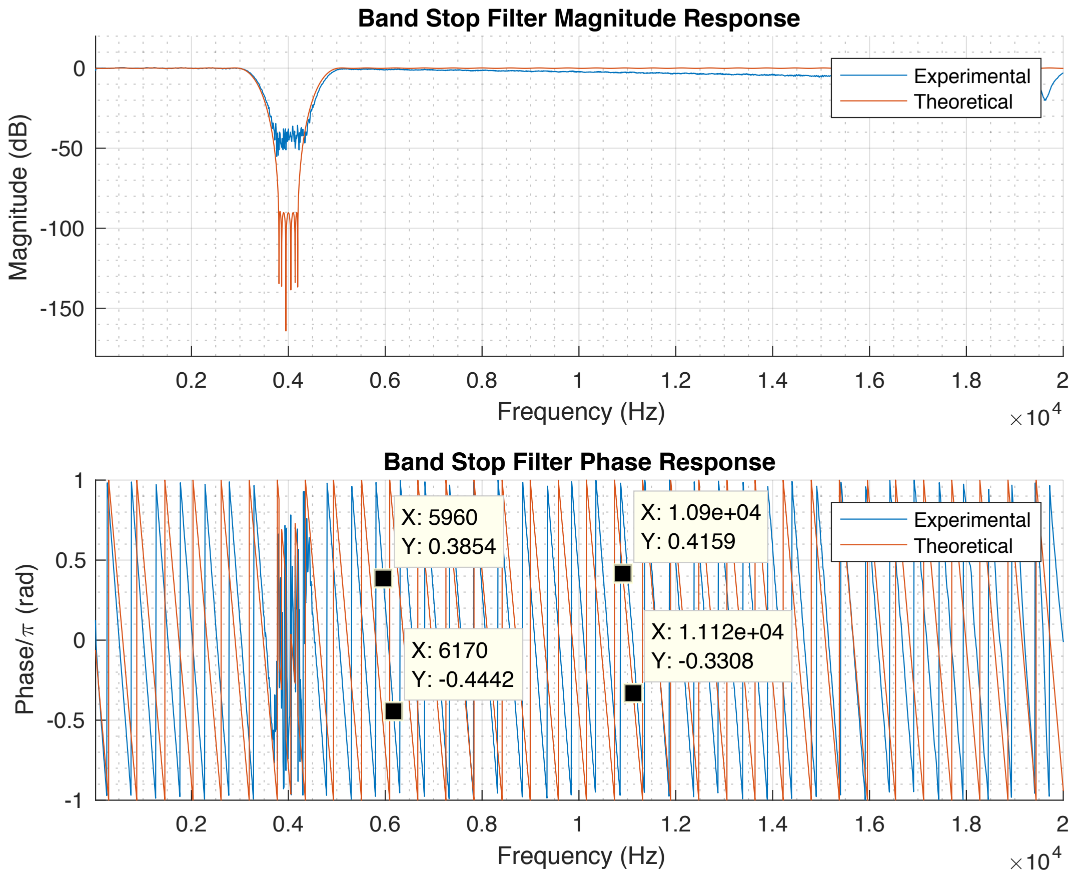


FIGURE XI

BAND STOP FILTER FREQUENCY RESPONSE COMPARISON

Using the FIR filter support logic from Figure 7, the FDATool was configured with a high pass filter. The filter assumed a sampling rate of 1 MHz, corresponding to the sampling rate of the ADC. This tested the full sampling rate capabilities of the system. The pass band allowed for 1 dB of ripple, and the stop band attenuation was 40 dB. Additionally, the transition band was from 400 Hz to 10 kHz. Due to the high sampling rate, the transition band had to be large to lower the overall filter order for the Basys board. The final filter order was 156, but the filter coefficients were symmetric. Therefore, only 79 multipliers were required for the filter block. The magnitude response shows that the pass band closely resembles the expected filter response. The response reveals that the platform's noise floor is -40 dB, which is lower than the desired 80 dB attenuation. The experimental phase response shows linear phase in the pass band, which verifies the FIR design. The phase response's y axis is scaled by π to clearly show points where the phase wraps. Figure 11's phase response differs through having a steeper experimental slope. The group delay directly translates into a sample delay for FIR filters, and the theoretical group delay is 78 samples. This conflicts with the experimental group delay of 190 samples, which was calculated as the change of phase with the change of radial frequency. This stark change stems from the VHDL synthesizer instantiating more support flip-flops, RAM, and look-up tables. This stems largely from the filter needing to operate at 1 MHz instead of 44.1 kHz.

Vivado provides a resource utilization report, detailing flip-flop, LUT, RAM, and DSP usage. While the DSP usage was as predicted, the flip-flop, LUT, and RAM usage significantly increased with a high order 1 MSPS filter. Where usage was previously below 2% for flip-flop, LUT, and RAM, the 1 MSPS filter usage sharply rose to as high as 35%. The additional memory introduced by the synthesizer adds the significant sample delay to the entire platform. However, the roll off still matches the characteristic seen in figure 11, verifying the datasheet's second order low pass filter response. This verifies that the filter magnitude response is unaffected, but the order of the filter directly impacts the phase response.

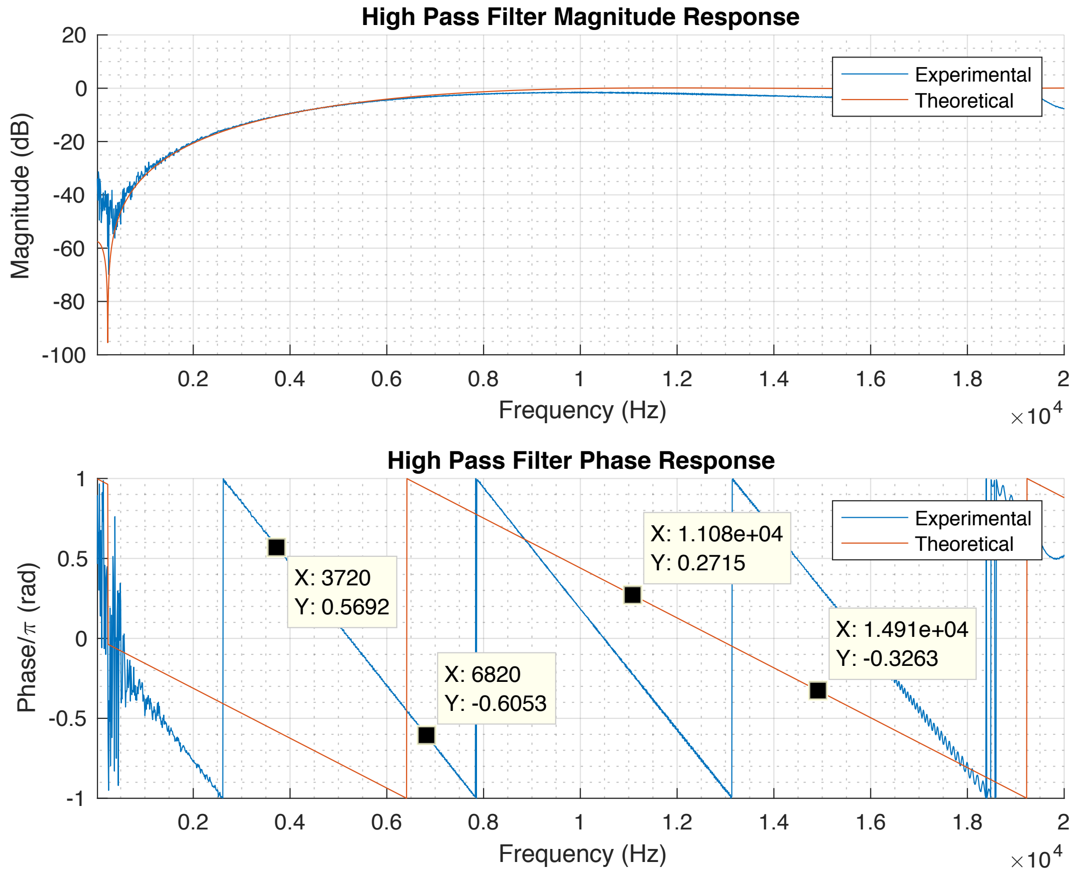


FIGURE XII

HIGH PASS FILTER FREQUENCY RESPONSE COMPARISON

To test the 3.5mm audio jacks on the platform, a computer was connected to the 3.5mm platform input, and speakers were connected to the 3.5mm audio output. The SPDT switches were then switched from BNC to 3.5mm for both input and output. To verify responses, the input and output waveforms were analyzed on the oscilloscope using the same configuration as the BNC tests. Oscilloscope probes were placed at the two probe points on the platform PCB. Using the same input frequency sweep, the observed responses for all three filters were identical to their BNC characterizations. These results were also verified audibly.

IX. CONCLUSIONS

The General-Purpose Digital Filter Platform delivers EE 459 an industry-oriented filtering solution that met all requirements and specifications [1, 2]. In many cases, the product exceeded original specifications. Specifically, the 90 DSP slices on the Basys 3 allowed for filters with 157 coefficients. The integrated features of the Basys 3 provided 16 LEDs and switches, which exceeded the original required four [12]. Vivado System Generator bypasses time-consuming HDL development and integration. Simulink's graphical user interface allows for easy specification of filter coefficients, and if desired, the FDATool allows for quick filter design. As seen with the three filter characterizations, multiple filter designs can be quickly implemented using one platform. This dynamic filter allows for hardware testing of EE 459 student designs.

However, there are numerous improvements that could be implemented in the future. Most of the improvement needs stem from insufficient time to fully develop the platform. The most significant design improvement is replacing the ADS8681 with another ADC without the 15 kHz antialiasing filter. Although the ADC samples at 1 MSPS, the filter has a -3 dB cutoff frequency well below the theoretical Nyquist limit, which significantly limits the input bandwidth of the platform. According to the ADS8681 datasheet, this antialiasing filter was instituted for limiting the analog front-end noise [22]. If another ADC without an analog front end was chosen, the designer could choose a more appropriate antialiasing filter to the ADC input. Additionally, the designer should also choose a differential ADC to increase common mode rejection. This stems from the platform's noise floor at -40 dB. While mono audio signals are single ended, a single ended to differential converter should be implemented using an ADC driver chip or customized low-noise op-amp circuits. If the ADC is not replaced, then the 15 kHz filter could be compensated through another Digital FIR Filter block in Simulink. The primary disadvantage of this approach is the use of DSP slices.

Another improvement to the platform stems from the idea that the it should provide an industry-oriented high-speed signal processing solution. In an article published in RTC Magazine by a Xilinx employee, the author states that the benefits from using an FPGA over a serial DSP start with systems that are 1 MSPS to 500 MSPS. Although the ADS8681 has a sampling rate of 1 MSPS, the 15 kHz antialiasing filter limits the platforms applications to frequencies in the audio range. The ADC replacement that satisfies the aforementioned improvements should also have a faster sampling rate. The output DAC should have an improved settling time to accommodate this new system sampling rate.

The OPA4354 contains four operational amplifiers on one IC [25]. This amplifier was chosen due to the high slew rate and relatively low noise. However, experimental data clearly shows that there is noise that distorts any signal below -40 dB. While the quad op-amp array is appealing for space constraints, it requires trace routing on the board between the input buffer and then input to the ADC. Additionally, there is a trace from the input to the buffer. This could be reduced by having a very low noise high bandwidth ADC driver mentioned previously. A part to look into could be Texas Instrument's OPA2625 [31]. This could eliminate noise in the circuit.

The current platform requires two boards to function: the peripheral board and the Basys 3. The boards talk to each other over SPI due to the limited digital I/O ports. This could be resolved by using the CMOD A7-35T from Digilent. This FPGA board has a smaller footprint and is intended to be integrated on a breadboard. However, this module could be soldered into a new PCB design, thus eliminating the need for two boards. The FPGA used in this module is the same as the Basys 3, so no core functionality would be lost [32]. The access to significantly more I/O would allow for parallel interface ADCs and DACs. However, switches and LEDs would need to be integrated into the new PCB design for diagnostic purposes, as the breadboard ready module lacks diagnostic capabilities seen on the Basys 3. If higher order filters are required that exceed the capabilities of the Basys 3's 90 DSP slices, the Nexys 4 from Digilent integrates an FPGA with 240 DSP slices [12, 33]. The only changes that would need to be made would be the Vivado constraints file and the regeneration of System Generator code for the new target Xilinx FPGA.

Finally, the grounding of the data converters could be improved. Another approach to fix the -40 dB noise floor is improving the digital grounding of the ADC and DAC. Right now, the board has separate analog and digital ground planes, but the data converters' digital ground pins are connected to the digital ground plane. There are articles that argue that the digital ground of a data converter should be connected to the analog ground. If digital currents are low, then using a local 100 nF decoupling capacitor between the digital supply and digital ground pin very close to the IC could perform better [34].

X. BIBLIOGRAPHY

[1] Acromag Inc., “A Primer on FPGA-based DSP Applications,” May 14, 2009.

[2] A. H. A. Razak, M. I. A. Zaharin and N. Z. Haron, "Implementing digital finite impulse response filter using FPGA," *Applied Electromagnetics, 2007. APACE 2007. Asia-Pacific Conference on,* Melaka, 2007, pp. 1-5.

[3] Xilinx Inc., “7 Series FPGAs Overview,” 7 Series Datasheet, Sep. 27, 2016.

[4] E. Ozpolat, B. Karakaya, T. Kaya and A. Gulten, "FPGA-based digital Filter Design for Biomedical Signal," *2016 XII International Conference on Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, Lviv, 2016, pp. 70-73.

[5] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays, Fourth Edition*. Berlin, Germany: Springer-Verlag Berlin Heidelberg, 2014.

[6] S. Y. Park and P. K. Meher, "Efficient FPGA and ASIC Realizations of a DA-Based Reconfigurable FIR Digital Filter," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 7, pp. 511-515, July 2014.

[7] A. Mehrnia and A. N. Willson, Jr., “Optimal factoring of FIR filters,” U.S. Patent 9 391 591, Jul. 12, 2016.

[8] S. M. Rabiul Islam, et al., "Design of a programmable digital IIR filter based on FPGA," *Informatics, Electronics & Vision (ICIEV), 2012 International Conference on,* Dhaka, 2012, pp. 716-721.

[9] G. S. Gawande and K. B. Khanchandani, "Efficient Design and FPGA Implementation of Digital Filter for Audio Application," *Computing Communication Control and Automation (ICCUBEA), 2015 International Conference on,* Pune, 2015, pp. 906-910.

[10] International Electrotechnical Commission, “World Plugs,” *International Electrotechnical Commission*, 2016. [Online]. Available: http://www.iec.ch/worldplugs/list_bylocation.htm. [Accessed: 08 Oct. 2016].

[11] Pacific Gas and Electric Company, “Voltage Tolerance Boundary,” *Pacific Gas and Electric Company*, Jan. 1999. [Online]. Available: http://www.pge.com/includes/docs/pdfs/mybusiness/customerservice/energystatus/powerquality/voltage_tolerance.pdf. [Accessed: 14 Nov. 2016].

[12] Digilent, “Basys 3 FPGA Board Reference Manual,” Basys 3 Datasheet, Apr. 8, 2016.

[13] Digilent, “Genesys 2 Reference Manual,” Genesys 2 Datasheet, Oct. 7, 2015.

[14] Samsung, “Galaxy S7 Features and Specs,” Galaxy S7 Datasheet, Mar. 11, 2016.

[15] The MathWorks, Inc. (n.d.), *HDL Code Generation For Digital Filters*, [MathWorks video]. Available: https://www.mathworks.com/videos/hdl-code-generation-for-digital-filters-68760.html2016. [Accessed: 20 Oct. 2016].

[16] Xilinx Inc., “Xilinx DSP Design Platforms: Simplifying the Adoption of FPGAs for DSP,” White Paper: Spartan-6 and Virtex-6 FPGAs, Dec. 9, 2009.

[17] E. C. Vivas González, D. M. Rivera Pinzón and E. J. Gomez, "Implementation and simulation of IIR digital filters in FPGA using MatLab system generator," *Circuits and Systems (CWCAS), 2014 IEEE 5th Colombian Workshop on,* Bogota, 2014, pp. 1-5.

[18] R. Ford and C. Coulston, *Design for Electrical and Computer Engineers*, McGraw-Hill, 2007, p. 37

[19] National Instruments, “Advantages of Digital Filtering Compared to Analog Filtering,” LabVIEW 2012 Help, Jun. 2012.

[20] A. Taylor, “The ‘Ins and Outs’ of digital filter design and implementation,” *EE Times*. [Online], Available: http://www.eetimes.com/document.asp?doc_id=1279449. [Accessed Nov. 14, 2016].

[21] Texas Instruments, “Choose the Right A/D Converter for your application,” *Texas Instruments*, Jan. 2016. [Online]. Available: https://www.ti.com/europe/downloads/Choose%20the%20right%20data%20converter%20for%20your%20application.pdf. [Accessed: 6 Jun. 2017].

[22] Texas Instruments, “ADS868x 16-Bit, High-Speed, Single-Supply, SAR ADC Data Acquisition System with Programmable, Bipolar Input Ranges,” ADS8681 Datasheet, Dec. 2016.

[23] Texas Instruments, “DAC8811 16-bit, Serial Input Multiplying Digital-to-Analog Converter,” DAC8811 Datasheet, Feb. 2016.

[24] Burr-Brown Products, “16-bit, Ultra-Low Power, Voltage-Output Digital-to-Analog Converters,” DAC8830 Datasheet, Sep. 2007.

[25] Texas Instruments, “OPAx354 250-MHz, Rail-to-Rail I/O, CMOS Operational Amplifiers,” OPA4354 Datasheet, Jun. 2016.

[26] Radio-Electronics.com, “Op Amp Slew Rate: Tutorial; Formula; Calculator,” *Radio-Electronics.com*, 2016. [Online]. Available: http://www.radio-electronics.com/info/circuits/opamp_basics/operational-amplifier-slew-rate.php. [Accessed: 08 Oct. 2016].

[27] Microchip, “2.7V 4-Channel/8-Channel 12-Bit A/D Converters with SPI Serial Interface,” MCP3208 Datasheet, Sep. 2002.

[28] Microchip, “12-Bit DAC with SPI Interface,” MCP4921 Datasheet, Feb. 2007.

[29] Linear Technology, “3A Low Dropout Regulators with Micropower Quiescent Current and Shutdown,” LT1529 Datasheet, Mar. 2005.

[30] Triad Magnetics, “WSU060-4000,” WSU060-4000 Datasheet, Jan. 24, 2007.

[31] Texas Instruments, “OPAx625 High-Bandwidth, High-Precision, Low THD+N, 16-Bit and 18-Bit Analog-to-Digital Converter (ADC) Drivers,” OPA2625 Datasheet, Oct. 2015.

[32] Digilent, “Cmod A7 Reference Manual,” Cmod A7 Datasheet, Jun. 24, 2016.

[33] Digilent, “Nexys 4 DDR Reference Manual,” Nexys 4 DDR Datasheet, 2016.

[34] H. Zumbahlen, “Staying Well Grounded,” *AnalogDialogue*. [Online], Available: http://www.analog.com/en/analog-dialogue/articles/staying-well-grounded.html. [Accessed Jun. 6, 2016].

[35] R. Chepesiuk, “Where the chips fall: environmental health in the semiconductor industry.,” *Environmental Health Perspectives*., 107(9), A452-A457, Sep. 1999.

[36] U.S. Energy Information Administration, “Coal Explained: Coal and the Environment,” Dec. 10, 2015. [Online]. Available: http://www.eia.gov/energyexplained/?page=coal_environment. [Accessed: 03-Nov-2016].

[37] International Rectifier Corp., Application Note 955, “Protecting IGBTs and MOSFETs from ESD,” International Rectifier Corp., Web, <http://www.infineon.com/dgdl/an-955.pdf?fileId=5546d462533600a40153559f06a511cc>

[38] IEEE, “IEEE Code of Ethics,” 2016, [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 17-Nov-2016].

[39] A. R. Blanco and R. Villaecija, “Blood and minerals: Who profits from conflict in DRC?,” Jan. 19 2016, [Online]. Available: http://www.aljazeera.com/indepth/features/2016/01/blood-minerals-profits-conflict-drc-160118124123342.html. [Accessed: 03-Nov-2016]

[40] E. Dou, “China’s Tech Factories Turn to Student Labor,” Sep. 24, 2014, [Online]. Available: http://www.wsj.com/articles/chinas-tech-factories-turn-to-student-labor-1411572448. [Accessed: 03-Nov-2016].

[41] S.K. Rastogi et al, “Long-term effects of soldering fumes upon respiratory symptoms and pulmonary functions.,” National Institutes of Health., 35(3), 299-307, Jun. 1991.

APPENDICES

Appendix A – Analysis of Senior Project Design

Project Title: General-Purpose Digital Filter Platform
Student's Name: Michael Cheng
Student's Signature:

Advisor's Name: Wayne Pilkington
Advisor's Initials: WCP

Date:

1. Summary of Functional Requirements
   The General-Purpose Digital Filter Platform affords EE 459 students the opportunity to work with hardware solutions like those found in industry. The filter parallelizes signal processing operations, improving on the previous platform's serial execution nature. It then digitizes single channel audio signals at 44.1 kHz sampling rate with 16-bit precision or 1 MHz sampling at 8-bit precision. After sampling an audio signal, a user programs a customizable filter to process the input signal. Operating at 1 Mega-sample per second, the new general-purpose platform simultaneously processes at least ten digital filtering difference equation coefficients. The platform can then output the desired result.

2. Primary Constraints
   Choosing an appropriate FPGA for this project proves difficult. Some FPGAs come integrated onto a platform, whereas others only come as single chips. Integrating a preexisting product limits the overall platform design, where the single chip method allows for greater control over platform design. However, designing a board with only the FPGA increases board layout difficulty through the numerous pins requiring testing. Parts selection also proved difficult through balancing the need for high performance parts with low costs. Additionally, the added Xilinx System Generator and HDL Coder requirement limits FPGA selection. These constraints defined the requirements and specifications found in Chapter 2.

3. Economic
   This platform benefits EE 459 instructors and students through providing a modern platform to learn signal processing. This translates to increased exposure to relevant industry techniques, which ultimately helps students find employment after graduation.

   This project's economic impact proves substantial through the large cost associated with developing the platform. Increased production of these platforms reduce the overall cost of the project, but EE 459 requires sharing equipment limiting amortizable costs. EE 459 students ultimately incur the final equipment cost through either assembling, obtaining, or maintaining the platform. The platform outweighs initial costs when students find employment because of the platform.
   Natural resource depletion serves as the biggest unredeemable sunk cost. Fabricating printed circuit boards and various electronic components require silicon, carbon, and

other semiconductor doping chemicals. The use of these chemicals depletes the earth's natural resources and makes it extremely difficult to reuse these materials [35].

The platform development requires numerous electronic components and supporting equipment. Initial material cost estimates amount to $550. The developer incurs all component costs, while the electrical engineering department provides all test equipment. The developer's tuition costs cover test equipment usage fees. Upon development completion, product costs shift from developer to customers. For complete listing of initial cost estimates, table 3 lists all projected costs.
Figure 2 provides a Gantt chart projecting project completion at the end of May 2017. The EE 459 instructor decides on the project's lifetime, depending on how long the project remains a relevant teaching aid.

4. If Manufactured on a Commercial Basis
If manufactured on a commercial basis, fifty platforms would sell during one year. The fabrication costs associated with each device would then amount to approximately $150. Customers would purchase the platform for $200 per device, yielding a $2500 profit margin in one year. The estimated cost for one user to operate the device for a given time equals the utility's electricity cost at that particular time period.

5. Environmental
Semiconductors prove vital to producing a functioning electronic platform. However, manufacturing semiconductors produces harmful chemical waste byproducts. These byproducts spill into water supplies and certain ground areas, damaging ecosystems and harming people [35]. Additionally, the platform's energy requirements mandate the generation of electricity. Electricity use requires different methods ranging from coal burning to solar panel generation. Coal burning releases greenhouse gases into the atmosphere, thereby increasing environmental impact [36]. Altering ecosystems may drive certain species to extinction, while causing ill health effects in other species. The inhalation of carcinogenic atmospheric waste proves lethal to humans.

6. Manufacturability
Integrating an existing FPGA product into the overall platform causes significant manufacturing issues. The potential use of third party FPGA products limits implementation of the final product's printed circuit board. Using a standalone FPGA chip furthers fabrication complexity because most FPGAs come in BGA packaging. With hundreds of pins, the BGA package makes it extremely difficult to test pins on the chip. This severely affects the verification phase during manufacturing.

7. Sustainability
The platform consists of static-electricity susceptible metal oxide field-effect transistors. Over the lifetime of the product, users might subject the system to static electricity, thereby damaging components beyond repair [37]. Some of these field-effect transistor integrated circuits require extensive maintenance work to repair. Programming support might also dwindle over time. This proves devastating to sustaining platform productivity. The platform's physical connectors degrade over

time through corrosion and wear. The user must replace these connectors as needed to ensure electrical conductivity and mechanical strength.

Platform development and production requires using different elements to create different chemical compounds. Some chemicals prove extremely difficult to separate back into their elemental forms. This prevents the reuse of these elements for different uses.

Bypassing the integration of third-party products into the platform would significantly improve the design of the project and lower end-user costs. However, this would significantly increase development costs and increase testing difficulty.

8. Ethical
   With regards to IEEE's Code of Ethics, the following ethical dilemmas arise [38].

   Certain semiconductor fabrication methods require rare-earth metals to ensure product functionality and robustness. The acquisition of these materials causes civil wars in the Democratic Republic of the Congo. These conflict materials prove extremely valuable through their integral role in producing electronics. By producing a product using these materials, the platform indirectly funds a civil war that devastates life in other parts of the world [39].

   Additionally, lab groups share the platform in a laboratory scenario. By using a shared resource, conflicts of interest might arise through arguments about who should keep the platform after the quarter ends. These arguments might include the use of bribery or physical violence over property.

   The platform requires computers for programming filter designs. This extends the ethical dilemma to computer fabrication. In addition to using conflict materials, personal computer manufacturers abuse factory workers through long work hours that violate labor laws [40].

   The platform manufacturing indirectly supports harming people in the product's supply chain. The product's use directly harms users involved through conflict stemming from sharing resources.

   From previous issues, it remains clear that human harm violates multiple IEEE codes. The platform must take precautions to avoid potential ethical violations. Additionally, the ethical issues still conflict with other ethical frameworks. The Platinum Rule states that people should treat others the way they would like to be treated. The human labor abuses and violence definitively produce the same ethical problems. Again, platform development must take appropriate precautions to avoid ethical mishaps.

9. Health and Safety
   Platform manufacturing produces health and safety concerns through using solder. Platform integration requires soldering to join printed circuit board components. This

solder contains lead, a chemical causing disabilities and harm to humans. Solder often contains rosin flux that when burned, creates smoke that pollutes the atmosphere with metal oxides. People might inhale these fumes, while exposing themselves to lead [41]. These lead to adverse health effects.

Operating the platform requires using electricity. The user might electrocute themselves on a potentially exposed electrical surface.

10. Social and Political
    The project impacts EE 459 students and instructors through directly providing a tool for learning. The platform indirectly impacts employers that hire EE 459 students based on experience with modern design techniques used in the product. The favoring of employing EE 459 students in this area displaces other potential candidates from filling potential employment positions and may generate a wage disparity favorable to those who have taken EE 459, as opposed to those who have not.

11. Development
    This project required learning printed circuit board design and layout. Additionally, it increased familiarity with FPGA based designs and the various tools to program them with customized digital filters. The project ultimately increased familiarity with bringing a product to market.

    For resources used in developing this project, one should refer to the references section found earlier in this document. The references reveal implementation methodologies associated with FPGA-based FIR and IIR filter designs. Additionally, it also describes working filters that were designed in Simulink. I had to learn how to use Vivado System Generator to translate Simulink designs to HDL code suitable for the FPGA. Additionally, I had to learn how FPGA architectures greatly improve parallelized processing.

Appendix B – Parts List and Costs

Table 7 outlines the projected project costs. Material costs include hardware and software costs associated with constructing the platform. Ideally, the material costs would amount to \$100, but realistic estimates amount to \$500. At worst, material costs could reach \$1000. Using Ford and Coulston's cost estimation formula found in Chapter 10.4.2, the projected material costs incurred equal \$550 [18]. Extending this formula to other costs, labor costs accounted for a wage of \$30 per hour. This compares with industry wage rates for senior undergraduate interns who have not received their degree. The entire project encompasses 35 weeks. For approximately ten hours per week, the labor costs for the project comes to \$10500. Material and shipping costs reflect projected parts costs along with the shipping charges associated with delivering said parts. Consultation costs accrue through advisor related interactions. Since advisors have graduate level degrees, they have an estimated wage cost of \$200 per hour. Assuming ten hours of cumulative consultation throughout the quarter, the total consultation cost amounts to \$2000. Test equipment fees stem from tuition payments. By making tuition payments, the Cal Poly Electrical Engineering department makes equipment available to students without incurring additional charges. Three tuition payments allow for test equipment usage during the entire project cycle.

TABLE VII
General-Purpose Digital Filter Platform Cost Estimate

Item	Total Cost
Material	\$550
Labor	\$10500
Shipping	\$50
Consultation	\$2000
Test Equipment Usage Fee	\$9075

Throughout development of the product, there were multiple design iterations that added to the overall material costs. Table 8 details the component costs for one platform. This does not include the cost of the board fabrication. There was a total of three board designs, which were all fabricated through OSH Park. Revision A cost \$33.95 to fabricate three copies of the initial PCB design. Revision B cost \$33.95 to fabricate another three copies. Finally, revision C cost \$28.35 to fabricate, but time constraints required OSH Park's super swift service. Coupled with fast shipping, the three boards cost \$56.70 to make. The development cost from board layouts totaled \$124.60. Three sets of parts in Table 8 were ordered to test each revision. However, only two power supplies were purchased, and only one development board was used. This development board was not purchased but was borrowed from school. This brought total component costs to \$233.08. The total material costs were \$357.68, which includes both component and board layout expenses. The project took the full 35 weeks to develop, and all other cost estimates were accurate. If 1000 platforms were manufactured per year, the per unit material cost would drop from \$162.52 to \$124.51 using Digi-Key bulk pricing as of May 27, 2017.

TABLE VIII

GENERAL-PURPOSE DIGITAL FILTER PLATFORM PARTS LIST

Item	**Manufacturer**	**Cost per Unit**	**Number of Units**	**Total Cost**	**Description**
RAPC722X	Switchcraft Inc.	$0.90	1	$0.90	5.5mm OD, 2.0mm ID, DC Barrel Jack
LT1529CQ-5#PBF	Linear Technology	$7.41	2	$14.82	5V linear voltage regulator
ADS8681IPWR	Texas Instruments	$15.57	1	$15.57	16-bit, SAR, 1MSPS ADC
DAC8830ID	Texas Instruments	$13.51	1	$13.51	16-bit, R-2R, 1us DAC
OPA4354AIDR	Texas Instruments	$3.93	1	$3.93	Quad CMOS OP Amp array
PPTC062LJBN-RC	Sullins Connector Solutions	$1.53	1	$1.53	2x6 pin jack, 0.1" pitch
MJ-3523-SMT-TR	CUI Inc.	$1.02	2	$2.04	3.5mm audio jack, SMT
731000154	Molex, LLC	$1.61	2	$3.22	BNC jack, bayonet lock
M20-9990246	Harwin Inc.	$0.11	1	$0.11	2-pin header, 0.1" pitch
EG1218	E-Switch	$0.58	2	$1.16	SPDT switch, through hole
RC0603FR-0710KL	Yageo	$0.10	3	$0.30	10k resistor, 1%, 0603 SMT
RC0603FR-071KL	Yageo	$0.10	1	$0.10	1k resistor, 1%, 0603 SMT
RC0603FR-073K3L	Yageo	$0.10	1	$0.10	3.3k resistor, 1%, 0603 SMT
CC0603KRX7R7BB105	Yageo	$0.10	6	$0.60	1uF ceramic, 10%, 0603 SMT
CC0603KRX7R7BB104	Yageo	$0.10	2	$0.20	0.1uF ceramic, 10%, 0603 SMT
CC0603KRX7R9BB103	Yageo	$0.10	2	$0.20	10nF ceramic, 10%, 0603 SMT
CC0603KRX7R9BB331	Yageo	$0.10	1	$0.10	330pF ceramic, 10%, 0603 SMT
F951A106MPAAQ2	AVX Corporation	$0.79	4	$3.16	10uF tantalum, 20%, 0805 SMT
F981D105MMA	AVX Corporation	$0.72	1	$0.72	1uF tantalum, 20%, 0603 SMT
CC0805MKX5R5BB226	Yageo	$0.26	2	$0.52	22uF ceramic, 20%, 0805 SMT
CC0603KRX5R6BB475	Yageo	$0.15	1	$0.15	4.7uF ceramic, 10%, 0603 SMT
24432	Keystone Electronics	$0.41	4	$1.64	M3 Aluminum standoff
MPMS 003 0005 PH	B&F Fastener Supply	$0.0594	4	$0.24	M3 Zinc Screw
5027	Keystone Electronics	$0.45	2	$0.90	Scope Probe Test Point
WSU060-4000	Triad Magnetics	$17.28	1	$17.28	5.9V Wall Supply
410-183	Digilent, Inc.	$149.00	1	$79.00	Basys 3 FPGA Development Board
PRPC006DABN-RC	Sullins Connector Solutions	$0.52	1	$0.52	2x6 Board Interconnect
PEC36SFAN	Sullins Connector Solutions	$1.94	1	$1.94	Scope Probe Ground Pin
UWT1C471MNL1GS	Nichicon	$0.49	1	$0.49	470 uF Aluminum capacitor, SMT
Grand Total:				$162.52	

APPENDIX C – PROJECT SCHEDULE – TIME ESTIMATES AND ACTUALS

Table 6 describes the deliverables time table for the project. To facilitate manageable goals, Dr. Braun provided generalized completion times for different checkpoints during EE 461 and 462. The table ultimately describes the project progression toward platform completion.

TABLE IX
GENERAL-PURPOSE DIGITAL FILTER PLATFORM DELIVERABLES

Delivery Date	Deliverable Description
Dec 2016	Project Plan Review
Feb 16, 2017	Design Review
Mar 6, 2017	EE 461 demo
Mar 19, 2017	EE 461 report
May 2, 2017	EE 462 demo
Jun 2, 2017	Sr. Project Expo Poster
Jun 16, 2017	ABET Sr. Project Analysis
Jun 16, 2017	EE 462 Report

Figure 13 displays the project's Gantt chart. This Gantt chart spans the entire academic year and includes checkpoints and due dates for various items in the project plan. For the implementation phases of the project, multiple design-build-test iterations allow for adequate time to revise various platform aspects. During Winter 2017, two FPGA filter testing cycles provide time to ensure the platform uses the right FPGA part for filtering. During Spring 2017, two board testing cycles provide time to ensure the final platform meets project requirements and specifications. Project completion happens well before the eighth week of May to ensure proper preparation for the Senior Project Expo in spring.

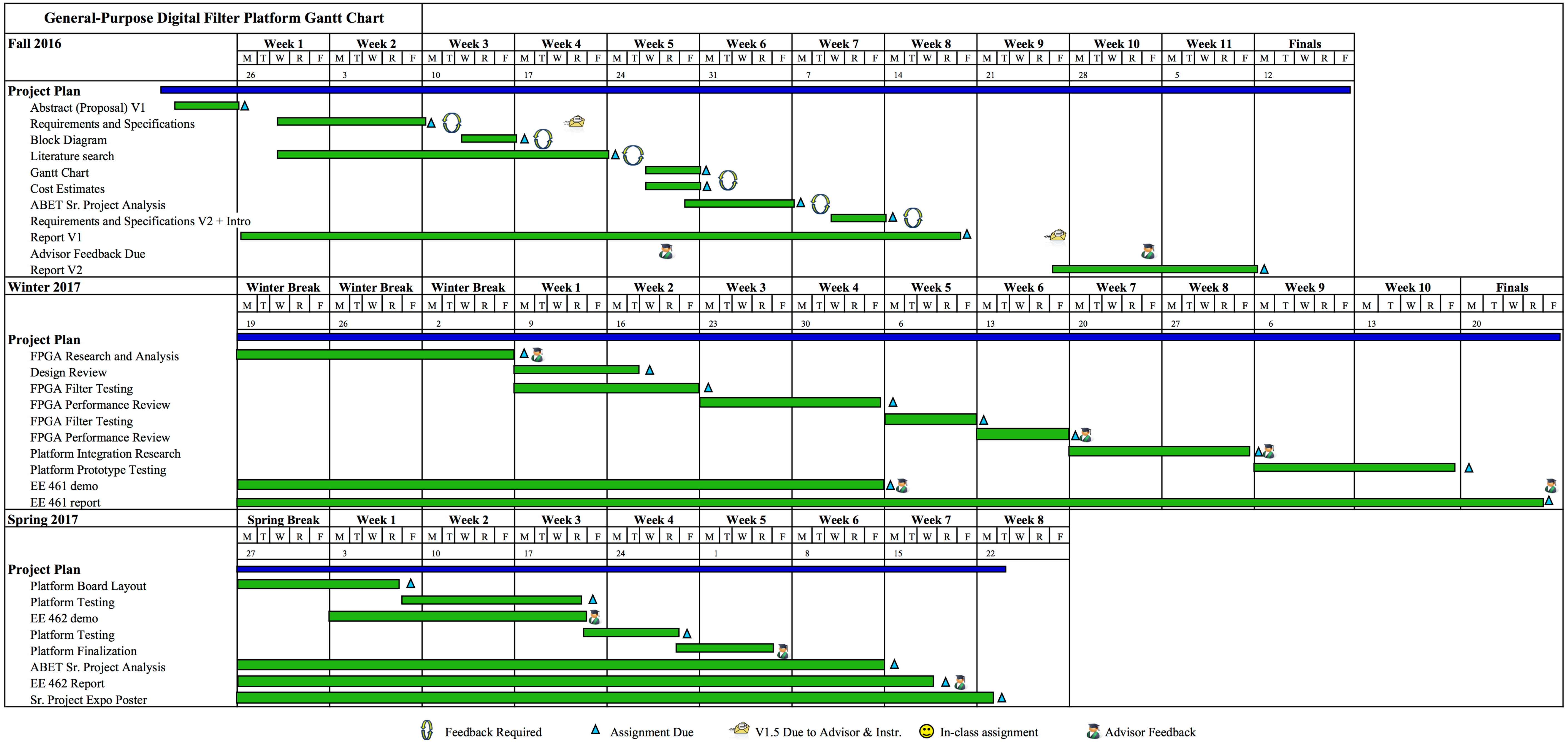


FIGURE XIII
GENERAL-PURPOSE DIGITAL FILTER PROJECT GANTT CHART

Appendix D – PC Board Layout

Figures 14 and 15 detail the exact placement of all components listed in the figure 9 schematic. The PCB has 2 layers with separate analog and digital ground planes. These planes meet underneath the ADC on the bottom side of the board. Most components are grounded through the analog ground plane except for the digital circuitry of the data converters and the FPGA support hardware.

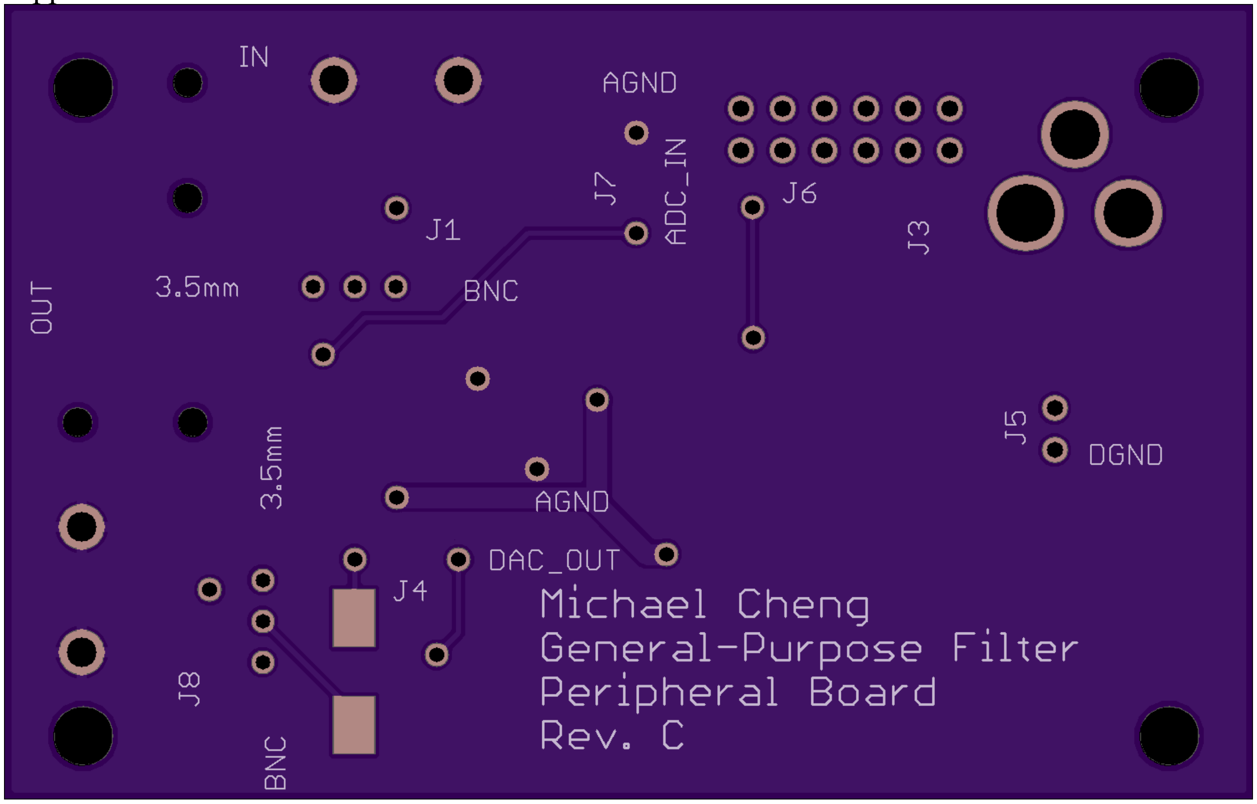


FIGURE XIV
BOARD LAYOUT TOP VIEW

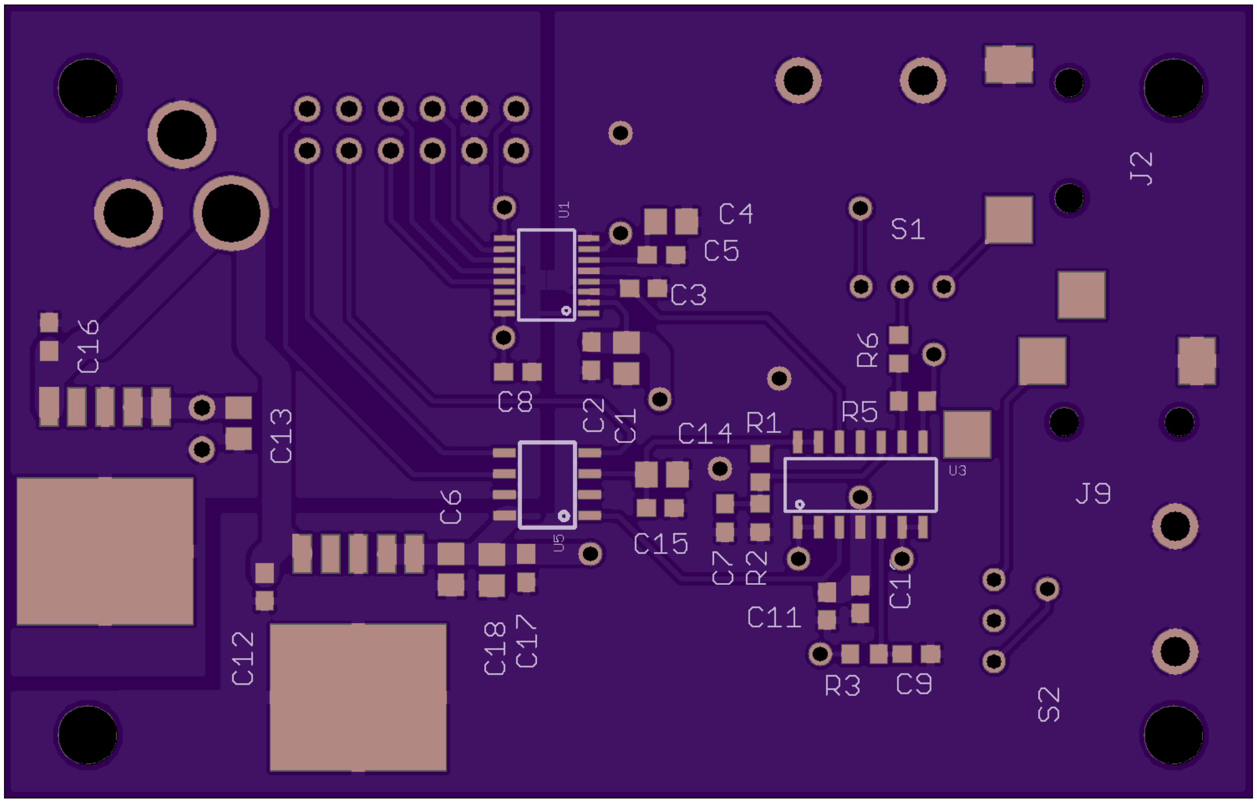


FIGURE XV
BOARD LAYOUT BOTTOM VIEW

APPENDIX E – PROGRAM LISTINGS

This project relied on the following programs:

- MATLAB r2016b
- EAGLE 8.1.1
- Xilinx Vivado Design Suite HL System Edition 2016.4

filter_system_wrapper.vhd

```vhdl
----------------------------------------------------------------------------------
-- Company: Cal Poly
-- Engineer: Michael Cheng
--
-- Create Date: 05/14/2017 08:32:22 PM
-- Design Name: Digital Filter System Wrapper
-- Module Name: filter_system_wrapper - Behavioral
-- Project Name: General-Purpose Digital Filter Platform
-- Target Devices: Basys 3/ Xilinx XC7A35T-1CPG236C
-- Tool Versions: Vivado 2016.4, System Generator 2016.4, MATLAB R2016b
-- Description: This module is the system wrapper to communicate with the General-
--              Purpose Digital Filter Peripheral Board Rev. C.  It uses SPI to
--              communicate with the TI ADS8681 ADC and TI DAC8830 DAC.  The FIR
--              filter support logic used is imported through Vivado's IP
--              Integrator.  See ADC and DAC datasheets for more information on
--              SPI transaction protocols.
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity filter_system_wrapper is
    Port ( adcCS    : BUFFER STD_LOGIC_VECTOR (0 downto 0);
           adcMISO  : IN     STD_LOGIC;
           CLK      : IN     STD_LOGIC;
           adcCLK   : BUFFER STD_LOGIC;
           LED      : OUT    STD_LOGIC_VECTOR (15 downto 0);
           dacCS    : BUFFER STD_LOGIC_VECTOR (0 downto 0);
           dacMOSI  : OUT    STD_LOGIC;
           adcMOSI  : OUT    STD_LOGIC;
           reset    : IN     STD_LOGIC;
           dacCLK   : BUFFER STD_LOGIC;
           SWITCHES : IN     STD_LOGIC_VECTOR (15 downto 0));
end filter_system_wrapper;

architecture Behavioral of filter_system_wrapper is
    -- There are three states:
    -- initialization - init
    -- filtering - filt
    -- diagnostic - diag
    TYPE State_type IS (init, filt, diag);
         SIGNAL State : State_Type; -- creating State signal

         -- This module contains all digital filtering support logic.
         -- It is generated using the Simulink GUI through
         -- Vivado System Generator.  This block always must have a 1 MHz
         -- input clock.  The gateway_in input connects to the received ADC
         -- value, and the gateway_out block connects to the transmitted
         -- DAC value.
    component FIR_block_wrapper is
      port (
        clk         : IN  STD_LOGIC;
        gateway_in  : IN  STD_LOGIC_VECTOR ( 15 downto 0 );
        gateway_out : OUT STD_LOGIC_VECTOR ( 15 downto 0 )
      );
    end component;

    component spi_master is
    GENERIC(
        slaves  : INTEGER; --number of spi slaves
        d_width : INTEGER); --data bus width
      PORT(
        clock   : IN     STD_LOGIC;                             --system clock
        reset_n : IN     STD_LOGIC;                             --asynchronous reset
        enable  : IN     STD_LOGIC;                             --initiate transaction
        cpol    : IN     STD_LOGIC;                             --spi clock polarity
        cpha    : IN     STD_LOGIC;                             --spi clock phase
        cont    : IN     STD_LOGIC;                             --continuous mode command
        clk_div : IN     INTEGER;                               --system clock cycles per 1/2 period of sclk
        addr    : IN     INTEGER;                               --address of slave
        tx_data : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
        miso    : IN     STD_LOGIC;                             --master in, slave out
        sclk    : BUFFER STD_LOGIC;                             --spi clock
        ss_n    : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
        mosi    : OUT    STD_LOGIC;                             --master out, slave in
        busy    : OUT    STD_LOGIC;                             --busy / data ready signal
        rx_data : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received
    end component;

    component clock_divider is
    PORT (
        clk         : IN  STD_LOGIC; -- system clock
        reset       : IN  STD_LOGIC; -- asynchronous reset
        clock_out   : OUT STD_LOGIC); -- divided clock
    end component;

    signal adcBusy          : STD_LOGIC; --busy / data ready signal
    signal dacBusy          : STD_LOGIC; --busy / data ready signal
    signal s_adc_rx         : STD_LOGIC_VECTOR (31 downto 0):=x"00000000"; -- received ADC data word
    signal s_dac_tx         : STD_LOGIC_VECTOR (15 downto 0):=x"0000"; -- value transmitted to DAC
    signal sclk             : STD_LOGIC; -- divided clock
    signal dac_conversion   : STD_LOGIC_VECTOR(15 downto 0); -- output value from filter
    signal s_adc_tx         : STD_LOGIC_VECTOR (31 downto 0) := x"00000000"; -- value transmitted to ADC
begin
    filter: FIR_block_wrapper PORT MAP (
        clk         => sclk, -- 1 MHz clock
        gateway_in  => s_adc_rx(30 downto 15), -- ignore the first bit received due to
                                               -- synchronization issues.  Quantized value
        gateway_out => dac_conversion -- filtered value to output to DAC
    );
    adcspi: spi_master
    GENERIC MAP (
        slaves  => 1,    -- only one slave
        d_width => 32)   -- ADC deals in 32-bit SPI transactions
    PORT MAP (
        clock   => CLK,  -- system clock for high-speed serial clocks
        reset_n => '1',  -- never reset
        enable  => sclk, -- 1 MSPS
        cpol    => '0',  -- standard SPI polarity
        cpha    => '0',  -- standard SPI phase
        cont    => '0',  -- not continuous mode, see developer's notes for details
        clk_div => 1,    -- 50 MHz serial clock
        addr    => 0,    -- arbitrary address, see developer's notes for details
        tx_data => s_adc_tx, -- data to transmit to ADC
        miso    => adcMISO,
        sclk    => adcCLK,
        ss_n    => adcCS,
        mosi    => adcMOSI,
        busy    => adcBusy, -- indicator flags from ADC
        rx_data => s_adc_rx -- data received from ADC
    );
    dacspi: spi_master
    GENERIC MAP (
        slaves  => 1, -- only one slave
        d_width => 16) -- DAC deals in 16-bit SPI transactions
    PORT MAP (
        clock   => CLK, -- system clock for high-speed serial clocks
        reset_n => '1', -- never reset
        enable  => sclk, -- complements 1 MSPS from ADC
        cpol    => '0', -- standard SPI polarity
        cpha    => '0', -- standard SPI phase
        cont    => '0', -- not continuous mode, see developer's notes for details
        clk_div => 1, -- 50 MHz serial clock
        addr    => 0, -- arbitrary address, see developer's notes for details
        tx_data => s_dac_tx, -- data to transmit to DAC
        miso    => '0', -- DAC does not return data
        sclk    => dacCLK,
        ss_n    => dacCS,
        mosi    => dacMOSI,
        busy    => dacBusy, -- DAC indicator flag
        rx_data => open -- no data received from DAC
    );
    clkdiv: clock_divider
    PORT MAP (
        clk       => CLK, -- 100 MHz input clock
        reset     => '1', -- never reset
        clock_out => sclk -- 1 MHz output clock
    );

    -- Finite State Machine - System Controller
    PROCESS (sCLK, reset) -- each state transitions at 1 MHz with an optional reset
      BEGIN
        If (reset = '1') THEN -- Upon asynchronous reset, set the state to init
        State <= init;

        ELSIF rising_edge(sCLK) THEN    -- else if there is a rising edge of the
                                        -- clock, then do the stuff below

        -- The CASE statement checks the value of the State variable,
        -- and based on the value and any other control signals, changes
        -- to a new state.
        CASE State IS
            -- The initialization state configures the appropriate input range
            -- for the ADS8681 ADC.  This makes the input range from 0 to 5.12 V.
            -- The state unconditionally transitions to the filtering state.
            WHEN init =>
                s_adc_tx <= x"D014000B"; -- write 0x0B to address 0x14
                State    <= filt;

            -- If the current state is filt, and the first switch is set, then
            -- the next state is a diagnostic state.  If the switch is not set,
            -- then the next state is another filtering state.
            WHEN filt =>
                s_adc_tx  <= x"00000000"; -- No-operation instruction to read ADC
                s_dac_tx  <= dac_conversion; -- send filtered value to DAC
                if (SWITCHES = x"0001") then
                    State <= diag;
                elsif (SWITCHES = x"0000") then
                    State <= filt;
                end if;

            -- If the current state is diag, and the first switch is set, then
            -- the next state is another diagnostic state.  If the switch is
            -- not set, then the next state is the filtering state.
            WHEN diag =>
                s_adc_tx  <= x"00000000"; -- No-operation instruction to read ADC
                s_dac_tx  <= s_adc_rx(30 downto 15); -- direct passthrough to DAC
                if (SWITCHES = x"0001") then
                    State <= diag;
                elsif (SWITCHES = x"0000") then
                    State <= filt;
                end if;

            -- if trapped in unknown state, loop initialization protocols
            WHEN others =>
                State <= init;
        END CASE;
        END IF;
      END PROCESS;

    LED <= SWITCHES; -- LED to indicate if a switch is flipped

end Behavioral;
```

clock_divider.vhd

```
----------------------------------------------------------------------------------
-- Company: Cal Poly
-- Engineer: Michael Cheng
--
-- Create Date: 05/14/2017 08:32:22 PM
-- Design Name: 1 MHz Clock Divider
-- Module Name: clock_divider - Behavioral
-- Project Name: General-Purpose Digital Filter Platform
-- Target Devices: Basys 3/ Xilinx XC7A35T-1CPG236C
-- Tool Versions: Vivado 2016.4, System Generator 2016.4, MATLAB R2016b
-- Description: This module divides the 100 MHz system clock down to 1 MHz.
--              Every 50 system clock cycles, the temporary signal inverts
--              to generate the 1 MHz square wave.
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clock_divider is
    Port ( clk       : IN  STD_LOGIC;
           reset     : IN  STD_LOGIC;
           clock_out : OUT STD_LOGIC);
end clock_divider;

architecture Behavioral of clock_divider is

signal count : INTEGER range 0 to 1000 :=0;
signal tmp   : STD_LOGIC :='0';

begin

process(clk,reset)
begin
    -- asynchronous reset
    if(reset='0') then
        count <= 0;
        tmp   <= '0';

    -- 1 MHz clock divider
    elsif(rising_edge(clk)) then
        -- invert value every 50 cycles of 100 MHz
        -- to generate the 1 MHz clock
        if (count = 50) then
            tmp   <= not(tmp); -- invert
            count <= 0; --reset counter
        else
            count <= count + 1; -- increment delay
        end if;
    end if;
end process;

clock_out <= tmp; -- output divided value

end Behavioral;
```

spi_master.vhd

```
--------------------------------------------------------------------------------
--
--   FileName:         spi_master.vhd
--   Dependencies:     none
--   Design Software:  Quartus II Version 9.0 Build 132 SJ Full Version
--
--   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
--
--   Version History
--   Version 1.0 7/23/2010 Scott Larson
--     Initial Public Release
--   Version 1.1 4/11/2013 Scott Larson
--     Corrected ModelSim simulation error (explicitly reset clk_toggles signal)
--
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY spi_master IS
  GENERIC(
    slaves  : INTEGER := 1;  --number of spi slaves
    d_width : INTEGER := 19); --data bus width
  PORT(
    clock   : IN     STD_LOGIC;                             --system clock
    reset_n : IN     STD_LOGIC;                             --asynchronous reset
    enable  : IN     STD_LOGIC;                             --initiate transaction
    cpol    : IN     STD_LOGIC;                             --spi clock polarity
    cpha    : IN     STD_LOGIC;                             --spi clock phase
    cont    : IN     STD_LOGIC;                             --continuous mode command
    clk_div : IN     INTEGER;                               --system clock cycles per 1/2 period of sclk
    addr    : IN     INTEGER;                               --address of slave
    tx_data : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
    miso    : IN     STD_LOGIC;                             --master in, slave out
    sclk    : BUFFER STD_LOGIC;                             --spi clock
    ss_n    : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
    mosi    : OUT    STD_LOGIC;                             --master out, slave in
    busy    : OUT    STD_LOGIC;                             --busy / data ready signal
    rx_data : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received
END spi_master;

ARCHITECTURE logic OF spi_master IS
  TYPE machine IS(ready, execute);                           --state machine data type
  SIGNAL state       : machine;                              --current state
  SIGNAL slave       : INTEGER;                              --slave selected for current transaction
  SIGNAL clk_ratio   : INTEGER;                              --current clk_div
  SIGNAL count       : INTEGER;                              --counter to trigger sclk from system clock
  SIGNAL clk_toggles : INTEGER RANGE 0 TO d_width*2 + 1;     --count spi clock toggles
  SIGNAL assert_data : STD_LOGIC;                            --'1' is tx sclk toggle, '0' is rx sclk toggle
  SIGNAL continue    : STD_LOGIC;                            --flag to continue transaction
  SIGNAL rx_buffer   : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --receive data buffer
  SIGNAL tx_buffer   : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --transmit data buffer
  SIGNAL last_bit_rx : INTEGER RANGE 0 TO d_width*2;         --last rx data bit location
BEGIN
  PROCESS(clock, reset_n)
  BEGIN

    IF(reset_n = '0') THEN        --reset system
      busy <= '1';                --set busy signal
      ss_n <= (OTHERS => '1');    --deassert all slave select lines
      mosi <= 'Z';                --set master out to high impedance
      rx_data <= (OTHERS => '0'); --clear receive data port
      state <= ready;             --go to ready state when reset is exited

    ELSIF(clock'EVENT AND clock = '1') THEN
      CASE state IS               --state machine

        WHEN ready =>
          busy <= '0';             --clock out not busy signal
          ss_n <= (OTHERS => '1'); --set all slave select outputs high
          mosi <= 'Z';             --set mosi output high impedance
          continue <= '0';         --clear continue flag

          --user input to initiate transaction
          IF(enable = '1') THEN
            busy <= '1';             --set busy signal
            IF(addr < slaves) THEN   --check for valid slave address
              slave <= addr;         --clock in current slave selection if valid
            ELSE
              slave <= 0;            --set to first slave if not valid
            END IF;
            IF(clk_div = 0) THEN     --check for valid spi speed
              clk_ratio <= 1;        --set to maximum speed if zero
              count <= 1;            --initiate system-to-spi clock counter
            ELSE
              clk_ratio <= clk_div;  --set to input selection if valid
              count <= clk_div;      --initiate system-to-spi clock counter
            END IF;
            sclk <= cpol;            --set spi clock polarity
            assert_data <= NOT cpha; --set spi clock phase
            tx_buffer <= tx_data;    --clock in data for transmit into buffer
            clk_toggles <= 0;        --initiate clock toggle counter
            last_bit_rx <= d_width*2 + conv_integer(cpha) - 1; --set last rx data bit
            state <= execute;        --proceed to execute state
          ELSE
```

```
            state <= ready;           --remain in ready state
          END IF;

        WHEN execute =>
          busy <= '1';         --set busy signal
          ss_n(slave) <= '0'; --set proper slave select output

          --system clock to sclk ratio is met
          IF(count = clk_ratio) THEN
            count <= 1;                     --reset system-to-spi clock counter
            assert_data <= NOT assert_data; --switch transmit/receive indicator
            IF(clk_toggles = d_width*2 + 1) THEN
              clk_toggles <= 0;               --reset spi clock toggles counter
            ELSE
              clk_toggles <= clk_toggles + 1; --increment spi clock toggles counter
            END IF;

            --spi clock toggle needed
            IF(clk_toggles <= d_width*2 AND ss_n(slave) = '0') THEN
              sclk <= NOT sclk; --toggle spi clock
            END IF;

            --receive spi clock toggle
            IF(assert_data = '0' AND clk_toggles < last_bit_rx + 1 AND ss_n(slave) = '0') THEN
              rx_buffer <= rx_buffer(d_width-2 DOWNTO 0) & miso; --shift in received bit
            END IF;

            --transmit spi clock toggle
            IF(assert_data = '1' AND clk_toggles < last_bit_rx) THEN
              mosi <= tx_buffer(d_width-1);                     --clock out data bit
              tx_buffer <= tx_buffer(d_width-2 DOWNTO 0) & '0'; --shift data transmit buffer
            END IF;

            --last data receive, but continue
            IF(clk_toggles = last_bit_rx AND cont = '1') THEN
              tx_buffer <= tx_data;                       --reload transmit buffer
              clk_toggles <= last_bit_rx - d_width*2 + 1; --reset spi clock toggle counter
              continue <= '1';                            --set continue flag
            END IF;

            --normal end of transaction, but continue
            IF(continue = '1') THEN
              continue <= '0';      --clear continue flag
              busy <= '0';          --clock out signal that first receive data is ready
              rx_data <= rx_buffer; --clock out received data to output port
            END IF;

            --end of transaction
            IF((clk_toggles = d_width*2 + 1) AND cont = '0') THEN
              busy <= '0';             --clock out not busy signal
              ss_n <= (OTHERS => '1'); --set all slave selects high
              mosi <= 'Z';             --set mosi output high impedance
              rx_data <= rx_buffer;    --clock out received data to output port
              state <= ready;          --return to ready state
            ELSE                       --not end of transaction
              state <= execute;        --remain in execute state
            END IF;

          ELSE        --system clock to sclk ratio not met
            count <= count + 1; --increment counter
            state <= execute;   --remain in execute state
          END IF;

      END CASE;
    END IF;
  END PROCESS;
END logic;
```

Basys3_Master.xdc

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the
project

## Clock signal
set_property PACKAGE_PIN W5 [get_ports CLK]
        set_property IOSTANDARD LVCMOS33 [get_ports CLK]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports CLK]

# Switches
set_property PACKAGE_PIN V17 [get_ports {SWITCHES[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[0]}]
set_property PACKAGE_PIN V16 [get_ports {SWITCHES[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[1]}]
set_property PACKAGE_PIN W16 [get_ports {SWITCHES[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[2]}]
set_property PACKAGE_PIN W17 [get_ports {SWITCHES[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[3]}]
set_property PACKAGE_PIN W15 [get_ports {SWITCHES[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[4]}]
set_property PACKAGE_PIN V15 [get_ports {SWITCHES[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[5]}]
set_property PACKAGE_PIN W14 [get_ports {SWITCHES[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[6]}]
set_property PACKAGE_PIN W13 [get_ports {SWITCHES[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[7]}]
set_property PACKAGE_PIN V2 [get_ports {SWITCHES[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[8]}]
set_property PACKAGE_PIN T3 [get_ports {SWITCHES[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[9]}]
set_property PACKAGE_PIN T2 [get_ports {SWITCHES[10]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[10]}]
set_property PACKAGE_PIN R3 [get_ports {SWITCHES[11]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[11]}]
set_property PACKAGE_PIN W2 [get_ports {SWITCHES[12]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[12]}]
set_property PACKAGE_PIN U1 [get_ports {SWITCHES[13]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[13]}]
set_property PACKAGE_PIN T1 [get_ports {SWITCHES[14]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[14]}]
set_property PACKAGE_PIN R2 [get_ports {SWITCHES[15]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {SWITCHES[15]}]


## LEDs
set_property PACKAGE_PIN U16 [get_ports {LED[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]
set_property PACKAGE_PIN E19 [get_ports {LED[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]
set_property PACKAGE_PIN U19 [get_ports {LED[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]
set_property PACKAGE_PIN V19 [get_ports {LED[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]
set_property PACKAGE_PIN W18 [get_ports {LED[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[4]}]
set_property PACKAGE_PIN U15 [get_ports {LED[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[5]}]
set_property PACKAGE_PIN U14 [get_ports {LED[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[6]}]
set_property PACKAGE_PIN V14 [get_ports {LED[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[7]}]
set_property PACKAGE_PIN V13 [get_ports {LED[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[8]}]
set_property PACKAGE_PIN V3 [get_ports {LED[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[9]}]
set_property PACKAGE_PIN W3 [get_ports {LED[10]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[10]}]
set_property PACKAGE_PIN U3 [get_ports {LED[11]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[11]}]
set_property PACKAGE_PIN P3 [get_ports {LED[12]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[12]}]
set_property PACKAGE_PIN N3 [get_ports {LED[13]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[13]}]
set_property PACKAGE_PIN P1 [get_ports {LED[14]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[14]}]
set_property PACKAGE_PIN L1 [get_ports {LED[15]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {LED[15]}]


##7 segment display
#set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
#set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
#set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
#set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
#set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
#set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
#set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

#set_property PACKAGE_PIN V7 [get_ports dp]
        #set_property IOSTANDARD LVCMOS33 [get_ports dp]

#set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
#set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
#set_property PACKAGE_PIN V4 [get_ports {an[2]}]
```

```
        #set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
#set_property PACKAGE_PIN W4 [get_ports {an[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


##Buttons
set_property PACKAGE_PIN U18 [get_ports reset]
        set_property IOSTANDARD LVCMOS33 [get_ports reset]
#set_property PACKAGE_PIN T18 [get_ports btnU]
        #set_property IOSTANDARD LVCMOS33 [get_ports btnU]
#set_property PACKAGE_PIN W19 [get_ports btnL]
        #set_property IOSTANDARD LVCMOS33 [get_ports btnL]
#set_property PACKAGE_PIN T17 [get_ports btnR]
        #set_property IOSTANDARD LVCMOS33 [get_ports btnR]
#set_property PACKAGE_PIN U17 [get_ports btnD]
        #set_property IOSTANDARD LVCMOS33 [get_ports btnD]



##Pmod Header JA
#Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {dacCLK}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dacCLK}]
#Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {dacCS}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dacCS}]
#Sch name = JA3
set_property PACKAGE_PIN J2 [get_ports {adcMISO}]
        set_property IOSTANDARD LVCMOS33 [get_ports {adcMISO}]
#Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {adcCS}]
        set_property IOSTANDARD LVCMOS33 [get_ports {adcCS}]
#Sch name = JA7
set_property PACKAGE_PIN H1 [get_ports {dacMOSI}]
        set_property IOSTANDARD LVCMOS33 [get_ports {dacMOSI}]
##Sch name = JA8
#set_property PACKAGE_PIN K2 [get_ports {adcMISO}]
#       set_property IOSTANDARD LVCMOS33 [get_ports {adcMISO}]
#Sch name = JA9
set_property PACKAGE_PIN H2 [get_ports {adcCLK}]
        set_property IOSTANDARD LVCMOS33 [get_ports {adcCLK}]
#Sch name = JA10
set_property PACKAGE_PIN G3 [get_ports {adcMOSI}]
        set_property IOSTANDARD LVCMOS33 [get_ports {adcMOSI}]



##Pmod Header JB
##Sch name = JB1
#set_property PACKAGE_PIN A14 [get_ports {JB[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JB[0]}]
##Sch name = JB2
#set_property PACKAGE_PIN A16 [get_ports {JB[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JB[1]}]
##Sch name = JB3
#set_property PACKAGE_PIN B15 [get_ports {JB[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JB[2]}]
##Sch name = JB4
#set_property PACKAGE_PIN B16 [get_ports {JB[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JB[3]}]
##Sch name = JB7
#set_property PACKAGE_PIN A15 [get_ports {JB[4]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JB[4]}]
##Sch name = JB8
#set_property PACKAGE_PIN A17 [get_ports {JB[5]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JB[5]}]
##Sch name = JB9
#set_property PACKAGE_PIN C15 [get_ports {JB[6]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JB[6]}]
##Sch name = JB10
#set_property PACKAGE_PIN C16 [get_ports {JB[7]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JB[7]}]



##Pmod Header JC
##Sch name = JC1
#set_property PACKAGE_PIN K17 [get_ports {JC[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JC[0]}]
##Sch name = JC2
#set_property PACKAGE_PIN M18 [get_ports {JC[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JC[1]}]
##Sch name = JC3
#set_property PACKAGE_PIN N17 [get_ports {JC[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JC[2]}]
##Sch name = JC4
#set_property PACKAGE_PIN P18 [get_ports {JC[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JC[3]}]
##Sch name = JC7
#set_property PACKAGE_PIN L17 [get_ports {JC[4]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JC[4]}]
##Sch name = JC8
#set_property PACKAGE_PIN M19 [get_ports {JC[5]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JC[5]}]
##Sch name = JC9
#set_property PACKAGE_PIN P17 [get_ports {JC[6]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JC[6]}]
##Sch name = JC10
#set_property PACKAGE_PIN R18 [get_ports {JC[7]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JC[7]}]


##Pmod Header JXADC
##Sch name = XA1_P
#set_property PACKAGE_PIN J3 [get_ports {JXADC[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[0]}]
##Sch name = XA2_P
```

```
#set_property PACKAGE_PIN L3 [get_ports {JXADC[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[1]}]
##Sch name = XA3_P
#set_property PACKAGE_PIN M2 [get_ports {JXADC[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[2]}]
##Sch name = XA4_P
#set_property PACKAGE_PIN N2 [get_ports {JXADC[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[3]}]
##Sch name = XA1_N
#set_property PACKAGE_PIN K3 [get_ports {JXADC[4]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[4]}]
##Sch name = XA2_N
#set_property PACKAGE_PIN M3 [get_ports {JXADC[5]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[5]}]
##Sch name = XA3_N
#set_property PACKAGE_PIN M1 [get_ports {JXADC[6]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[6]}]
##Sch name = XA4_N
#set_property PACKAGE_PIN N1 [get_ports {JXADC[7]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[7]}]



##VGA Connector
#set_property PACKAGE_PIN G19 [get_ports {vgaRed[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[0]}]
#set_property PACKAGE_PIN H19 [get_ports {vgaRed[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[1]}]
#set_property PACKAGE_PIN J19 [get_ports {vgaRed[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[2]}]
#set_property PACKAGE_PIN N19 [get_ports {vgaRed[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[3]}]
#set_property PACKAGE_PIN N18 [get_ports {vgaBlue[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[0]}]
#set_property PACKAGE_PIN L18 [get_ports {vgaBlue[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[1]}]
#set_property PACKAGE_PIN K18 [get_ports {vgaBlue[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[2]}]
#set_property PACKAGE_PIN J18 [get_ports {vgaBlue[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[3]}]
#set_property PACKAGE_PIN J17 [get_ports {vgaGreen[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[0]}]
#set_property PACKAGE_PIN H17 [get_ports {vgaGreen[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[1]}]
#set_property PACKAGE_PIN G17 [get_ports {vgaGreen[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[2]}]
#set_property PACKAGE_PIN D17 [get_ports {vgaGreen[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[3]}]
#set_property PACKAGE_PIN P19 [get_ports Hsync]
        #set_property IOSTANDARD LVCMOS33 [get_ports Hsync]
#set_property PACKAGE_PIN R19 [get_ports Vsync]
        #set_property IOSTANDARD LVCMOS33 [get_ports Vsync]


##USB-RS232 Interface
#set_property PACKAGE_PIN B18 [get_ports RsRx]
        #set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
#set_property PACKAGE_PIN A18 [get_ports RsTx]
        #set_property IOSTANDARD LVCMOS33 [get_ports RsTx]


##USB HID (PS/2)
#set_property PACKAGE_PIN C17 [get_ports PS2Clk]
        #set_property IOSTANDARD LVCMOS33 [get_ports PS2Clk]
        #set_property PULLUP true [get_ports PS2Clk]
#set_property PACKAGE_PIN B17 [get_ports PS2Data]
        #set_property IOSTANDARD LVCMOS33 [get_ports PS2Data]
        #set_property PULLUP true [get_ports PS2Data]


##Quad SPI Flash
##Note that CCLK_0 cannot be placed in 7 series devices. You can access it using the
##STARTUPE2 primitive.
#set_property PACKAGE_PIN D18 [get_ports {QspiDB[0]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[0]}]
#set_property PACKAGE_PIN D19 [get_ports {QspiDB[1]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[1]}]
#set_property PACKAGE_PIN G18 [get_ports {QspiDB[2]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[2]}]
#set_property PACKAGE_PIN F18 [get_ports {QspiDB[3]}]
        #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[3]}]
#set_property PACKAGE_PIN K19 [get_ports QspiCSn]
        #set_property IOSTANDARD LVCMOS33 [get_ports QspiCSn]
```

PCB Component Placement:

For soldering components to the PCB, the following items are recommended:

- Rosin flux pen or no-clean flux
- Soldering iron with tip equivalent to Weller CT6C7 tip
- 60/40 rosin core solder
- Isopropyl alcohol (91% or greater)
- Kimwipes
- Acid brush
- Kapton tape
- Precision anti-static ESD tweezers

1. Starting with the blank PCB, place the bottom side (without the board revision detailing) face down on a flat surface.
2. Solder all 0603 sized components using tweezers to position. Consult manufacturer datasheets and board schematics to ensure tantalum capacitors are soldered in the correct orientation for intended polarity.
3. Solder all 0805 sized components using the same method. Consult manufacturer datasheets and board schematics to ensure tantalum capacitors are soldered in the correct orientation for intended polarity.
4. Tin the voltage regulator pads and then solder the two LT1529 voltage regulators.
5. Solder the ADC, DAC, and op-amp array, using flux to ensure proper connection of the pins with their respective pads.
6. Solder the two mono 3.5mm jacks, making sure that the jack sits in the appropriate mounting holes.
7. Flip the PCB over, where the top side is facing up. Place the 2x6 header jack pins through, so the pins emerge on the bottom side. Tape the header with Kapton tape to ensure the jack sits flush with the PCB.
8. Flip the PCB over, where the bottom side is now facing up, and solder the jack.
9. Repeat the component placement methodology listed in steps 7 and 8 for the two BNC jacks, the DC barrel jack, two switches, 470 µF SMT capacitor, two scope probe grounding pins, and two scope probe loops. Consult manufacturer datasheets and board schematics to ensure the capacitor is soldered in the correct orientation for intended polarity.
10. Clean all soldered surfaces by placing a Kimwipe over the target area to be cleaned. Then use the acid brush to introduce isopropyl alcohol. This will dissolve the flux, and the Kimwipe will absorb the solution. Repeat with more Kimwipes and alcohol until board is fully cleaned. *Note: the ADC's TSSOP package is extremely sensitive to particular kinds of flux. It interferes with normal operation of the reference, so the ADC reference pins must be especially clear of flux to ensure proper functionality.*
11. Place the M3 hex standoffs on the bottom side of the PCB and secure using M3 screws from the top of the PCB.
12. Insert 2x6 interconnect pin header into jack for appropriate interfacing with the Basys 3 board.

Getting Started:

1. Ensure all power to the Basys 3 board and development board are off, especially disconnecting the USB programming cable from the Basys 3 to the computer if being used.
2. In the top right of the Basys 3 board, configure the JP1 header to be in USB mode. This configures the FPGA with a flash drive formatted in FAT32. This flash drive must only have a BIT file in the root directory for the programming to succeed.
3. Insert the flash drive with the BIT file into the USB type A port at the top right of the development board.
4. Connect the peripheral board to the Basys 3 development board via the JA header (PMOD header). The peripheral board's top side should be facing up.
5. In the top left of the Basys 3 board, configure the JP2 header to source power from the EXT pins instead of the USB pins.
6. Connect the J5 header on the peripheral board to the J6 header on the Basys 3 in order to power the Basys 3 from the peripheral board. *Make sure the USB programming cable is disconnected from the Basys 3. The 5 V USB supply will conflict with the peripheral board's voltage regulators if left connected.*
7. The non-energized platform should look like this:

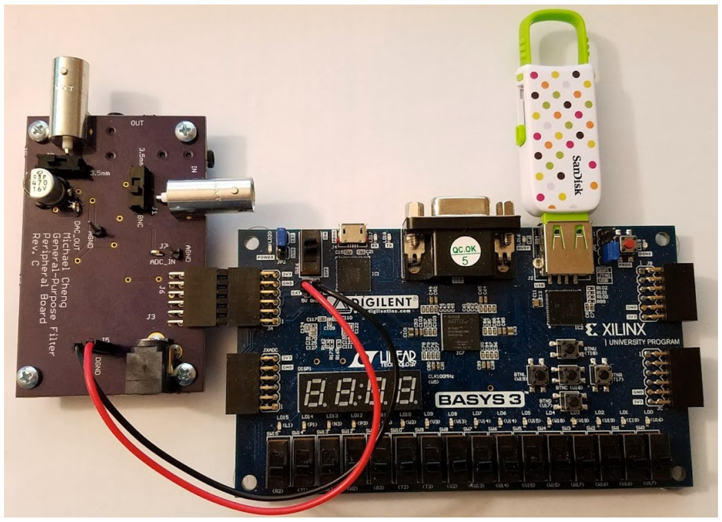


FIGURE XVI
UN-ENERGIZED PLATFORM SETUP

8. Energize the board through the barrel jack, J3. This should be power from the Triad Magnetics 6 V supply.
9. Connect the input source to the desired connector interface. Switch the SPDT switch to switch the ADC input to that source.
10. Connect the output source to the desired output connector interface. Switch the SPDT switch to switch the DAC output to that target.
11. Connect oscilloscope probes to the J4 and J7 probe test points to see the waveform that the ADC samples and DAC outputs.

Generating FIR Filter Logic in System Generator:
Two programs are essential for generating FIR filter logic in VHDL from Simulink:

- Xilinx Vivado HL System Edition
- MATLAB

There are version dependencies between these two products. For more information on which versions are compatible, see the following link:
https://www.xilinx.com/support/answers/55830.html

For the purposes of this project, Vivado Design Suite 2016.4 System Edition was used in conjunction with MATLAB r2016b. All testing was performed with Windows 7 Professional 64-bit.

Starting Notes:

- When installing the full version of Vivado, follow the instructions to link MATLAB and Vivado. If no instructions appear, open the start menu and select All Programs → Xilinx Design Tools → Vivado 2016.4 → System Generator → System Generator 2016.4 MATLAB Configurator. Click the checkbox next to the MATLAB icon and select Apply. If an error appears asking for administrator privileges, reopen the program with administrator privileges to configure this. This only needs to be done once.
- After configuration, open System Generator 2016.4. This should bring up MATLAB r2016b. System Generator automatically shifts the native directory to somewhere in the AppData folder. This can be easily changed by changing the workspace directory back to the default MATLAB folder in My Documents.

Customization Steps:

1. Move the Filter_Logic_Template.slx file inside the MATLAB workspace directory. Open this file after starting System Generator.
2. A block structure similar to Figure 8 should appear. In order to get the functionality detailed in Figure 7, you can right click on the Down Sample and Up Sample blocks to comment through them. This will effectively short them.
3. The system assumes that data will be provided at 1 MSPS to the Gateway In block. In order to modify the sample rate that the Digital FIR Filter block sees, the Up and Down Sample blocks need to be adjusted accordingly. Any block can be customized by double-clicking on it.
4. If the overall system sample rate must be adjusted, there are two places that need to be configured. Double-click on the Gateway In and adjust the sample period. Then, double-click the System Generator block, and under the clocking tab, adjust the FPGA clock period and Simulink System period.
5. If another FPGA chip is used, the Part under the Compilation tab of the System Generator block must be changed. The default setting is the Artix7 xc7a35t-1cpg236, which is the FPGA part on the Basys 3.
6. Any logic between the Gateway In and Gateway Out blocks will be turned into VHDL code. When ready to generate the code, double-click on the System Generator block and click the Generate button along the bottom of the window. Upon successful generation, System Generator will state “Generation Completed” in a pop window. Exit all pop-up windows and the filter logic is now ready for implementation in VHDL.

Integrating a System Generator Design into a Vivado Project:

1. Start Vivado 2016.4 and open the SeniorProjectFinal.xpr file.
2. Navigate to the Sources window in the top left corner and expand the drop down menu titled filter_system_wrapper – Behavioral.
3. Right-click the filter – FIR_block_wrapper – STRUCTURE file and remove it from the project.
4. Right-click the FIR_block and remove it from the project.
5. In the Flow Navigator, there should be an expanded menu titled IP Integrator. Click on Create Block Design and name the Design "FIR_block" exactly in order for the file to integrate properly with the existing design.
6. A new window should open titled Diagram, and on the left side of the window should be various icons. Find the IP Setting icon, which looks like a gear, and click on it. This should open the Project Settings window with the IP tab currently open.
7. Click on the Repository Manager tab under the IP tab. This is going to bring up which hard drive directory to search for your System Generator design.
8. Click the Add button, which is a green plus icon. The directory that you placed your Simulink file is where the VHDL code was generated. Navigate to this directory and you should see a netlist folder. In the netlist folder, there is an ip folder. Highlight this ip folder and click on Select to exit out of the IP Repositories window. Click OK to exit the Project Settings window.
9. In the middle of the Diagram window, there should be the Add IP icon. Click this icon. It will bring up many IP designs that you can choose from, but scroll down to filter_logic_template and double-click on it.
10. On each of the inputs and outputs of the block that appears, right-click on them. It will bring up a menu with options to copy, paste, search, and etc. Click on the Make External button (Ctrl + T).
11. Click on the validate design icon in the left toolbar of the Diagram window. Save after validation and exit the Block Design using the x button in the top right corner.
12. This will return you to your original Sources window. Right-click the FIR_block file in the Design Sources drop down menu, and click on Generate Output Products. Click on Generate at the bottom of the pop up window, leaving all settings as is.
13. After generation is completed, then right-click on the FIR_block file again, and click on Create HDL Wrapper. Let Vivado manage wrapper and auto-update, then click OK.
14. This should return the Design sources window to how it first appeared, but the new filter design has been implemented.
15. In the Flow Navigator, click on Bitstream settings. Make sure that under the Write Bitstream tab, the –raw_bitfile* option is checked. Click OK to exit the window.
16. Click on the Generate Bitstream button in the Flow Navigator.
17. When complete, navigate to the directory the Vivado project is stored in. In the SeniorProjectFinal folder, go to SeniorProjectFinal.runs→impl_1.
18. There should be a file named filter_system_wrapper.bit. This file needs to be copied to the flash drive used in the platform. It should be the only file on the flash drive.
19. After copying the file to the flash drive and returning it to the platform, the system is ready to use that particular filter design.