Xilinx FIR Coefficient Configuration Implemented in ROACH Architecture

Zoriel M. Salado Martínez
Master of Engineering in Electrical Engineering
Luis M. Vicente, Ph.D.
Electrical Engineering Department
Polytechnic University of Puerto Rico

Luis Quintero
Electronics Department
Arecibo Observatory

Abstract *— Dynamically run time reconfigurable FIR Filter with a control logic architecture for Coefficient reload is designed and tested on a Xilinx Virtex 5 FPGA ROACH board for signal recording at the Arecibo Observatory. A filter with fixed coefficients is used as to compare it with the reconfigurable filter. The resulting control logic design and the filter can be reconfigured with any coefficient and filter type limited only by its length (filter order) or word size.*

Key Terms *— CASPER (Collaboration for Astronomy Signal Processing and Electronics Research), Control Logic, Field Programmable Gate Array (FPGA), Finite Impulse Response (FIR) Filter, Reconfigurable Open Architecture Computing Hardware (ROACH).*

INTRODUCTION

FIR filters are digital filters with finite impulse response and one of the primary types of digital filters used in Digital Signal Processing (DSP). FIR filters can be used with fixed coefficients or with reconfigurable coefficients. Reconfigurable architectures have had a huge demand in the last decade due to its compatibility, reusability, and performance. On the other hand, static configurations can only be loaded before the system synthesis.

FPGAs are field programmable gate arrays that can be configured to perform any digital systems implementation. Coefficient configuration allows part of the FPGA to be reconfigured in run time while the rest of the design continues to work unaffected. Xilinx Virtex architectures with the aid of System Generator simulation interfaces allow shared memory blocks to transfer data between the PC and FPGA that let us reconfigure the filter's coefficients in the design and makes it useful for a vast amount of applications.

The FPGA used for this project is in a ROACH architecture board developed by the Collaboration for Astronomy Signal Processing and Electronics Research (CASPER).

The objective of this project is to design the control logic, which will allow coefficient reloading to the FPGA's FIR filter in run time. The design tools used are the MSSGE (MATLAB, Simulink Graphical Modeling Tool, Xilinx's System Generator Block set and Xilinx EDK and ISE tools). The reconfigurable FIR filter along with the control logic design will save time and reduce the amount of effort needed for radio astronomy instrumentation for coefficient changes as required. The Arecibo Observatory provided all the software and hardware tools used for this project.

Enlarged images are included at the end of this paper in the Appendix section for clarity.

PROJECT BACKGROUND

The Virtex 5 FPGA processing board will be used to record signals received by the Arecibo Observatory Antenna. The Arecibo Observatory currently holds a 2380MHz transmitter with a power up to 1MW. This signal is sent to objects within the Solar System, such as Asteroids, and the response signal gives information about their path, rotation, among others due to the Doppler effect.

The bandwidth caused by the object's rotation is not constant, but could reach up to 50Mhz. The current system at the Arecibo Observatory records up to 10MHz bandwidth, but with this FPGA implementation, we are trying to achieve 50MHz. The FIR Filter reconfiguration is needed to modify the filter based on the desired bandwidth. The system itself is called Digital Down Converter, which holds the FIR Reconfigurable Filter and the control logic design in this project.

CASPER

CASPER goal is to reduce the amount of effort done for radio astronomy instrumentation design through “an open-source, platform-independent design approach that enables astronomers to quickly and efficiently design new custom digital backend for existing and new instrumentation.” [1]

Software

With the aid of Simulink, CASPER tools developed complex DSP systems for radio astronomy instrumentation design using FPGAs. The advantages of CASPER DSP tools are that not only they are highly configurable, but also their extensive DSP library imported used in collaboration with Simulink.

These tools provide a platform for communication between scientists and engineers due to its high level of abstraction. From CASPER libraries a scientist can draw a high-level block diagram for their instruments and the engineer can add the control logic to go from a logic block design to a functional instrument.

CASPER MSSGE blocks used in this design include: slices, multiplexers, LFSR (Linear Feedback Shift Register part of the Xilinx library toolset in Simulink) and FIR filters. CASPER libraries provided software registers and BRAMs Block Random Access Memory) read and write accessible through the Python scripts.

At the Arecibo Observatory one of the computers has MSSGE and CASPER libraries; the other computer was used to load the bitstream file to the FPGA through a Python script and had direct access to the ROACH board through the PowerPC (PPC).

Hardware

The stand alone FPGA processing board used in this project is the ROACH board with a 100MHz internal clock. The FPGA is connected to several peripherals and input/output interfaces. The top sampling rate of one of the Analog to Digital Converter (ADC) boards in dual channel is 1.5GHz, but sampling is not done at more than 800MHz for this project. Noise sources were generated using a LFSR for a pseudo random input signal.

Communication from the computer to the board is done through the PPC 440 EPx subsystem, which is the primary command and control mechanism for the ROACH board. Figure 1 shows the ROACH board and Figure 2 shows the high level block diagram for the board.

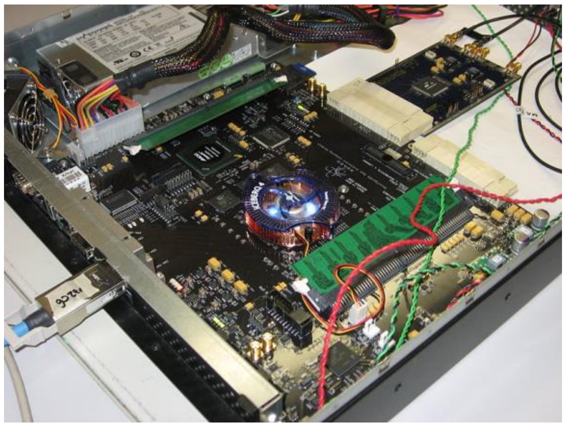


Figure 1
ROACH Board [1]

XILINX FIR FILTER

Xilinx FIR Compiler v5.0 allows to change the coefficients via control ports. The configuration used for the FIR filter is the Conventional single-rate with distributed arithmetic architecture. Only one set of coefficients, 16-bits wide, will be used and re-written each time the user wants to reload a new set of coefficients. The sampled signals are only 8-bits long and generated pseudo-randomly by a LFSR.

Coefficients for the FIR filters (reloadable and fixed) are generated through Matlab’s fir1(n,Wn) function, which returns a row vector containing the n+1 coefficients of an order n lowpass, highpass, bandpass or bandstop FIR filter. Wn is the normalized cutoff frequency for the hamming-window linear-phase. For the fixed coefficient architecture, the fir1 function is written directly into the Simulink Xilinx FIR Filter block before

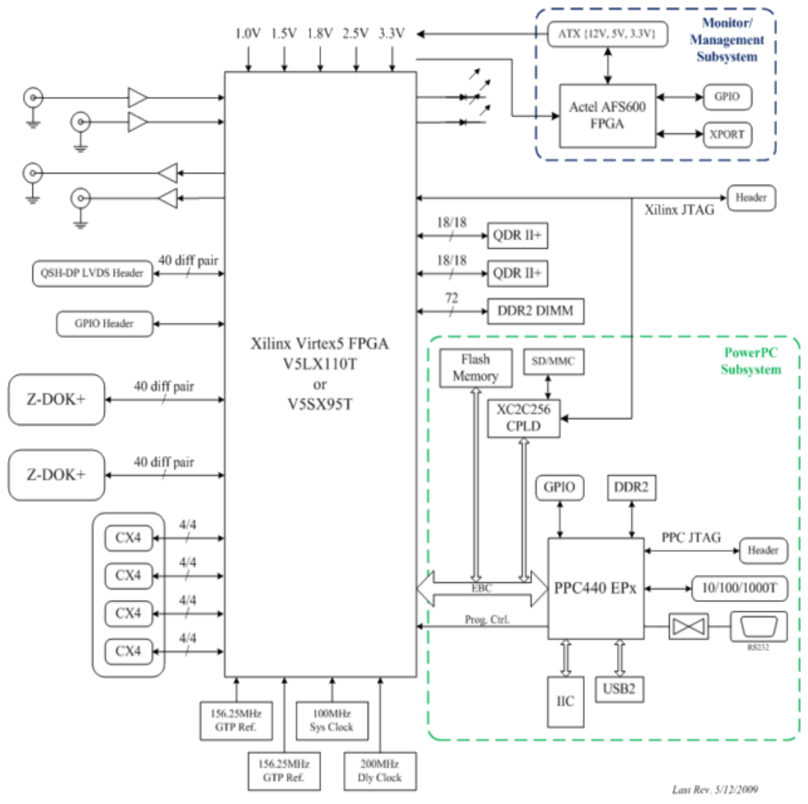


Figure 2
High-level Block Diagram of the ROACH Board [1]

synthesizing the design; for reloadable coefficient architecture, Matlab's code generates a binary file with the 16bit fixed point filter coefficients, which is read into the design while the FPGA is running with the synthesized design.

FIR with Fixed Coefficients

A FIR filter with fixed coefficients architecture was implemented to compare against the FIR filter with reloadable coefficients. With the ctrl software register shown in Figure 11, we coded a Python script to select between the input and the output signal of the filter, took a snapshot of the selected signal and stored it in a BRAM inside the snap block. There is no control logic added to the FIR filter with fixed coefficients, rather than a simple software register read/write in python.

Figure 11 shows the FIR filter architecture with fixed coefficients.

The portion of the Python code in Figure 3 is used as an example to show how the bitstream file (CONFIG_FILE variable in progdev routine) is loaded into the FPGA, the input/output signals snap selection and the plot command to show the signals and filter behavior for the FIR filter with fixed coefficients.

```python
HOST = 'rrb1'
CONFIG_FILE = 'fir_lfsr_snap_75.bof'

def read_snap(snap_sel):
  fpga.write_int('ctrl', snap_sel) # fir input
  fpga.write_int('snap_ctrl',1) # fill snap
  fpga.write_int('snap_ctrl',0) # clear flag
  x = fpga.read('snap_bram',8192)  # 2 * 2^11
  x = asarray(struct.unpack('>8192b', x)) # unpack & array
  x = x[3::4] # remove zeros
  return x

# initial setup
fpga = corr.katcp_wrapper.FpgaClient( HOST )
sleep(1)
print "Config status: %s" %( fpga.progdev( CONFIG_FILE ) )
print "Configuring FPGA (%s)" %(CONFIG_FILE)


fs  = 100           # Sampling freq. in MHz
X = fft(read_snap(0))
Y = fft(read_snap(2))
H = Y / X
N = len(X)

f = linspace(0, fs, N)
p1,=plot(f, log10(abs(X)), 'b')
p2,=plot(f, log10(abs(Y)), 'r')
p3,=plot(f, log10(abs(H)), 'k')
legend([p1, p3, p2], ["X(w)","H(w)","Y(w)"])
```

Figure 3
Python Code for Fixed Coefficients FIR Filter Design

Figure 4 shows the frequency response of the input X(ω), the output Y(ω) and the calculated filter response H(ω) = Y(ω)/X(ω). All the Fast Fourier Transform (FFT) plots in this paper are made from 0 to Fs (sampling frequency), and frequency responses are symmetric with respect to Fs/2=50MHz (Nyquist).

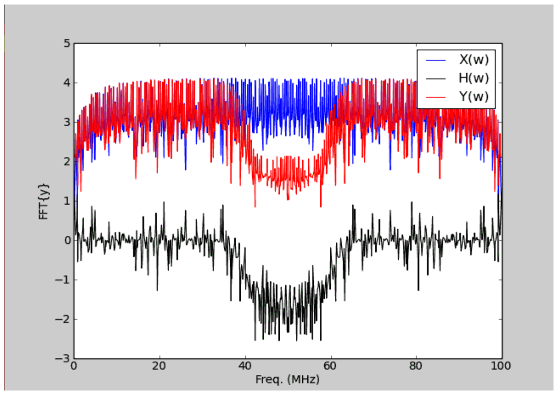


Figure 4
Fixed Coefficients Lowpass FIR filter - Spectrum. Sampling Frequency Fs=100MHz. The cutoff frequency of this filter is (Fs/2)*0.75=37.5MHz (0.75 normalized).

FIR with reloadable coefficients

Simulink yellow blocks (shared memory) in the design can be read/written directly from/to the FPGA through Python. These yellow blocks for the FIR with reloadable coefficients architecture include: two software registers (coef_load and coef_len) and a BRAM (coef_mem). Coef_len determines the amount of coefficients to be loaded into the coef_mem BRAM, all 32bit wide and coef_ld tells when to transfer the coefficients from coef_mem to the filter. Input and output signals of the filter are snapped with the snap block and plotted.

Figures 5, 6 and 7 show frequency responses for low-pass, high-pass and band-pass designs of the reconfigurable filter, respectively.

The portion of the Python code in Figure 8 is used as an example to show how the bitstream file (CONFIG_FILE) is loaded into the FPGA, Matlab's binary file (filter coefficients in coef_40_60.bin) read, the input/output signals snap selection and the plot command for the input, output, and calculated frequency responses.

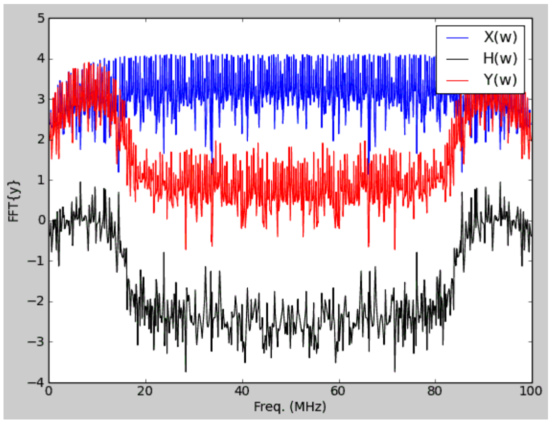


Figure 5
Reloadable Coefficients Lowpass FIR filter - Spectrum. Sampling Frequency Fs = 100MHz. The cutoff frequency of this filter is (Fs/2)*0.25 = 12.5MHz (0.25 normalized).

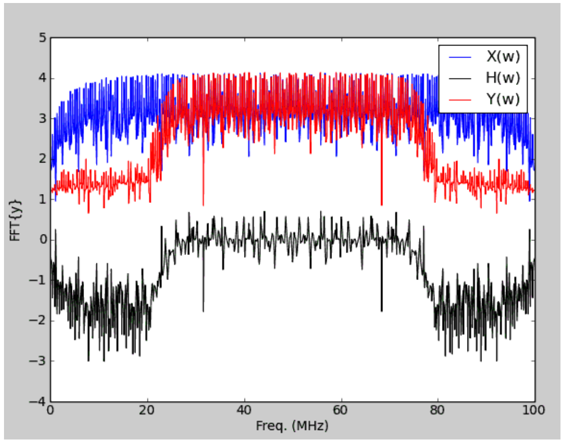


Figure 6
Reloadable Coefficients Highpass FIR filter - Spectrum. Sampling Frequency Fs = 100MHz. The cutoff frequency of this filter is (Fs/2)*0.50 = 25MHz (0.50 normalized).

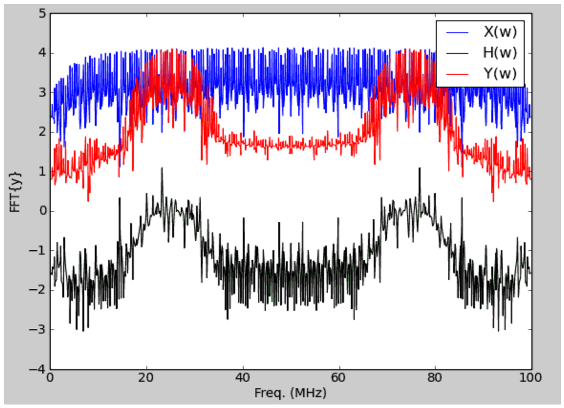
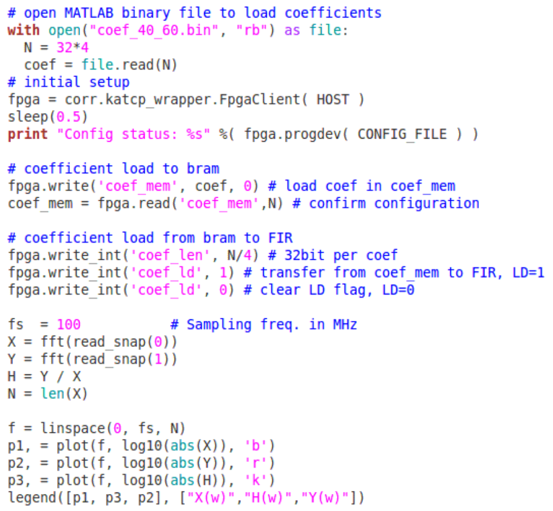


Figure 7
Reloadable Coefficients Bandpass FIR filter - Spectrum. Sampling Frequency Fs = 100MHz. The cutoff frequency of this filter is (Fs/2)*[0.40 0.60] = [20 30] MHz (0.40-0.60 normalized).

For this example in particular, the filter coefficients are for the band-pass FIR filter with normalized cut of frequencies between 0.40 and 0.60 (20MHz and 30MHz respectively).



```
# open MATLAB binary file to load coefficients
with open("coef_40_60.bin", "rb") as file:
  N = 32*4
  coef = file.read(N)
# initial setup
fpga = corr.katcp_wrapper.FpgaClient( HOST )
sleep(0.5)
print "Config status: %s" %( fpga.progdev( CONFIG_FILE ) )

# coefficient load to bram
fpga.write('coef_mem', coef, 0) # load coef in coef_mem
coef_mem = fpga.read('coef_mem',N) # confirm configuration

# coefficient load from bram to FIR
fpga.write_int('coef_len', N/4) # 32bit per coef
fpga.write_int('coef_ld', 1) # transfer from coef_mem to FIR, LD=1
fpga.write_int('coef_ld', 0) # clear LD flag, LD=0

fs  = 100           # Sampling freq. in MHz
X = fft(read_snap(0))
Y = fft(read_snap(1))
H = Y / X
N = len(X)

f = linspace(0, fs, N)
p1, = plot(f, log10(abs(X)), 'b')
p2, = plot(f, log10(abs(Y)), 'r')
p3, = plot(f, log10(abs(H)), 'k')
legend([p1, p3, p2], ["X(w)","H(w)","Y(w)"])
```

Figure 8
Python Code for Reloadable Coefficients FIR Filter Design

Control Logic design flow

The control logic design subsystem is fed with the coef_ld bit, which is passed to the coef_mem with a delay in order to be synchronized with the rest of the signals. Based on the coef_len, the coef_we will be high while coefficients are being transferred from the coef_mem to the filter.

Figure 13 shows the control logic block design and Figure 14 the timing diagram for the FIR filter coefficient load from the BRAM coef_mem.

Python code

The python code for both, the filter with fixed and reloadable coefficients starts with the library imports and the bitstream file load (synthesis file generated by Simulink).

For the filter with fixed coefficients, the coefficients are loaded when the FPGA, as well as the coefficient length (this information is included in the bitstream file loaded into the FPGA). In Python we do a fpga.write_int function call to write the ctrl software register and fill the snap, see Figure 11.

For the filter with reloadable coefficients, we open Matlab's binary file with the set of coefficients. First the coefficients are stored in the coef_mem BRAM starting at the first address location by using the fpga.write function call. The coefficient load from the BRAM to the filter is done by writing the coef_len and coef_ld software registers based on Xilinx's timing diagram in the datasheet [2]. The registers are 32-bit wide when they exit the BRAM, but are truncated to 16bits for the filter (the FIR filter could hold the whole 32-bits but requirements for this design specified a 16-bits wide coefficients). The coefficients are loaded one by one triggered by the coef_ld signal while the FPGA is turned on. Once all the coefficients have been loaded, the coef_ld bit is cleared. Finally, the snap of the filter's input and output is captured.

With the input x(n) and output y(n) stored in both snap BRAMs, we can take the FFT (Fast Fourier Transform) of the inputs and the outputs to go from time domain to the frequency domain (MHz). Now, we can divide the output Y(ω) by the input X(ω), and obtain the filter behavior as shown in Figure 9. Filter performances for the FIR with fixed coefficients and the FIR with reloadable coefficients are compared in the plot shown in Figure 10. Since the snap for both designs are not begin captured at the same time, we will expect some variations, but H(ω) for both filters should be almost the same.

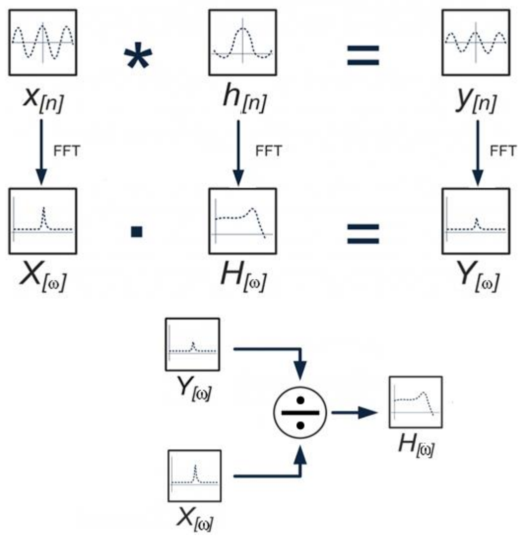


Figure 9
Time Domain to Frequency Domain Change and Filter Behavior Output Diagram [3]

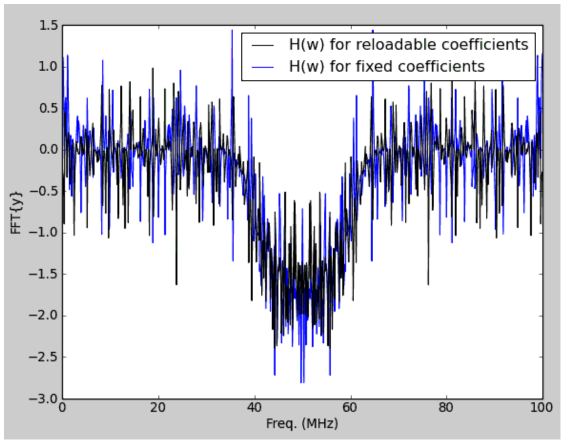


Figure 10
Filters with Fixed and Reloadable Coefficients Behavior

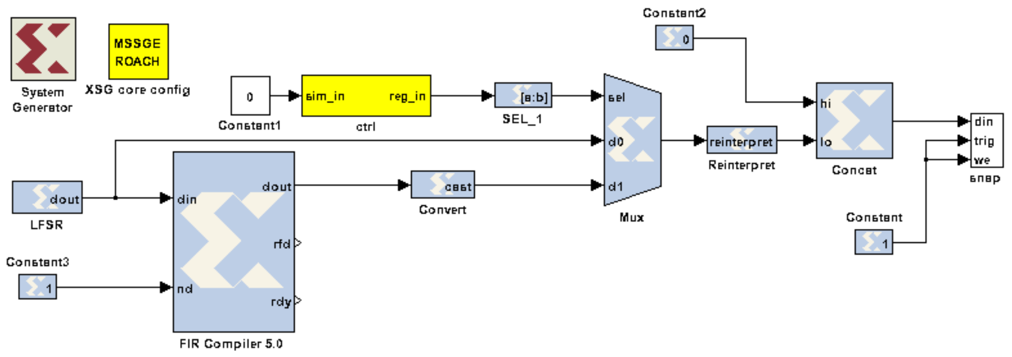


Figure 11
Fixed Coefficients FIR Filter Design

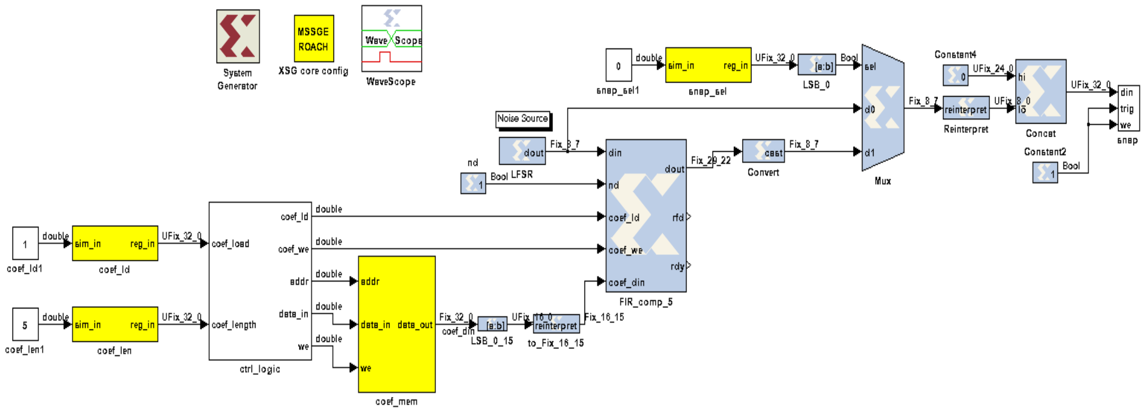


Figure 12
FIR Filter with Reloadable Coefficients, Memory Transfer and Control Logic Subsystem

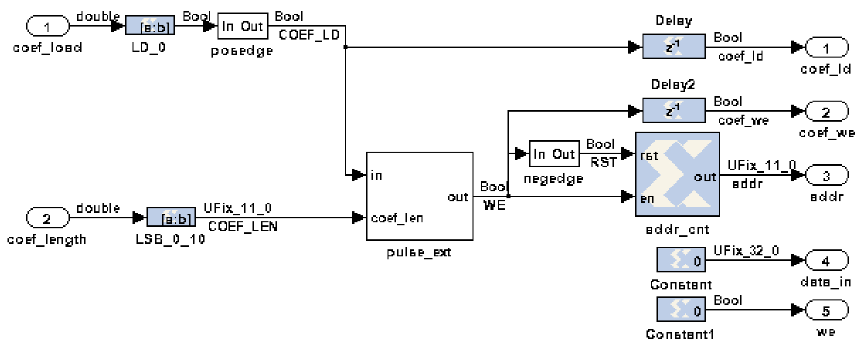


Figure 13
Control Logic Subsystem Design

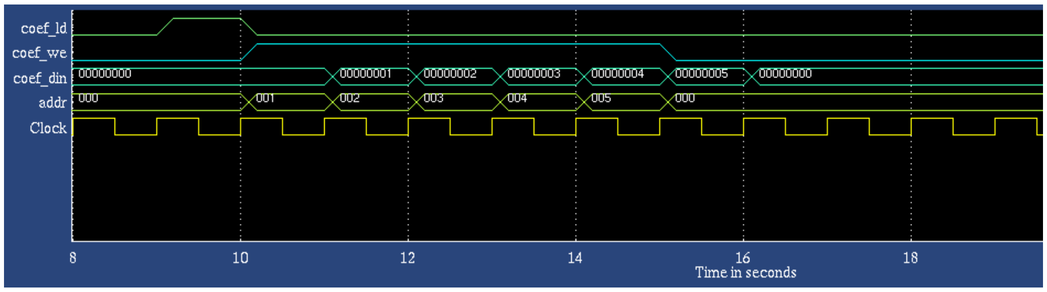


Figure 14
Coefficient Reload Timing for Control Logic Design

Conclusion and Future Work

Two FIR filter designs for a Virtex 5 FPGA were analyzed. Coefficients for both FIR filters were generated using Matlab's fir1 function. The first filter had its coefficients defined by the time the synthesis to the FPGA was loaded; on the other hand, the reloadable FIR filter loaded its coefficients while the FPGA was on using an imported Matlab's bit file. Having the flexibility of loading the coefficients without reloading the entire FPGA saves time and allows the scientists to use a wide range of filter designs for different bandwidths with the same bitstream configuration file.

Having this said, the scientist will not have to add the coefficients and synthesize the design each time they want to filter different input signals.

Both filter behaviors were tested and it is shown that they have almost the same behavior, which proves that the coefficient reloading as well as the timing between the BRAM data transfer and the filter, worked successfully. The results show that the design is capable of working with any filter type: bandpass, lowpass and highpass; also for any cutoff frequency value.

As a future work, we can compare resource utilization in the FPGA for different parameters such as filter order and coefficient width; study the effects of signal output truncation of the filter, from full precision to a lower bit width; improve the LFSR-pseudo random noise generated internally at the FPGA in order to get cleaner filter response; test the filter with real signals coming from ADCs and external noise sources; and synthesize the design at higher FPGA clock rates, i.e. 200MHz, and see if it meets timing constraints.

Acknowledgement

I would like to thank the Arecibo Observatory staff, in particular Sixto Gonzalez who was the first contact person and initial link to the Electronics Department. I would also like to thank Melissa Rivera for contacting the Arecibo Observatory for this research project, Osvaldo Mangual for the brainstorming ideas, Sharissa Kamer for understanding each time I had to work outside regular working hours, my partner Michelle Flores for all her support throughout my Masters Degree and to Professor Luis Vicente for supporting this project.

REFERENCES

[1] Berkley University, “CASPER”, CASPER Group – Collaboration for Astronomy Signal Processing and Electronics Research, [Online], April 2013. Retrieved from: https://casper.berkeley.edu.

[2] Xilinx, “IP LogiCORE FIR Compiler v5.0”, Xilinx, Inc. [Online], March 2011. Retrieved from: http://www.xilinx.com/support/documentation/ip_documentation/fir_compiler_ds534.pdf.

[3] Hamilton Kibbe, “Finite Impulse Response Filters Using Apple’s Accelerate Framework – Part III”, HAMILTON KIBBE, [Online], April 2014. Retrieved from: http://hamiltonkib.be/finite-impulse-response-filters-using-apples-accelerate-framework-part-iii/.