Lab 6: FPGA Parallel Processing Techniques – FIR Filter Design

EEL 4712 – Fall 2014

Objective:

The objective of this lab is to study parallel processing techniques for the design and implementation of a finite impulse filter (FIR) on an FPGA. An algorithmic state machine is used to control a data path component to exploit parallelism through functional decomposition, replication, and pipelining.

Required tools:

DE0 board, Quartus II Design Software, altfp_mult megafunction (floating-point multiply), altfp_add_sub megafunction (floating-point add), altsyncram megafunction, and Quartus In-System Memory Content Editor

Discussion:

Filters are signal conditioners by accepting an input signal, blocking some specified frequency components, and passing the input signal minus those components to the output. Finite impulse response (FIR) filters are one of the most common types of digital filters used in Digital Signal Processing (DSP) applications. A brief introduction to FIR filters can be found in the article "Introduction to Digital Filters".

The basic structure of a FIR filter consists of a series of multiplications followed by an addition, as represented by the following equation:

$$y(k) = \sum_{n=0}^{N-1} a(n)x(k-n) \qquad k = 0, 1, \ldots$$

In this equation, x(k) represents the sequence of input samples, y(k) is the output, a(n) represents the filter coefficients, and N is the number of "taps". Note that an FIR filter simply produces a weighted average of its N most recent input samples. All the "magic" is in the coefficients (a(n)), which dictate the actual output for a given pattern of input samples. To design a filter means to select the coefficients (or generate the coefficients using a software package like MATLAB) such that the filter has specific characteristics (e.g., low-pass, band-stop, or high-pass).

An outline of an implementation of an N-tap FIR filter, written in C and based on the above equation, is given in the article "Introduction to Digital Filters". Note that it is basically a ***FOR loop***.

A hardware design of an 8-tap filter based on the equation is illustrated in Figure 1. Each register (Ri) provides a unit sample delay (i.e., shift registers). Each output stage of a particular register is multiplied by a known coefficient. The resulting outputs of the multipliers are then summed to create the filter output. Note that the FOR loop of "N" iterations in the C program has been "unrolled" into "N" identical (register-multiplier) modules, exploiting parallelism through functional decomposition/replication.

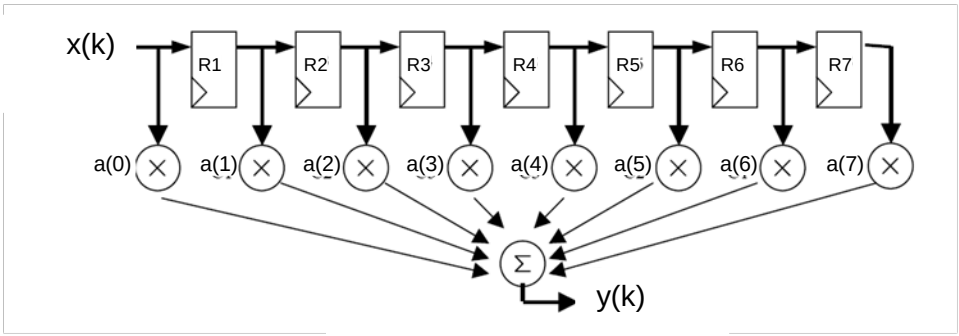

Figure 1. An 8-tap FIR filter

A further refinement of the design is shown in Figure 2 in block diagram form, in the form of a 4-tap FIR filter. Note that this design allows the inputs to be pipelined so that a new output is available for every cycle of the input (after some initial latency).

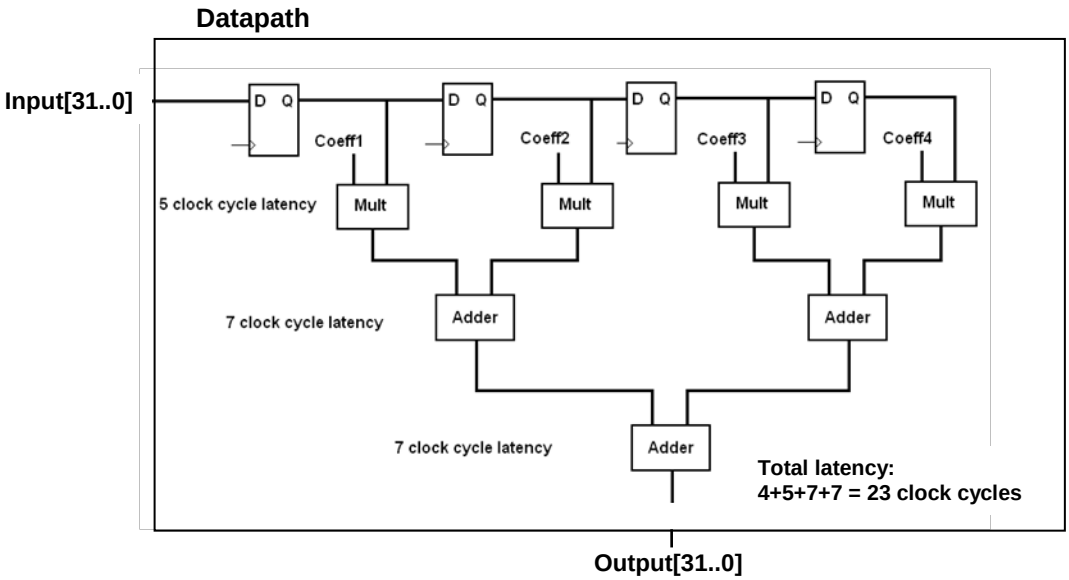

Figure 2. Block diagram of a 4-tap FIR filter with pipelining

Pre-lab requirements:

Part 1:

Using structural VHDL (PORT MAP), design and implement a component named "Datapath" by completing the detailed design of an 8-tap FIR filter (based on the design of the 4-tap filter Figure 2).

1. All signal paths are 32 bits.
2. For this lab, the following filter coefficients for a 8-tap filter (in decimal) are used:

   [0.068556404040904, 0.111805808555109, 0.149037722285938, 0.170581276322676, 0.170581276322676, 0.149037722285938, 0.111805808555109, 0.068556404040904]

   - You need to convert them into the IEEE single-precision floating point (32 bit) format (e.g., use converter at: http://babbage.cs.qc.edu/IEEE-754/)

     sign bit exponent fractional
     [31] [30..23] [22..0]

3. Generate a (32-bit floating-point multiplier) "Mult" component in VHDL using the altfp_mult megaWizard (in the "Arithmetic" folder) with the default settings except:
   - Make sure that the device family is Cyclone III.
   - Select Single Precision (32 bits). Select minimum latency (Should be 5). Check 'Use dedicated multiplier circuitry'.
   - Page 6: Check only the '.cmp' and the '_inst.vhd' options.
4. Generate the (32-bit floating-point) "Adder" component in VHDL using the altfp_add_sub megaWizard (in the "Arithmetic" folder) with the default settings except:
   - Make sure that the device family is Cyclone III.
   - Select Single Precision (32 bits). Select minimum latency (Should be 7). Select Addition only.
   - Page 6: Check only the '.cmp' and the '_inst.vhd' options.
5. Create the rest of the components and PORT MAP them to produce the "Datapath" for the 8-tap filter.
6. To simulate the Datapath component, create a top-level entity and a testbench. Use an altsynram megafunction to implement a ROM component. The Input[31..0] for the Datapath component will be outputted from the ROM component, the contents of which is provided to you in FIR_INPUT.mif. They

represent the sequence of input samples for the FIR filter. Each value is in the IEEE standard 32-bit floating point format.

Submit to e-Learning and printout for TA:

1. All the design and testbench files (.vhd files).
2. Simulation of the Datapath component with at least the following outputs: CLK, ROMaddr[9..0], Input[31..0], Output[31..0].

Part 2:

Using the Datapath and Input ROM components from Part 1, design and implement a complete 8-tap FIR filter as shown in Figure 3.

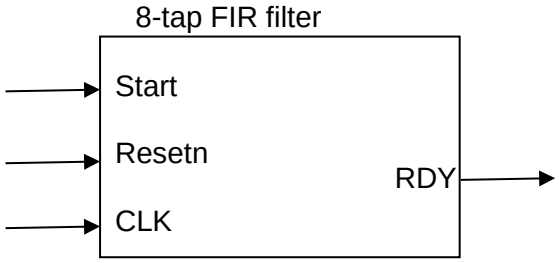

Figure 3. Complete FIR filter

Resetn: Active low reset signal that initializes the FIR filter to the initial state.

Start (active high): The FIR filter wait for this signal to be true before it begins its operation.

CLK: Clock input for the FIR filter.

RDY: Ready (active high), indicates that the FIR filter is finished and is ready for a new operation.

A block diagram design of the FIR filter is shown in Figure 4.

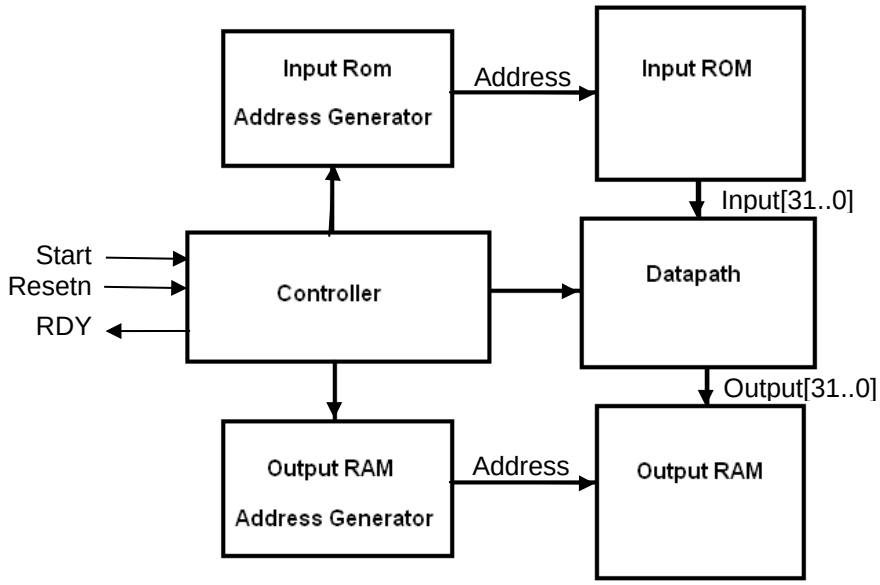

Figure 4. Block diagram design of FIR filter

1. The Input ROM Address Generator and the Output RAM Address Generator are both simple counters.
2. The Output RAM component is another altsynram megafunction configured as a single-port RAM. Note that the sequence of outputs from the Datapath component is stored in consecutive locations in the Output RAM. During the in-lab portion, you will use the Quartus In-System Memory Content Editor to observe the contents of the Output RAM to verify the correctness of the outputs.
   - Note that the Output RAM has to be configured to do so in the Megafunction Wizard in order to use the In-System Memory Content Editor. (Please see the In-System Memory Content Editor tutorial.)

3. The controller will need three phases:
   - Wait for the Start signal.
   - Process enough input samples to satisfy the initial latency period required by the pipeline (registers, multipliers, and adders) before writing the first valid output to the Output RAM. Hint: how many clock cycles will it take to fill up the pipeline?
   - Process the remaining input data and write to the Output RAM (after the initial latency. Note that the Output RAM should be of the same size as the input ROM.
4. The simulation of the FIR filter should include the following signals
   - Clock, Resetn, Start, RDY
   - Controller states.
   - Datapath register values
   - ROM control signals, Address and Output bus
   - RAM control signals, Address and Data/Output bus

Submit to e-Learning and printout for TA:

- ASM chart for your controller.
- All the design and testbench files (.vhd files).
- Simulation outputs for the complete FIR filter as shown in the sample outputs. Print only the key parts of the simulation (e.g., beginning, some of the middle, end)

In-lab Tasks:

1. Compile the complete circuit for the FIR filter from Pre-lab Part 2 and download it to the DE-0 board.
2. Using the In-System Memory Content Editor, observe the contents of the Output RAM to verify the correctness of the outputs. Debug if necessary. Wait until the RDY LED lights up to scan the RAM contents.
3. When you are satisfied with the outputs, In the In-System Memory Content Editor, go to Edit -> Export data to file. Save to a '.mif' file. Submit the '.mif' file online.
4. Perform an in-lab task assigned by your TA.

Appendix A:

While conducting an experiment, WallE Gator expected the input signal to be a sine wave of frequency 1KHz sampled at 25KHz. However, the observed input was a square wave with ringing (shown in blue in Figure A-2). WallE mathematically interpolated the input to the square wave (shown in green in Figure A-2). Realizing that the signal was corrupted by higher order harmonics, Wally designed a FIR filter to clean up the signal. The magnitude and impulse response of the filter are shown in Figure A-1. The expected output is a near-sine wave (shown in red in Figure A-2) delayed by the group delay of the filter which is 3.5 samples. The FFT of the input (blue) and the filtered output (red) are shown in Figure A-3.

Filter coefficients:
For this lab, the following filter coefficients for a 8-tap filter (in decimal) are used:

[0.068556404040904, 0.111805808555109, 0.149037722285938, 0.170581276322676, 0.170581276322676, 0.149037722285938, 0.111805808555109, 0.068556404040904]

- You need to convert them into the IEEE single-precision floating point (32 bit) format (e.g., use converter at: http://babbage.cs.qc.edu/IEEE-754/)

sign bit exponent fractional
[31] [30..23] [22..0]

Decimal value = $(-1)^s * 2^{e-127} * 1 + f$

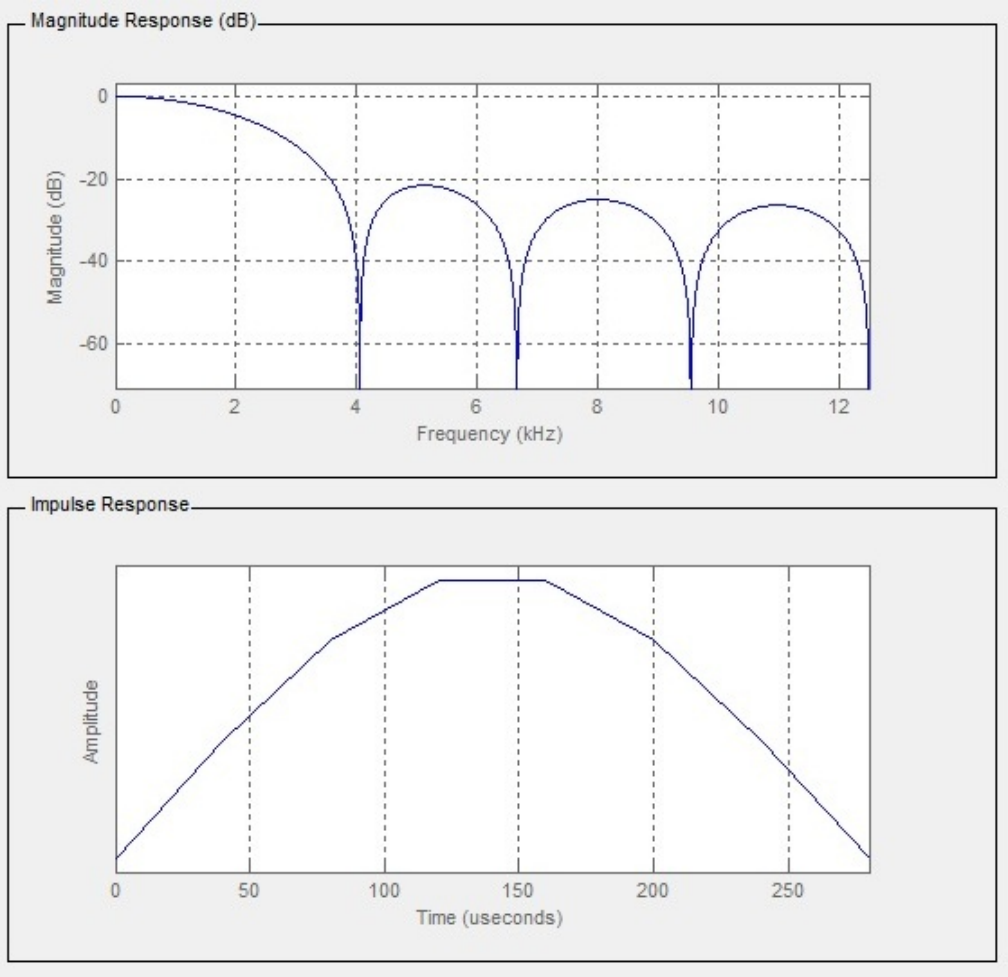

Figure A-1. Corresponding Filter Magnitude and Impulse Response

For this lab, the input vector (already converted to single-precision floating point) is given in FIR_INPUT.mif.

To verify the correctness of your output, the first 4 elements of input vector = [0x00000000, 0x3F96FE83, 0x3F66C78A, 0x3F88BC21 …]
which is equivalent to decimal [0, 1.179642127146860, 0.901482240402704, 1.068241253763636 …]

The first 16 output from MATLAB (convolution of the filter coefficients with the input vector) is: 0.0; 0.080872022292352; 0.193693222540071; 0.349836905534984; 0.519945872966512; 0.691668332609275; 0.835610111214736; 0.944075796021697; 1.005825233266888; 0.997671734634238; 1.000970868755058; .000128130759343; 0.993578584887082; 0.859460044230899; 0.636072206561142; 0.339308741242015.

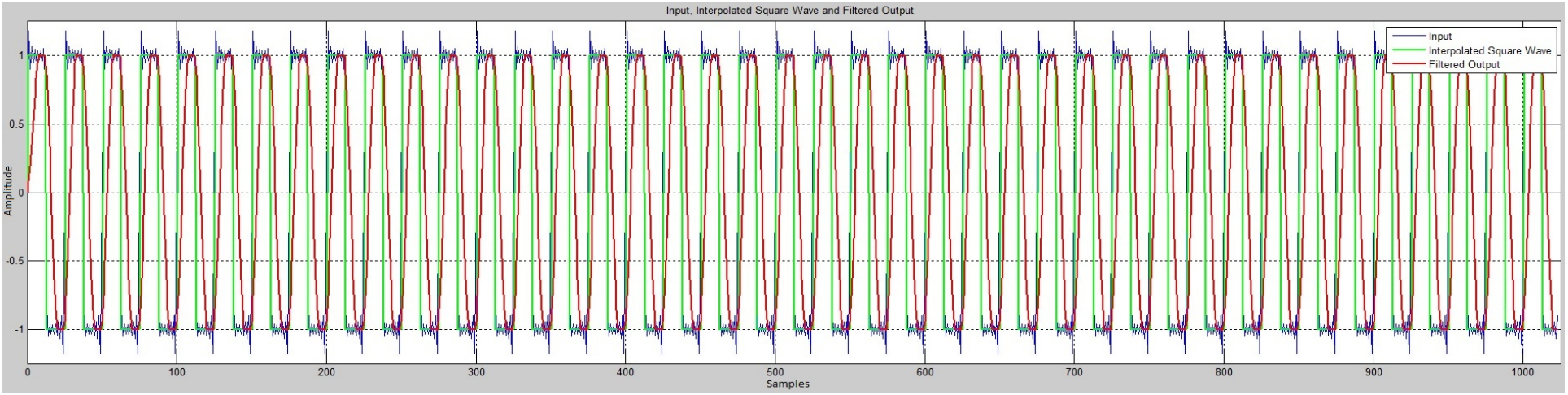

Figure A-2. Input, Interpolated Square Wave and Filtered Output

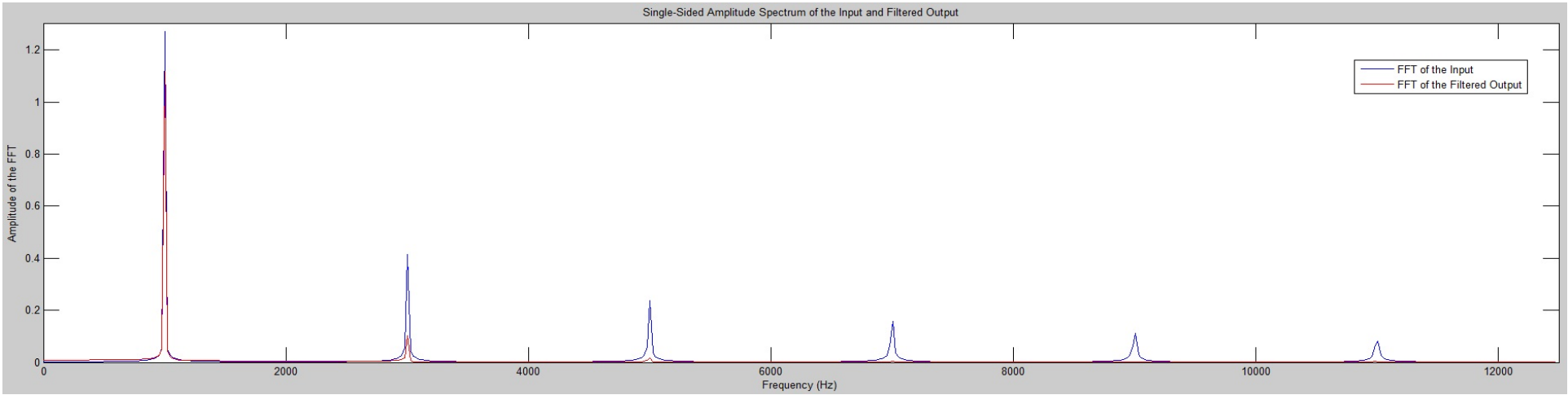

Figure A-3. FFT of the Input and Filtered Output