Lab 6: FPGA Parallel Processing Techniques – FIR Filter Design

EEL 4712 – Fall 2023

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:

Intel Quartus Prime, ModelSim-Altera, Terasic DE10-Lite, Quartus IP Catalog to create memory and floating-point components, 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.

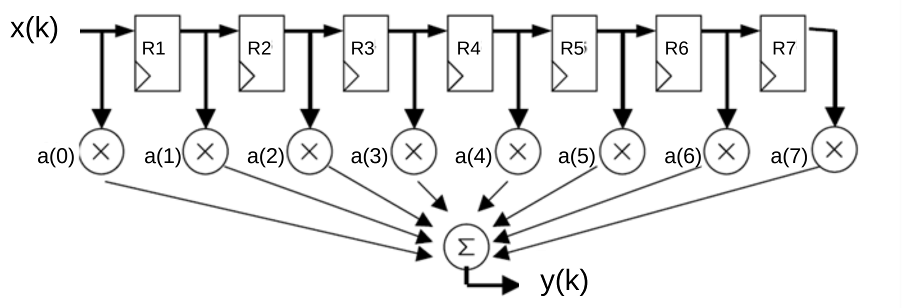


Figure 1. An 8-tap FIR filter

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).

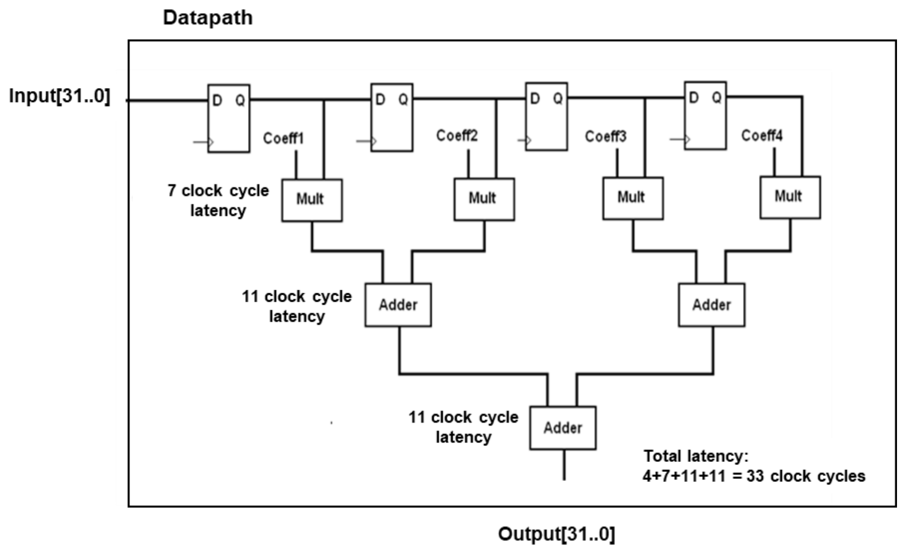


Figure 2. Block diagram of a 4-tap FIR filter with pipelining

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.cuny.edu/IEEE-754/)

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

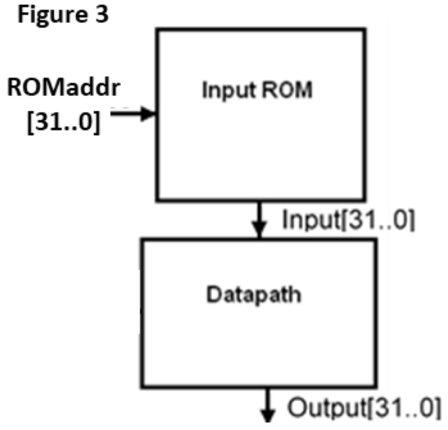
3. Create a Quartus project for Lab 6.
4. Generate a (32-bit floating-point multiplier) "MultFP32" component in VHDL using the **FP_FUNCTIONS Intel FPGA IP** in the Quatus IP Catalog: Library -> Basic Functions -> Arithmetic -> FP_FUNCTIONS Intel FPGA IP (double-click).
   - IP variation file name: .../MultFP32.
   - Select **VHDL** (instead of Verilog).
   - Click OK.

   A FP_FUNCTIONS Intel FPGA IP window pops up.
   - In the Function/Name drop-down box, select **Multiply**.
   - Click on the Performance tab and see the resource estimates and verify that the latency is 7 cycles.
   - Click Finish; Click Exit; then **Click "Yes" to add the IP file to the project.**
   - After which you can use the MultFP32 component as usual (i.e., PORT MAP).

5. Generate the (32-bit floating-point adder) "AddFP32" component in VHDL using the **FP_FUNCTIONS Intel FPGA IP** in the Quatus IP Catalog: Library -> Basic Functions -> Arithmetic -> FP_FUNCTIONS Intel FPGA IP (double-click).
   - IP variation file name: .../AddFP32.
   - Select **VHDL** (instead of Verilog).
   - Click OK.

   A FP_FUNCTIONS Intel FPGA IP window pops up.
   - In the Function/Name drop-down box, select **Add**.
   - Click on the Performance tab and see the resource estimates and verify that the latency is 11 cycles.
   - Click Finish; Click Exit; then **Click "Yes" to add the IP file to the project.**
6. Create the rest of the components and PORT MAP them to produce the "Datapath" for the 8-tap filter.
7. To simulate the Datapath component, create a top-level entity and a testbench.
   Create the top-level entity (Figure 3) – Use the **ROM: 1-PORT** function in the Quatus IP Catalog: Library -> Basic Functions -> On Chip Memory -> ROM: 1-PORT (double-click) to create the Input ROM.
   - IP variation file name: .../ROM.
   - Select **VHDL** (instead of Verilog).
   - Click OK.

   Figure 3

   A MegaWizard Plug-In Manager **[page 1 of 5]** pops up:
   - How wide should the 'q' output bus be: **32 bits**.
   - How many 8-bit words of memory: **1024**.
   - Leave everything else as default and click Next.

   [page 2 of 5]
   - Which ports should be registered? **Uncheck** the 'q' output port box.
   - Check the Create a **'rden'** read enable signal.
   - Leave everything else as default and click Next.

   [page 3 of 5]
   - In the window for "Yes, use this file for the memory content data", browse to provided **FIR_INPUT.mif** file and load it.
   - Note tht the type file should be MIF file, not HEX file.
   - Leave everything else as default and click Next.

   [page 4 of 5] is a summary page: Click Next.

   [page 5 of 5] is a summary page: Click Finish.
   - After clicking Finish and Exit, **click "Yes" to add the IP file to the project.**

   Note that he 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.
   - **Create a testbench** which simulates the top-level entity (ROM + Datapath), displaying at least the following outputs: CLK, ROMaddr[9..0], Input[31..0], Output[31..0].

Note: To avoid getting the following error in your Quartus compilation: "Error (16031): Current Internal Configuration mode does not support memory initialization or ROM. Select Internal Configuration mode with ERAM", you need to do the following before compilation:

> **Assignments -> Device -> Device and Pin Options -> Configuration ->** Configuration Mode: Single uncompressed image with Memory Initialization.

Submit to e-Learning:

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

Bring to lab: your laptop with Quartus and ModelSim projects containing all the VHDL files and all simulation results

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 4.

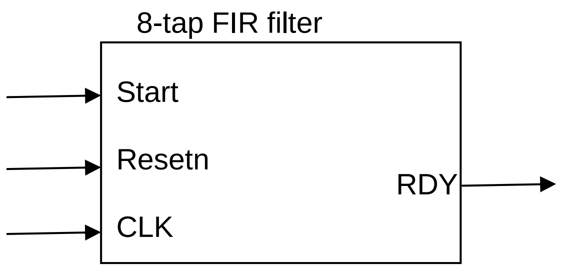


Figure 4. Complete FIR filter

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 5.

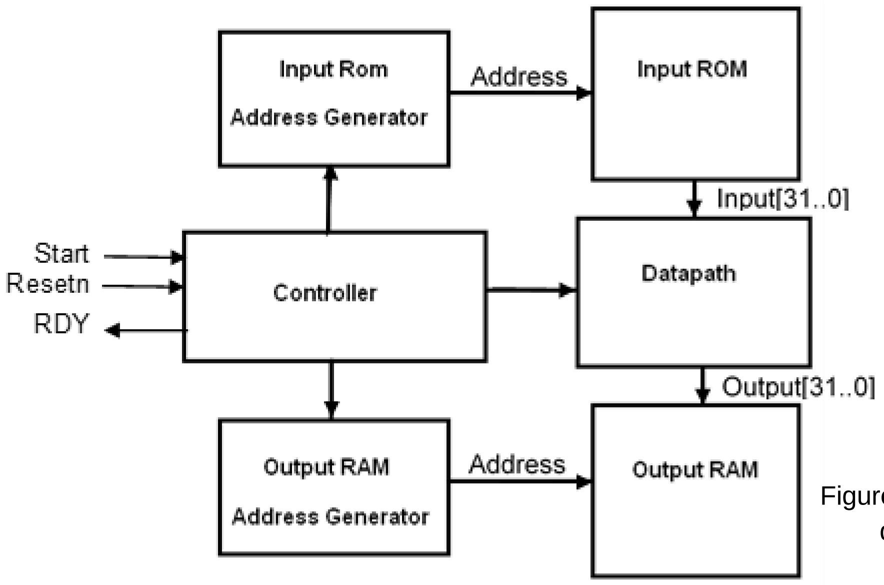


Figure 5. Block diagram design of FIR filter

1. The **Input ROM Address Generator** and the **Output RAM Address Generator** are both simple counters.
2. **Output RAM** component: Use the **RAM: 1-PORT** function in the Quatus IP Catalog: Library -> Basic Functions -> On Chip Memory -> RAM: 1-PORT (double-click).

- IP variation file name: .../RAM.
- Select **VHDL** (instead of Verilog).
- Click OK.

A MegaWizard Plug-In Manager **[page 1 of 6]** pops up:

- How wide should the 'q' output bus be: **32 bits**.
- How many 8-bit words of memory: **1024**.
- Leave everything else as default and click Next.

[page 2 of 6]

- Which ports should be registered? **Uncheck** the 'q' output port box.
- Leave everything else as default and click Next.

[page 3 of 6]

- Leave everything as default and click Next.

[page 4 of 6]

- Check the box which "**allow In-System Memory Content Editor** to capture and update content independently of the system clock".
- Leave everything as default and click Next.

[page 5 of 6] is a summary page: Click Next.

[page 6 of 6] is a summary page: Click Finish.

- After clicking Finish and Exit, **click "Yes" to add the IP file to the project.**

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.

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:

- 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.

Bring to lab: your laptop with Quartus and ModelSim projects containing all the VHDL files and all simulation results.

In-lab Tasks:

1. Compile the complete circuit for the FIR filter from Pre-lab Part 2 and download it to the DE-10 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.