GEORGE MASON UNIVERSITY

Serial FIR Filter

A Brief Study in DSP

ECE448
Spring 2011
Tuesday Section
15 points

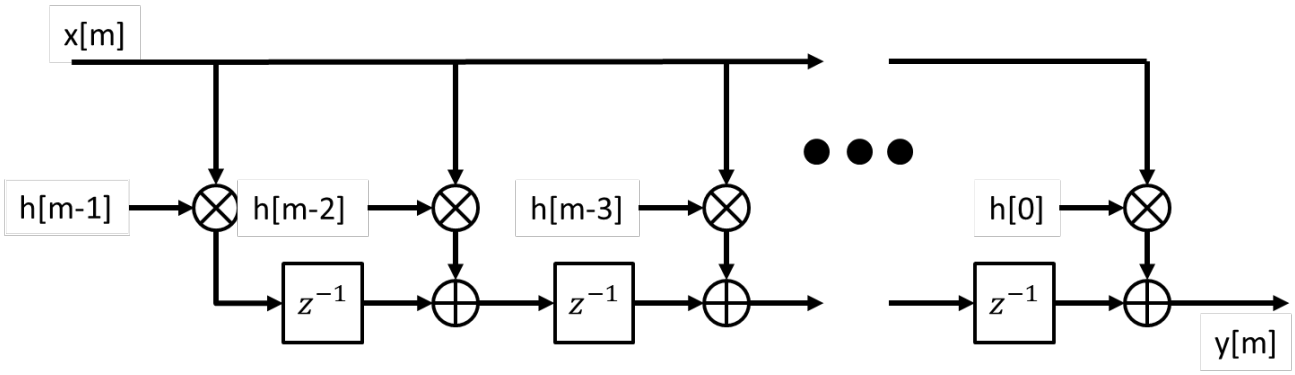
3/8/2011

Instructions:

Zip all your deliverables into an archive <last_name>.zip and submit it through Blackboard no later than Tuesday, March 8, 10:15 PM EST.

Introduction

A Finite Impulse Response (FIR) filter is a basic building block for Digital Signal Processing (DSP). The purpose of a FIR filter is to process an incoming signal to remove or enhance spectral components. A typical FIR filter is a low-pass filter. It gets this name because it passes the low frequencies and filters out the higher frequencies. A FIR filter consists mainly of a multiply-accumulate structure, as can be seen in the following diagram:

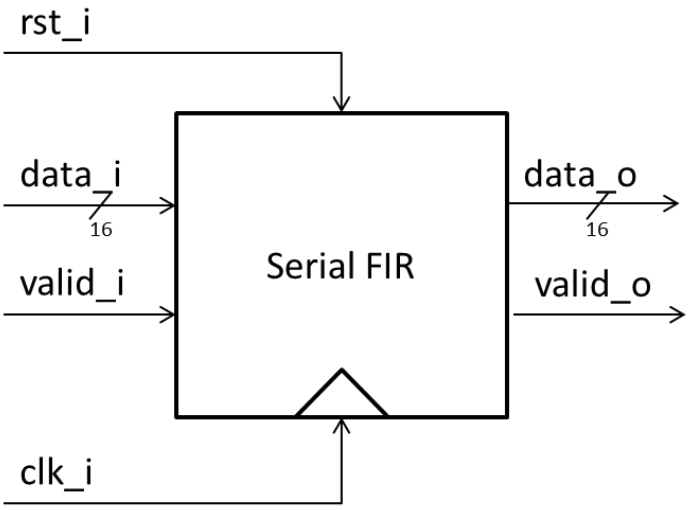


This is a typical FIR filter block diagram. This is a parallel structure, and requires many multipliers, adders and registers. If time requirement is sufficiently low, we can perform a serial version and save resources.

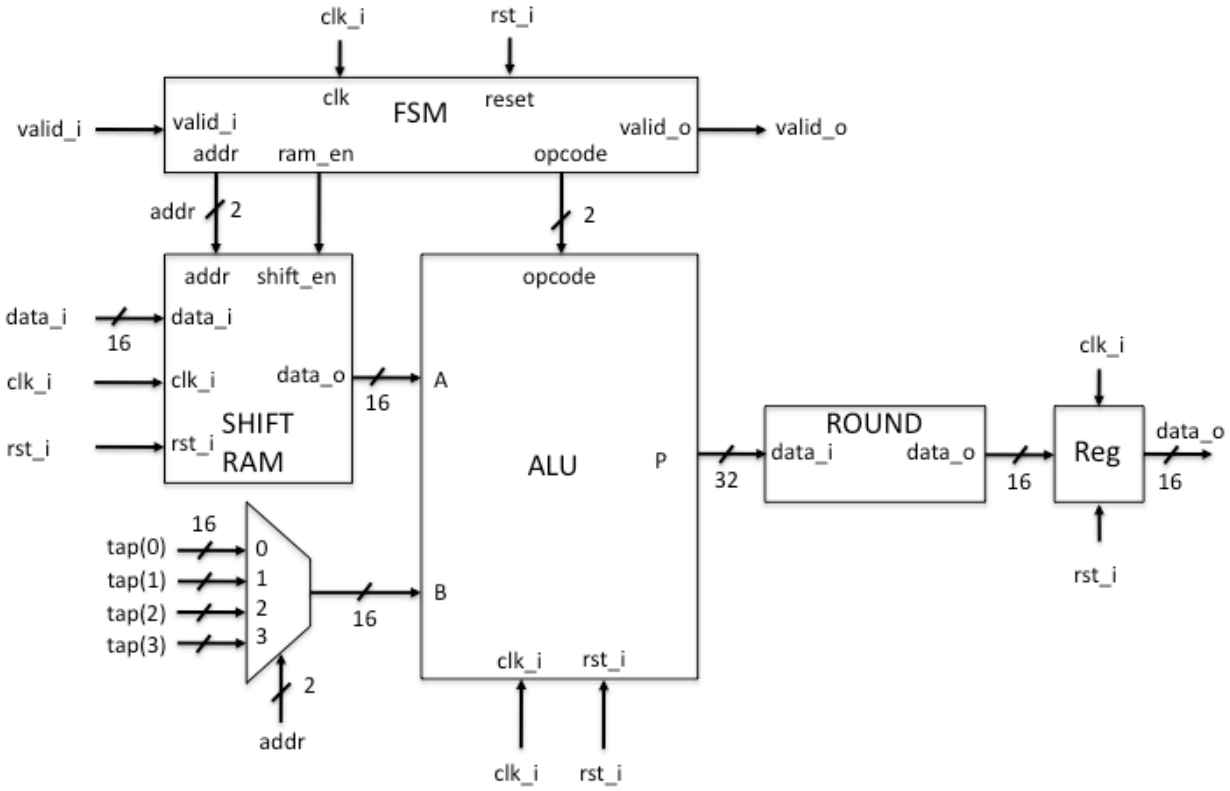
Serial FIR Filter

A Serial FIR filter gives the same output as a typical FIR filter, but not as quickly. It shares the multipliers and adders to save on the resources, but does so at the cost of time. In this example, we will use only one multiplier/adder and signify our completion by setting the valid_out signal high. Further description can be seen below.

Interface



Block Diagram of the Top-level Unit



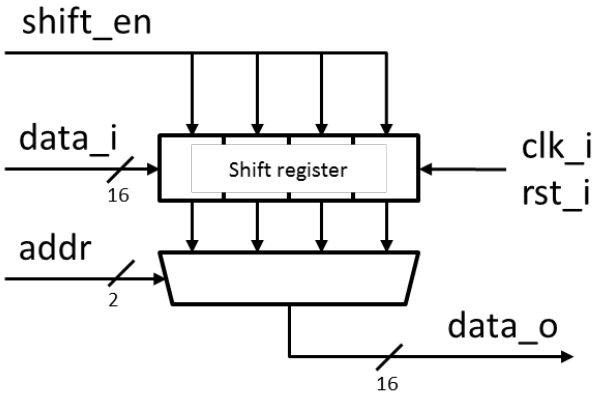
tap(0)..tap(3) are values of the constant tap defined in the package sfir_pack.vhd provided using Blackboard.

Block Diagrams of the Lower-Level Units

We will now discuss the lower level circuits of the serial FIR filter.

SHIFT RAM

The SHIFT RAM is where the data is stored. The data is shifted in, and then the address input can be used to scroll through all of the data. Clock and reset are not shown in the diagram but are connected to the respective inputs of Shift register.



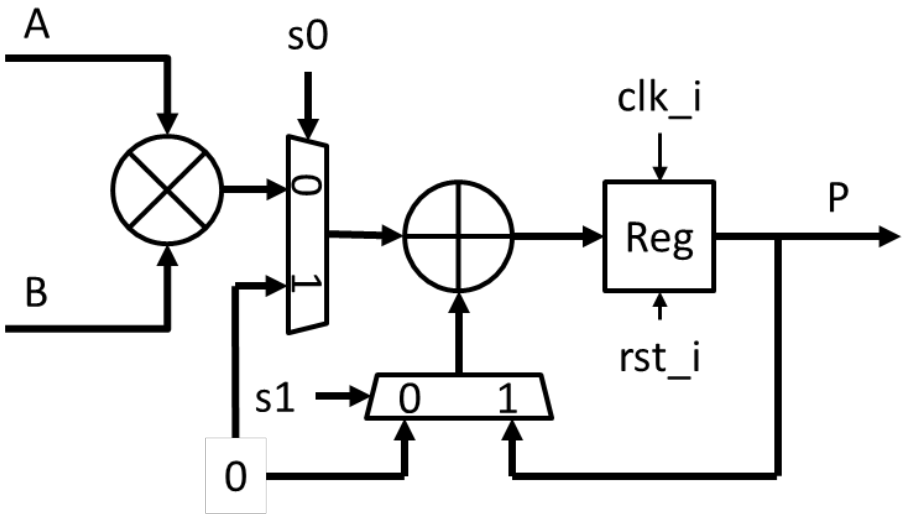
ALU

The Arithmetic Logic Unit (ALU) is the basic math processor. It has four (4) opcodes: reset, mult, macc, and hold. A description can be found in the following table:

Opcode	Description
clear	Reset the P register
mult	The P register gets the product of A and B
macc	The P register gets the product of A and B plus P
hold	The P register stays unaffected

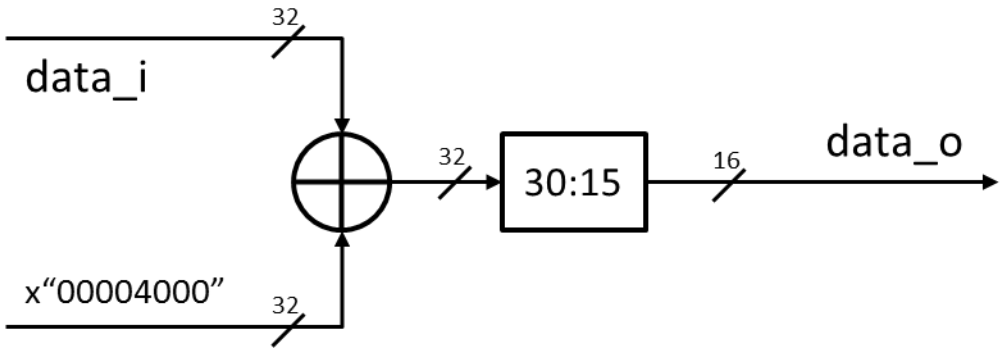
Based on the opcode input, set the select signals of the multiplexers shown in the block diagram below. Note the following:

1) P should be registered on the rising edge of the clock.
2) P is 32 bits (the size of A and B combined).
3) A and B should be treated as signed numbers, and therefore, so should P.

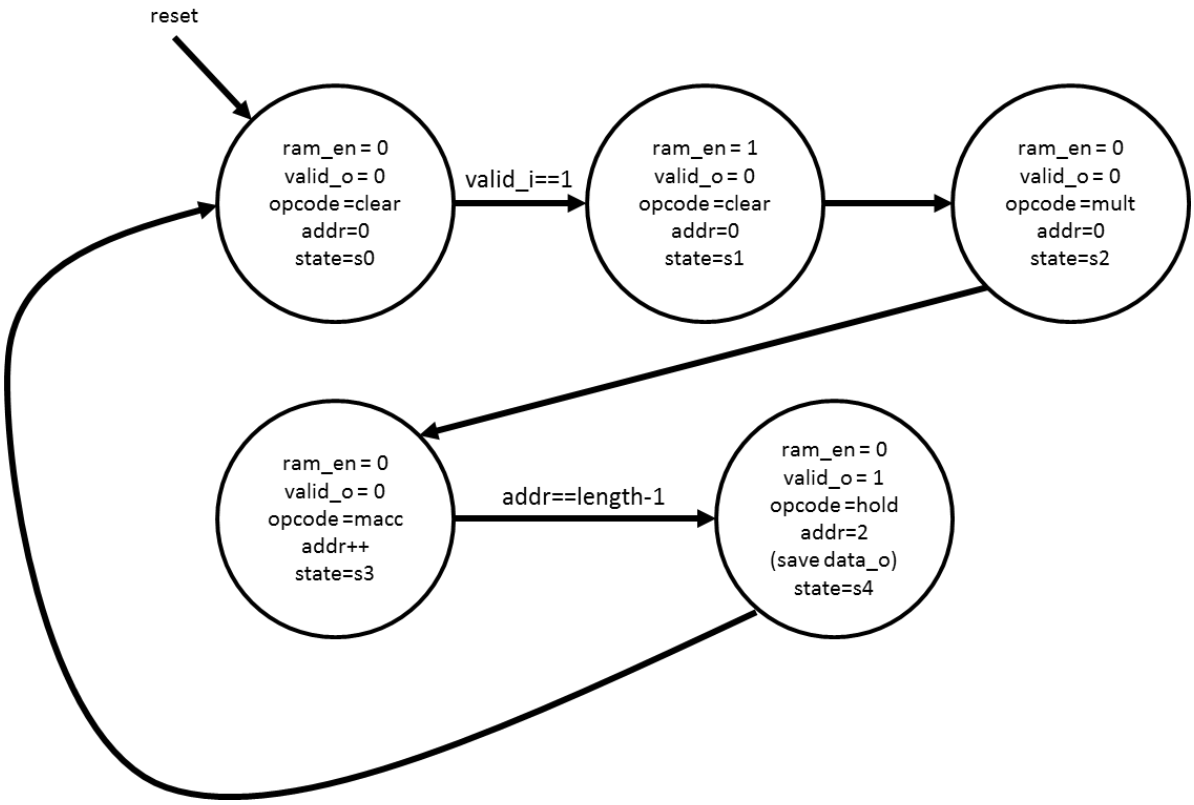


ROUND

It is too costly to continue to use all of the bits out of the multiplier. For this reason, we round the 32-bit data coming out of the FIR filter down to 16 bits. We need to keep the most significant bits (minus the extra sign bit), but truncating alone causes a DC offset. By adding a small offset before we truncate, we can round away from zero, and avoid the offset. After the addition, we take bits 30 downto 15. Note that there is no clock in this circuit.



State Machine Diagram of FSM



length is the length of the filter (the number of taps). It is assumed to be 4 for this circuit.

Testbench

We have included the test vector in the package to compare with the output of the circuit. That should help you debug the midterm. To make the test vector (the vector of outputs) paste the following code in your signal declaration section of your test bench:

```
constant test_vector   : word_vector := make_test_vector(filter_length);
```

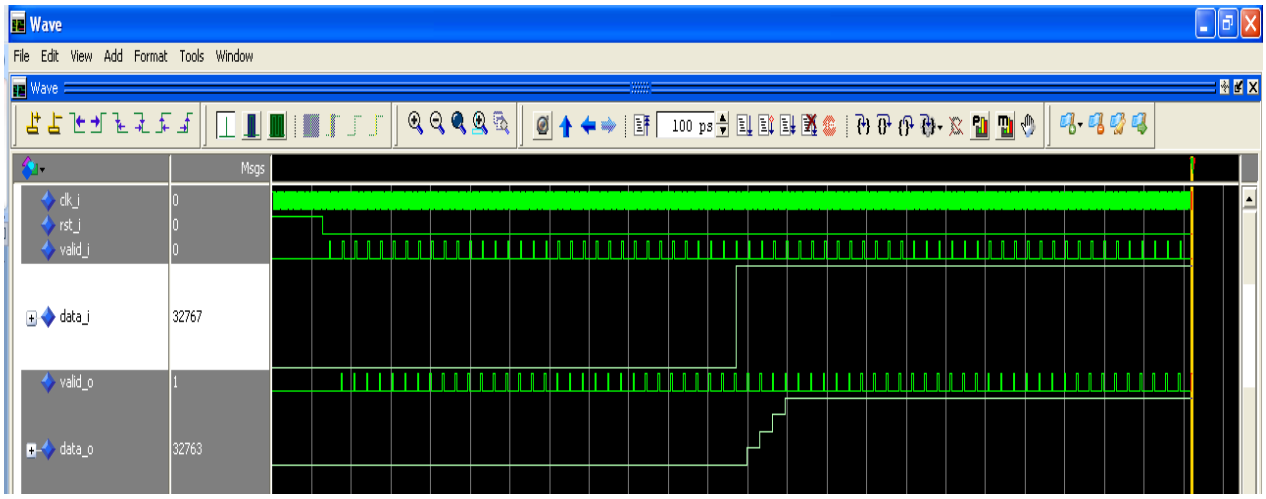
Test Plan

We will be testing our Serial FIR Filter by checking its step response. In your test bench, you should do the following:

1. Generate your 50 MHz clock
2. Input x"0000" for 32 consecutive samples (samples are signified by valid_i going high. valid_i should be high for one clock cycle and wait until after valid_o goes high before going high again.)
3. Input x"7FFF" for 32 samples
4. Stop the simulation

Your waveform should show all internal signals. Your input and output signals (data_i and data_o, respectively) should be shown as an analog signal (signed, 2's complement, height=100) to aid in testing.

Expected Waveform



Tasks

1. Write VHDL synthesizable code for the Serial FIR filter.
2. Write a testbench verifying the operation of your circuit.
3. Perform a functional simulation of your circuit. Debug your code.
4. Synthesize your circuit using Xilinx XST.
5. Implement your circuit using Xilinx ISE.
6. Perform timing simulation of your circuit using Active-HDL or ModelSim.
7. Based on the circuit block diagram and the Implementation reports, determine the most critical path in your circuit and its length.

Deliverables

1. VHDL code of your entire circuit fulfilling the requirements specified in the Design Requirements section above.
2. VHDL code of your testbench.
3. Functional simulation waveforms demonstrating the correct operation of your circuit.
4. Description of the critical path in your circuit.
5. Timing simulation waveforms demonstrating the critical path.
6. FPGA resource utilization.
7. Minimum clock period of your circuit.