

FIR Filter Module with Oversampling

Introduction

Design a digital system that will perform a filtering operation using a 4-tap FIR filter. The system clock rate is 4x faster than the incoming data rate, which allows efficient computation of the convolution operation using only one multiplier. A state machine will be required to signal when the filter is ready for new data on every 4th clock cycle as well as cycle through the addresses of the input data and filter coefficients to perform each of the 4 multiply/add operations required before the new data is made available.

The filter has 4 coefficients and will be stored in a ROM initialized in the code to:

```
h0 = 1
h1 = 10
h2 = 10
h3 = 1
```

The output is computed as follows:

```
y0 = x0*h0 + 0*h1 + 0*h2 + 0*h3
y1 = x1*h0 + x0*h1 + 0*h2 + 0*h3
y2 = x2*h0 + x1*h1 + 0*h2 + 0*h3
y3 = x3*h0 + x2*h1 + x1*h2 + 0*h3
y4 = x4*h0 + x3*h1 + x2*h2 + x1*h3
y5 = x5*h0 + x4*h1 + x3*h2 + x2*h3
y6 = x6*h0 + x5*h1 + x4*h2 + x3*h3
y7 = x7*h0 + x6*h1 + x5*h2 + x4*h3
.
.
.
```

As can be seen, there are 4 multiplies and 3 adds required per output, which are all performed before the next input is shifted in.

Then input to be used for simulations should be:

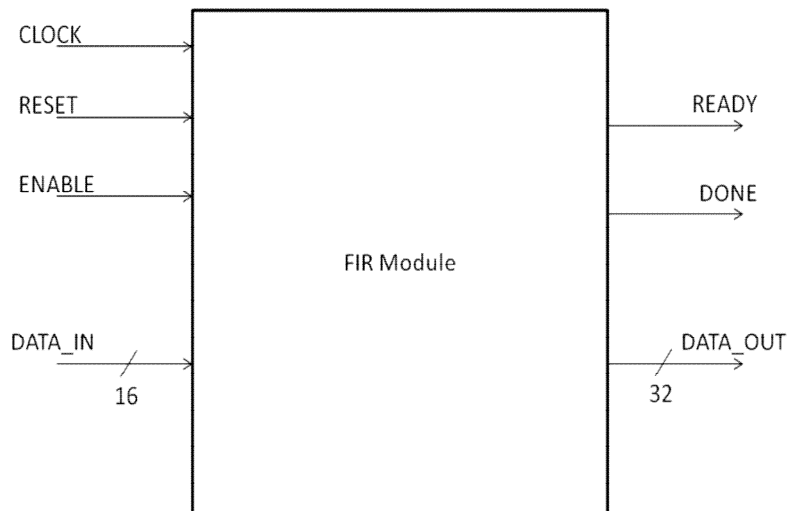
```
x0 = 1
x1 = 2
x2 = 3
x4 = 4
x5 = 5
x6 = 6
x7 = 7
x8 = 8
```

The pseudocode for the filtering operation is given below:

```
while(ENABLE = 1)
{
    DONE = 0;
    READY = 0;
    load new data into shift register (DATA_IN -> x0, x0->x1, x1->x2, x2->x3)
    y = 0;
    y = y + x0*h0;
    y = y + x1*h1;
    y = y + x2*h2;
    y = y + x3*h3;
    DATA_OUT = y;
    READY = 1

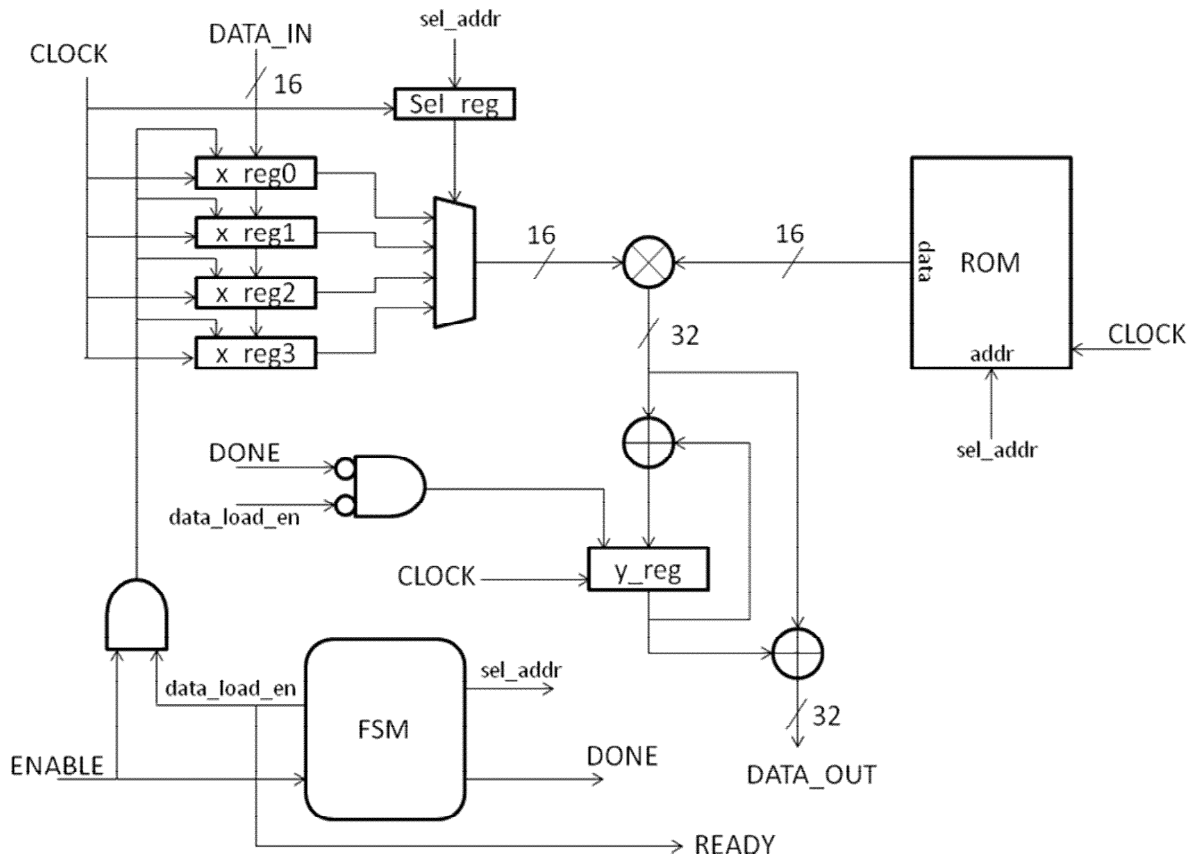
    if(ENABLE = 0)
    {
        DONE = 1;
    }
}
```

The interface of the circuit is shown below:

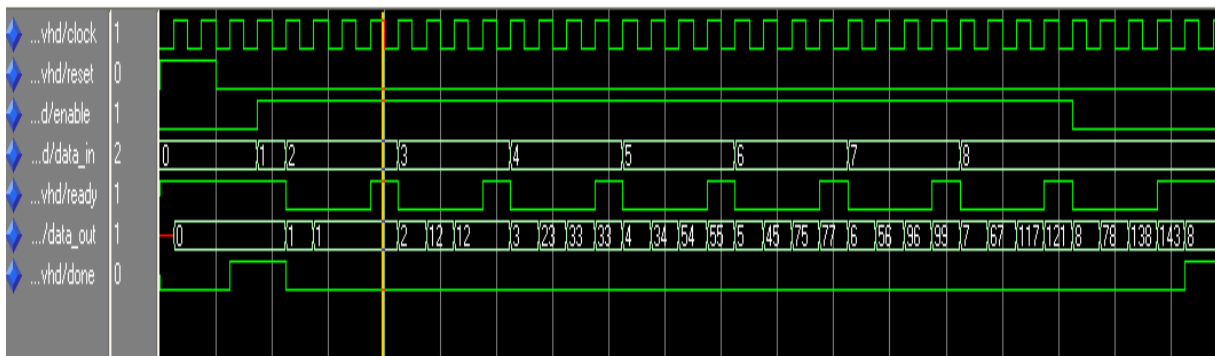


Signal	Mode	Size (bits)	Function
CLOCK	Input	1	20MHz, 50% duty cycle master clock
RESET	Input	1	Asynchronous reset
ENABLE	Input	1	Starts the computation cycle, active high while filter runs
DATA_IN	Input	16	Input data to the filter, new data is valid on the clock cycle when READY is high
READY	Output	1	Signals when FIR Module is ready for next input, and when DATA_OUT is valid
DONE	Output	1	Signals when the FIR Module is no longer processing
DATA_OUT	Output	32	Filtered data, valid when READY is high

The execution unit of the circuit is described below



The timing waveform is shown below for N = 4



Design Requirements

The combinational portion of the circuit should be described using the dataflow VHDL code, and the sequential portion of the circuit should be described using the synthesizable behavioral code. Your code should infer a circuit that requires a minimum amount of FPGA resources. The target clock frequency should be 40 MHz.

Tasks

Perform the following tasks:

1. Write a VHDL code of the execution unit of the described above circuit (shown in the block diagram above).
2. Write a testbench verifying the operation of your execution unit.
3. Perform functional simulation of your circuit and use it to debug your VHDL code.
4. Synthesize your circuit using Synplify Pro or Xilinx ISE. Save the RTL schematic.
7. Implement your circuit using Xilinx ISE.
8. Perform timing simulations of your circuit using Active-HDL or Xilinx ISE.
9. Run the static timing analysis of your circuit.
10. 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 testbenches
3. RTL schematic of your circuit
4. Timing waveforms from the functional simulation demonstrating the correct operation of your circuit.
5. Description of the critical path in your circuit
6. Timing waveforms from the timing simulation demonstrating the delay of the circuit most critical path
7. FPGA resource utilization
8. Minimum clock period of your circuit.