Fall
2024

FIR Filter Design

PROJECT 1 CE6325 VLSI DESIGN: VERILOG HDL

LAMIN JAMMEH NET-ID: DAL852207

Project Description:

A Finite Impulse Response (FIR) filter was designed with pre-determined specifications. The filter order, and the Data_in size were parameterized to allow the design to be scalable. The filter takes in a sample input, processes the sample input and passes it to an output register

Design Specifications

Filter order: 5
Filter coefficients: [7, 8, 9, 12, 4]
Data_in size: 4
Data_out size: 16

Data flow of the design

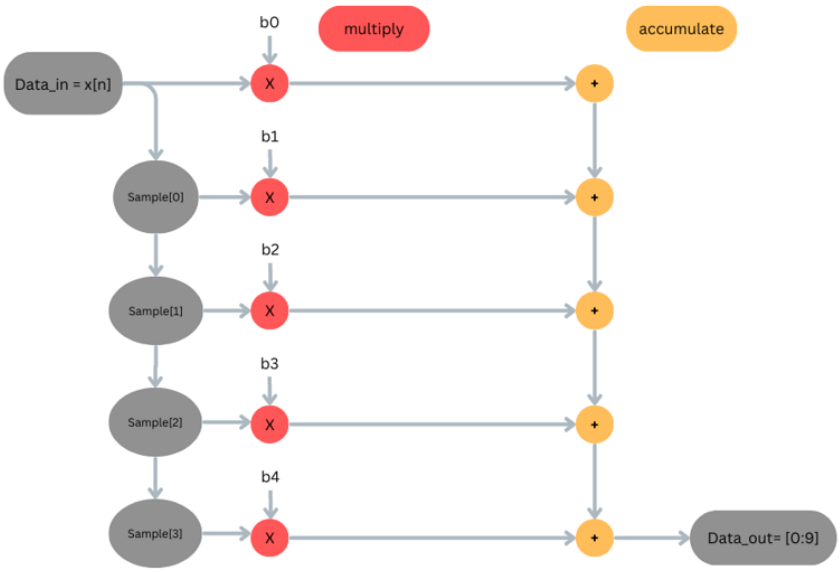


Figure1: Finite Impulse Response (FIR) Filter using Multiply and Accumulate

Design Netlist

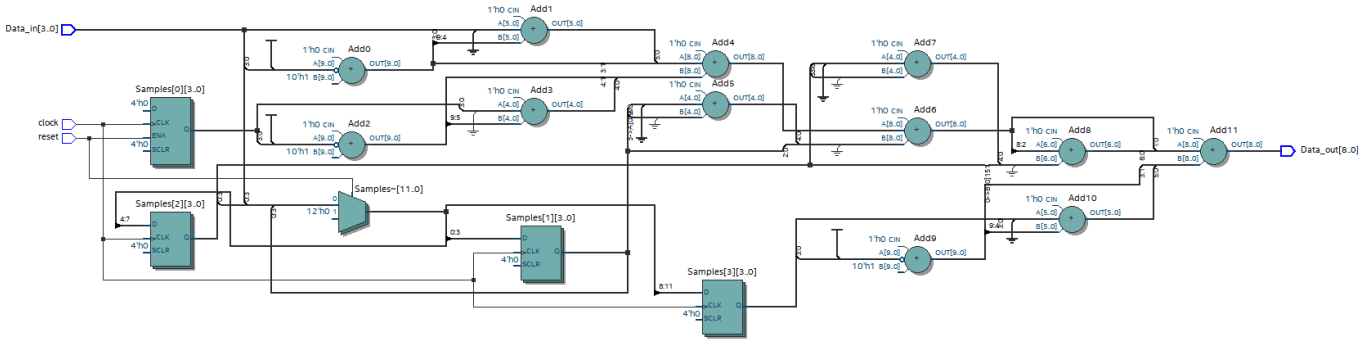


Figure 2: Netlist for the FIR filter

Testbench Process Flow

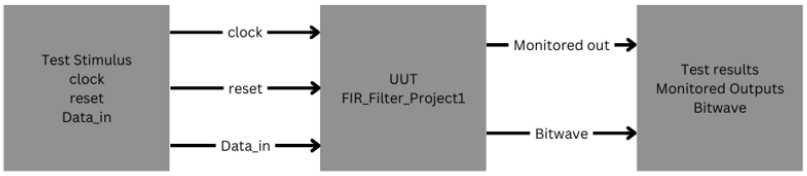


Figure 3: Shows the Testbench Process flow

Table 1 showing Filter out with when Data_in = decimal (6)

Data_in		Sample[k]	Filter Coefficient (bn)		bn * Sample[k]		Acc @ filter stage
6		6	7		42	acc0	42
	Data_in	6	8	bn * Sample[k]	48	acc1	90
		6	9		54	acc2	144
		6	12		72	acc3	216
		6	4		24	Data_out/acc4	240

Table 1 shows that Data_in will go shift through all of Sample[k] when enough time is allowed before changing the value at the input. The value in each register of Sample[k] is multiplied by the filter

coefficient (bn). The values of the multiply is accumulated or summed to form the output of the filter.

$$Mux = b_n * Sample[k]$$

$$acc = \sum_{n}^{0} Mux_n$$

Register Transistor Logic (RTL) Simulation Result

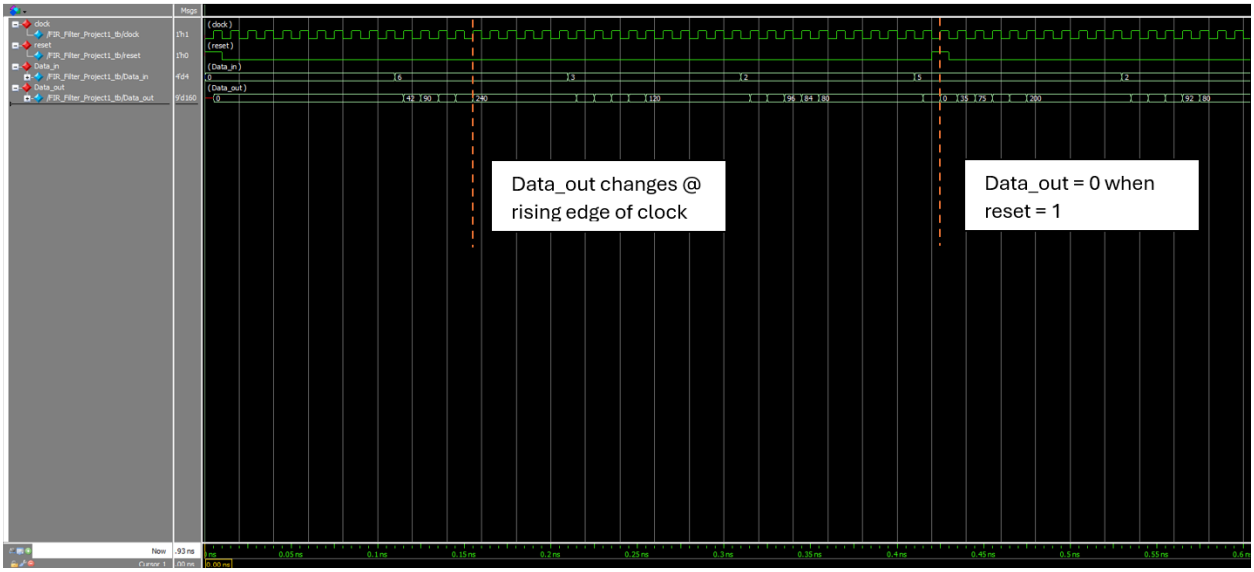
Table3 shows the observed values from the simulation results

```
# run -all
# Time:     0  | Data_in: 0  | Data_out:    x  | reset: 1
# Time:     5  | Data_in: 0  | Data_out:    0  | reset: 1
# Time:    10  | Data_in: 0  | Data_out:    0  | reset: 0
# Time:   110  | Data_in: 6  | Data_out:    0  | reset: 0
# Time:   115  | Data_in: 6  | Data_out:   42  | reset: 0
# Time:   125  | Data_in: 6  | Data_out:   90  | reset: 0
# Time:   135  | Data_in: 6  | Data_out:  144  | reset: 0
# Time:   145  | Data_in: 6  | Data_out:  216  | reset: 0
# Time:   155  | Data_in: 6  | Data_out:  240  | reset: 0
# Time:   210  | Data_in: 3  | Data_out:  240  | reset: 0
# Time:   215  | Data_in: 3  | Data_out:  219  | reset: 0
# Time:   225  | Data_in: 3  | Data_out:  195  | reset: 0
# Time:   235  | Data_in: 3  | Data_out:  168  | reset: 0
# Time:   245  | Data_in: 3  | Data_out:  132  | reset: 0
# Time:   255  | Data_in: 3  | Data_out:  120  | reset: 0
# Time:   310  | Data_in: 2  | Data_out:  120  | reset: 0
# Time:   315  | Data_in: 2  | Data_out:  113  | reset: 0
# Time:   325  | Data_in: 2  | Data_out:  105  | reset: 0
# Time:   335  | Data_in: 2  | Data_out:   96  | reset: 0
# Time:   345  | Data_in: 2  | Data_out:   84  | reset: 0
# Time:   355  | Data_in: 2  | Data_out:   80  | reset: 0
# Time:   410  | Data_in: 5  | Data_out:   80  | reset: 0
# Time:   415  | Data_in: 5  | Data_out:  101  | reset: 0
# Time:   420  | Data_in: 5  | Data_out:  101  | reset: 1
# Time:   425  | Data_in: 5  | Data_out:    0  | reset: 1
# Time:   430  | Data_in: 5  | Data_out:    0  | reset: 0
# Time:   435  | Data_in: 5  | Data_out:   35  | reset: 0
# Time:   445  | Data_in: 5  | Data_out:   75  | reset: 0
# Time:   455  | Data_in: 5  | Data_out:  120  | reset: 0
# Time:   465  | Data_in: 5  | Data_out:  180  | reset: 0
# Time:   475  | Data_in: 5  | Data_out:  200  | reset: 0
# Time:   530  | Data_in: 2  | Data_out:  200  | reset: 0
# Time:   535  | Data_in: 2  | Data_out:  179  | reset: 0
# Time:   545  | Data_in: 2  | Data_out:  155  | reset: 0
# Time:   555  | Data_in: 2  | Data_out:  128  | reset: 0
# Time:   565  | Data_in: 2  | Data_out:   92  | reset: 0
# Time:   575  | Data_in: 2  | Data_out:   80  | reset: 0
# Time:   630  | Data_in: 0  | Data_out:   80  | reset: 0
# Time:   635  | Data_in: 0  | Data_out:   66  | reset: 0
# Time:   645  | Data_in: 0  | Data_out:   50  | reset: 0
# Time:   655  | Data_in: 0  | Data_out:   32  | reset: 0
# Time:   665  | Data_in: 0  | Data_out:    8  | reset: 0
# Time:   675  | Data_in: 0  | Data_out:    0  | reset: 0
# Time:   730  | Data_in: 2  | Data_out:    0  | reset: 0
# Time:   735  | Data_in: 2  | Data_out:   14  | reset: 0
# Time:   745  | Data_in: 2  | Data_out:   30  | reset: 0
# Time:   755  | Data_in: 2  | Data_out:   48  | reset: 0
# Time:   765  | Data_in: 2  | Data_out:   72  | reset: 0
# Time:   775  | Data_in: 2  | Data_out:   80  | reset: 0
# Time:   830  | Data_in: 4  | Data_out:   80  | reset: 0
# Time:   835  | Data_in: 4  | Data_out:   94  | reset: 0
# Time:   845  | Data_in: 4  | Data_out:  110  | reset: 0
# Time:   855  | Data_in: 4  | Data_out:  128  | reset: 0
# Time:   865  | Data_in: 4  | Data_out:  152  | reset: 0
# Time:   875  | Data_in: 4  | Data_out:  160  | reset: 0
# ** Note: $stop    : C:/Users/lmnjm/OneDrive/Documents/Grad School/UTD/Fall 2024/EEDG 6325/Projects/Project 1/FIR_Filter_Project1_tb.v(66)
#    Time: 930 ps  Iteration: 0  Instance: /FIR_Filter_Project1_tb
```

When Data_in =6 Data_out=240 @ next rising clock after Data_in goes through all the filter taps or stages

@ reset = 1 Data_out = 0 since the Sample register is cleared @reset high. This happens @ next rising clock

Simulation Wave form



```
1  /*-------------------------------------------------------------------------
2  Name:    Lamin Jammeh
3  CLass:   CE6325 Fall_2024
4  Project: 1
5  Project Description: this is a Finite Impulse Response (FIR) filter design using Verilog HDL
6  The design follows the concepts below
7  **The filter order is selected and parameterized so the design can be scaled in the fututre
8  **The filter cooefficents are pre-determined
9  **Data_in samples will be provided in the testbench to determine the filter behavior
10 **The Data_in sample is Multiplied and accumalated through the diffrent filter stages/taps
11 **The Data_out word_size = word_in + coeff_size + [log2[N]]  where N= # of taps in the filte
12
13 -------------------------------------------------------------------------*/
14 module FIR_Filter_Project1 #(parameter order = 5,                          //filter
   order [pre-determined]
15                              parameter word_size_in = 4,                   //size of
   data_in [pre-determined]
16                              parameter word_size_out = 16)  //word_size = word_in +
   coeff_size + [log2[N]] N=4, & coeff=word_in
17
18                            //declare inputs and outputs
19                            (
20                            output   [word_size_out - 1:0]   Data_out,
21                            input    [word_size_in - 1:0]    Data_in,
22                            input                            clock, reset
23                            );
24
25 reg      [word_size_in -1:0]     Samples[order-1:0];     //temp storage for input samples
   (x(n))
26 reg      [word_size_out -1:0] acc;              //temp storage for output data
27 integer  k;
28
29 //Filter Coefficients
30 parameter b0 = 4'd7;
31 parameter b1 = 4'd8;
32 parameter b2 = 4'd9;
33 parameter b3 = 4'd12;
34 parameter b4 = 4'd4;
35
36 //define the formula for the output
37 assign   Data_out = acc;
38
39 always @(posedge clock)
40  begin
41    if (reset == 1)
42    //when reset is high and clock is rising samples[k] = 0 regardless of k value
43       begin
44          for (k = 0; k < order;  k = k+1)
45          begin
46             Samples[k] <= 0;
47             acc        <= 0;
48          end
49       end
50       else
51       //when reset is low and clock is rising compute Samples with data_in to get Data_out
52          begin
53             Samples[0] <= Data_in;                  //@ k=0 Samples[0] <= Data_in
54             for (k = 1; k < order;  k = k + 1)        // from k=1 to k=order Samples[k]
   <=Samples[k-1]
55             begin
56                Samples[k] <= Samples[k - 1];          //Data_in will go throught all the
   filter coefficeints
57                acc <=   b0*Data_in
58                     + b1*Samples[0]
59                     + b2*Samples[1]
60                     + b3*Samples[2]
61                     + b4*Samples[3];
62             end
63          end
```

```
64      end
65    endmodule
```

```
/*--------------------------------------------------------------------------------
Name:    Lamin Jammeh
CLass:   CE6325 Fall_2024
Project: 1
Project Description: Teestbench for Project1
**creeate a clock signal
**Initialize all the input signals
**apply some Sample Data_in as [6325] in 10 time unit intervals
**apply reset to test the reset signal
**apply anothe set of sample Data_in as [2024] in 10 time unit intervals
**monitor the signals and end the test

--------------------------------------------------------------------------------*/

module FIR_Filter_Project1_tb;

parameter order = 4;
parameter word_size_in = 4;
parameter word_size_out = 2 * word_size_in + 1;

//declare ports for the design
wire    [word_size_out -1:0]    Data_out;
reg     [word_size_in -1:0]      Data_in;
reg                             clock, reset;

//declare the unit under test UUT
FIR_Filter_Project1 #(order, word_size_in, word_size_out) UUT(.Data_out(Data_out),
                                                              .Data_in(Data_in),
                                                              .clock(clock),
                                                              .reset(reset)
                                                             );

// Instantiate the clock signal
initial
   begin
      clock = 0;
      forever #5 clock = ~clock;
   end

//Instantiate the diiferent test scenarios to validate the design
//*******Scenario 1 initialize all input signals
initial
   begin
      reset = 1;
      Data_in =0;

      #10   reset = 0;  //wait for 10 timing units for the inital signals to go through

//Scenario 2 apply some Data_in samples and observe the outputs use [6 3  2  5] @ 100 time unit intervals
      #100  Data_in = 4'd6;         //make sure time is long enough for Data_in to mkae it to the last filter COefficient
      #100  Data_in = 4'd3;
      #100  Data_in = 4'd2;
      #100  Data_in = 4'd5;

//Scenario 3 test the reset signal to validate the behavior
      #10   reset = 1;        //Sample registers should be cleared
      #10   reset = 0;        //Sample register will accept Data_in

//Scenario 4 apple more samples to make sure the design works after reset
      #100  Data_in = 4'd2;
      #100  Data_in = 4'd0;
      #100  Data_in = 4'd2;
      #100  Data_in = 4'd4;

//stop the test
      #100  $stop;
   end
```

```
68
69    // Monitor output
70    initial
71       begin
72          $monitor("Time: %4t\t | Data_in: %0d\t | Data_out: %3d\t | reset: %b", $time,
      Data_in, Data_out, reset); //use %10 as padding for time and data_out
73       end
74
75    endmodule
76
77
78
```