EE417 Advanced Digital Logic with Verilog HDL Name: Ron Kalin / Lamin Jammeh Date: 08/02/24

Lesson 12 **Final Project**

Multiply and Accumulate MAC operation

Objective

In computing, especially digital signal processing, the multiply–accumulate (MAC) or multiply-add (MAD) operation is a common step that computes the product of two numbers and adds that product to an accumulator. The hardware unit that performs the operation is known as a multiplier–accumulator (MAC unit); the operation itself is also often called a MAC or a MAD operation. The MAC operation modifies an accumulator a:
$a \leftarrow a + (b \times c)$ [1]

The FIR MAC (Multiply and Accumulator) code was designed and tested the coherence of the data and find the latency (in terms of clock cycles) for each design. The actual propagation delay in ns for the design was found without pipelining and with different locations for the cut-set. Comment on the latency and alignment of data in your simulation results. Registers will be added at the outputs of all the multipliers. This way multiplication will occur in one clock cycle and then addition will occur in the following clock cycle. Report the latency and the Hardware usage in this case. [1]

This design will implement a FIR MAC 4th order Datapath-controller structure with the use of pipelining. The Verilog design includes a testbench to verify the results. The testbench input data and results were saved as a text file and plotted in MATLAB. Timing Analysis is included, and report information will show the maximum throughput or clock frequency that can be implemented on the design. A block diagram will be presented that clearly shows the Datapath-controller structure of the design with the datapath and control signals defined. The state graph of the controller will be shown along with the building blocks of the datapath. Finally Simulation results in Questa with internal probes for functionaslity verification will be depicted showing specific cases, such as reset, dataflow between registers with expected delays … etc.

Application

Modern computers may contain a dedicated MAC, consisting of a multiplier implemented in combinational logic followed by an adder and an accumulator register that stores the result. The output of the register is fed back to one input of the adder, so that on each clock cycle, the output of the multiplier is added to the register. Combinational multipliers require a large amount of logic, but can compute a product much more quickly than the method of shifting and adding typical of earlier computers [1]

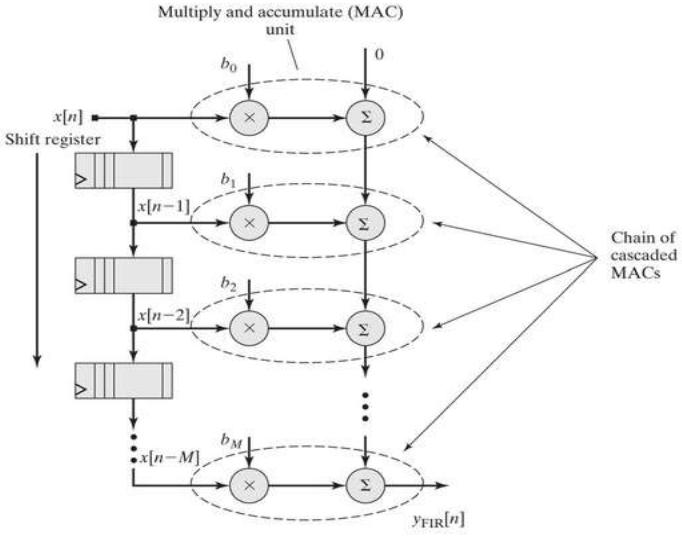


Figure 12.1 - FIR MAC diagram

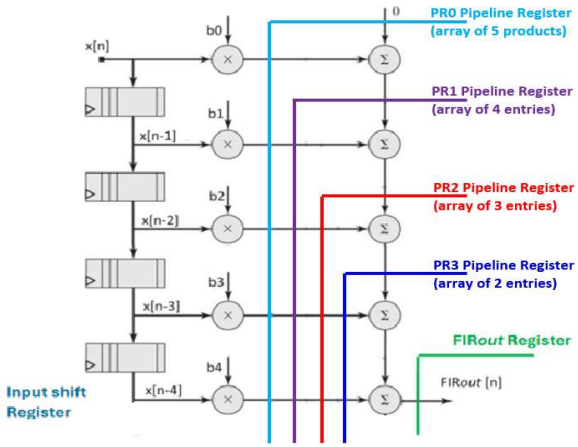


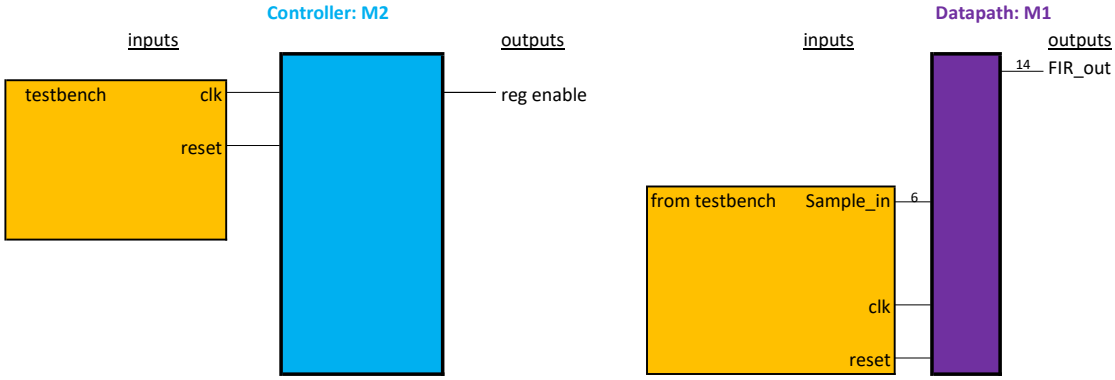
Figure 12.2 -
FIR MAC diagram, 4th order, showing pipelining cutlines

Design Methodology

A controller - datapath structure will be utilized with a reduced number of states vs. a FSM design

A block diagram and Finite State Machine (FSM) were developed from the Datapath in order to design the Controller code.
Ready was added as an output to the Datapath and as an input to the Controller.

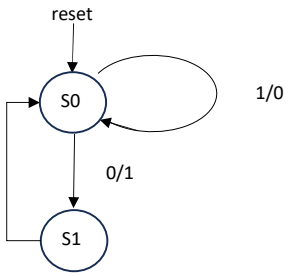
Block Diagram



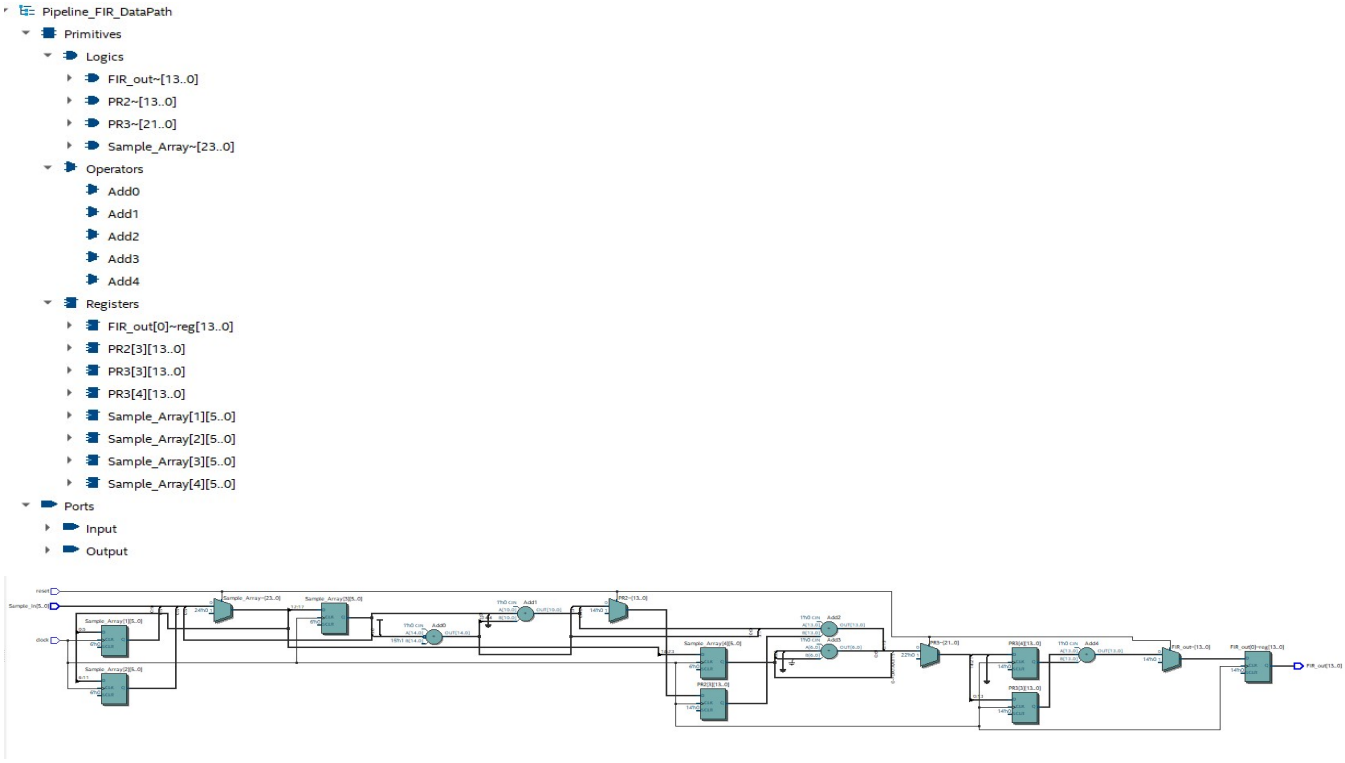
FSM (State Graph) for Controller

State Table

input	output	
reset	enable	state
1	0	S0
0	1	S1



Building Blocks of the Datapath - RTL Viewer



Design Verilog HDL Code -

Controller

ate: August 02, 2024 Pipeline_FIR_Controller.v Project: Pipeline_FIR_DataPath

```
1   // ee417 Final Project L12A2
2   // Name: Ron Kalin / Lamin Jammeh, Date: 08-02-24  Group: Kalin/Jammeh
3   // Design: FIR filter chain of cascaded MACs, given 5-bit coefficients
4   // inputs are 6-bit positive values, output register is 14 parallel bits
5   // Project Description: Controller module drives the enable signal, which enables the
6   // computation of the FIR_MAC
7
8   module Pipeline_FIR_Controller  (
9                                    input clock,
10                                   input reset,
11                                   output reg enable
12                                  );
13
14  // Control logic for the FIR filter
15  always @ (posedge clock or posedge reset)
16  begin
17     if (reset)
18        begin
19          enable <= 0;
20        end
21        else
22        begin
23          enable <= 1;
24        end
25  end
26
27  endmodule
28
```

Datapath

Date: August 02, 2024 Pipeline_FIR_DataPath.v Project: Pipeline_FIR_DataPa

```
1   /*-----------------------------------------------------------------------------------------------
    --------------
2   Name Lamin Jammeh / Ron Kalin
3   CLass: EE417 Summer 2024
4   FINAL PROJECT: Datapath
5   Group: Ron Kalin/ Lamin Jammeh
6   Project Description: Datapath module computesfilters input Sample by Multiplying and
    Accumulating
7   -----------------------------------------------------------------------------------------------
    --------------*/
8
9   module Pipeline_FIR_DataPath  (FIR_out, Sample_in, clock, reset);
10
11  //define the parameter sets for the design
12  parameter FIR_order        = 4;
13  parameter Sample_size      = 6;                  //maximum sample value is 63
14  parameter weight_size      = 5;                  //maximum value may be 31
15  parameter word_size_out    = Sample_size + weight_size + 3; //log2(2^2 * 2^5 * (order+1))
16
17  //define output
18  output reg [word_size_out -1:0]  FIR_out;
19
20  //define inputs
21  input     [Sample_size -1:0]      Sample_in;
22  input                             clock, reset;
23
24  //define the filter coefficients
25  parameter   b0 = 5'd3;
26  parameter   b1 = 5'd7;
27  parameter   b2 = 5'd20;
28  parameter   b3 = 5'd7;
29  parameter   b4 = 5'd3;
30
31  reg         [Sample_size -1:0]   Sample_Array[1:FIR_order]; //5th coefficient multiplied by
    Data_in
32  integer     k;
33
34  //define PR0 to PR3 as registers
35  reg      [word_size_out -1:0] PR0 [0:FIR_order];
36  reg      [word_size_out -1:0] PR1 [1:FIR_order];
37  reg      [word_size_out -1:0] PR2 [2:FIR_order];
38  reg      [word_size_out -1:0] PR3 [3:FIR_order];
39
40  //define the transition logic
41  always @ (posedge clock)
42     if (reset == 1)
43     /*---------------------------------------
44     if reset is high do the following
45     ****set all Pipeline registers to zero
46     *************IR = 0      Input register
47     *************PR[0:order-1] = 0    Pipeline register
48     *************OR = 0      Output register
49     ---------------------------------------*/
50        begin
51        //The input shift register
52           for (k=1; k <= FIR_order; k = k + 1)
53              Sample_Array[k] <= 0;
54
55        //The pipeline register
56           for (k = 0; k <= FIR_order; k = k + 1)
57              PR0[k] <= 0;
58        //The pipeline register
59           for (k = 1; k <= FIR_order; k = k + 1)
60              PR1[k] <= 0;
61        //The pipeline register
62           for (k = 2; k <= FIR_order; k = k + 1)
63              PR2[k] <= 0;
64        //The pipeline register
65           for (k = 3; k <= FIR_order; k = k + 1)
66              PR3[k] <= 0;
```

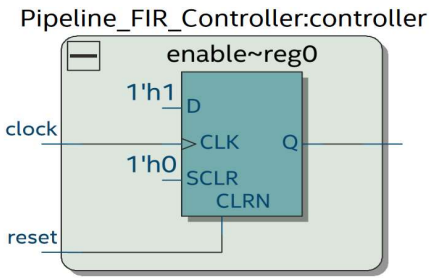
```
67          //The outpput register
68                FIR_out <= 0;
69          end
70       else
71       /*----------------------------------------
72       if reset is low do the following
73       *************1 => move the Sample in into a cutset (Input register) to reduce idle time
      of the input
74       *************2 => insert the PR at the input of the add and perform x[n] * b(n) and save
      in Pipeline register (PRn[n-1])
75       *************3 => add all the PR registers at the input of the output register and save
      in the Output register
76       ----------------------------------------*/
77          begin
78          //The input shift register
79             Sample_Array[1] <= Sample_in;
80             for (k = 2; k <= FIR_order; k = k + 1)
81                Sample_Array[k] <= Sample_Array[k-1];
82
83          //The pipeline register at PR0
84              PR0[0] <= b0 * Sample_in;
85              PR0[1] <= b1 * Sample_Array[1];
86              PR0[2] <= b2 * Sample_Array[2];
87              PR0[3] <= b3 * Sample_Array[3];
88              PR0[4] <= b4 * Sample_Array[4];
89
90          //The pipeline register at PR1
91              PR1[1] <= b1 * Sample_Array[1] + PR0[1];
92              PR1[2] <= b2 * Sample_Array[2];
93              PR1[3] <= b3 * Sample_Array[3];
94              PR1[4] <= b4 * Sample_Array[4];
95
96          //The pipeline register at PR2
97              PR2[2] <= b2 * Sample_Array[2] + PR1[2];
98              PR2[3] <= b3 * Sample_Array[3];
99              PR2[4] <= b4 * Sample_Array[4];
100
101         //The pipeline register at PR3
102             PR3[3] <= b3 * Sample_Array[3] + PR2[3];
103             PR3[4] <= b4 * Sample_Array[4];
104
105         //The outpput register
106               FIR_out <= PR3[3] + PR3[4];
107         end
108   endmodule
109
```

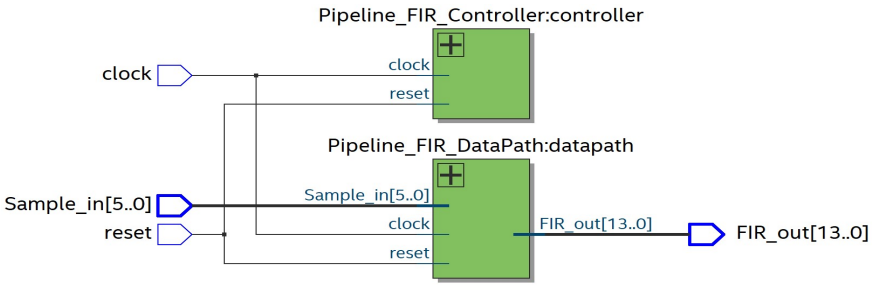
Top level module

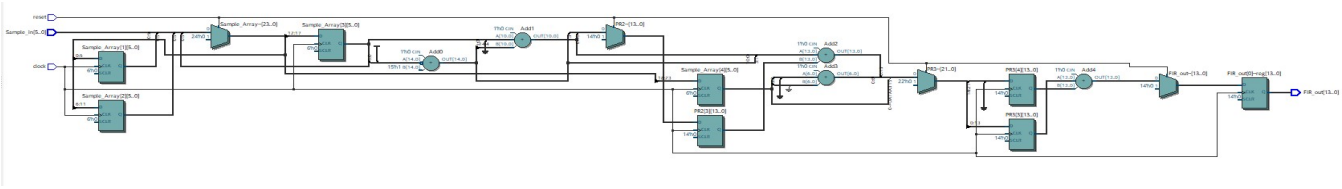
Date: August 02, 2024 Pipeline_FIR_MAC.v Project: Pipeline_FIR_DataPath

```
1   // Name Lamin Jammeh / Ron Kalin
2   // CLass: EE417 Summer 2024
3   // FINAL PROJECT: FIR MAC module
4   // Group: Ron Kalin/ Lamin Jammeh
5   // Summary: Module combines Datapath with controller to form a FIR_MAC
6
7   module Pipeline_FIR_MAC (FIR_out, Sample_in, clock, reset);
8
9   // Define the parameter sets for the design
10  parameter FIR_order        = 4;
11  parameter Sample_size      = 6;                          // Max sample value 2^n= 63
12  parameter weight_size      = 5;                          // Max value may be 31
13  parameter word_size_out    = Sample_size + weight_size + 3;  // log2(2^2 * 2^5 * (order+1))
14
15  //define the outputs
16  output [word_size_out - 1:0] FIR_out;
17
18  //define the inputs
19  input  [Sample_size - 1:0]   Sample_in;
20  input                        clock, reset;
21
22  // Internal signals
23  wire enable;
24
25  // Instantiate the DataPath module
26  Pipeline_FIR_DataPath #(FIR_order, Sample_size, weight_size, word_size_out) datapath (
27      .FIR_out(FIR_out),
28      .Sample_in(Sample_in),
29      .clock(clock),
30      .reset(reset)
31  );
32
33  // Instantiate the Controller module
34  Pipeline_FIR_Controller controller (
35      .clock(clock),
36      .reset(reset),
37      .enable(enable)
38  );
39
40  endmodule
41
```

RTL Viewer


Pipeline_FIR_Controller:controller
enable~reg0
1'h1
D
clock
CLK
Q
1'h0
SCLR
CLRN
reset


Pipeline_FIR_Controller:controller
clock
clock
reset
Pipeline_FIR_DataPath:datapath
Sample_in[5..0]
Sample_in[5..0]
reset
clock
FIR_out[13..0]
FIR_out[13..0]
reset

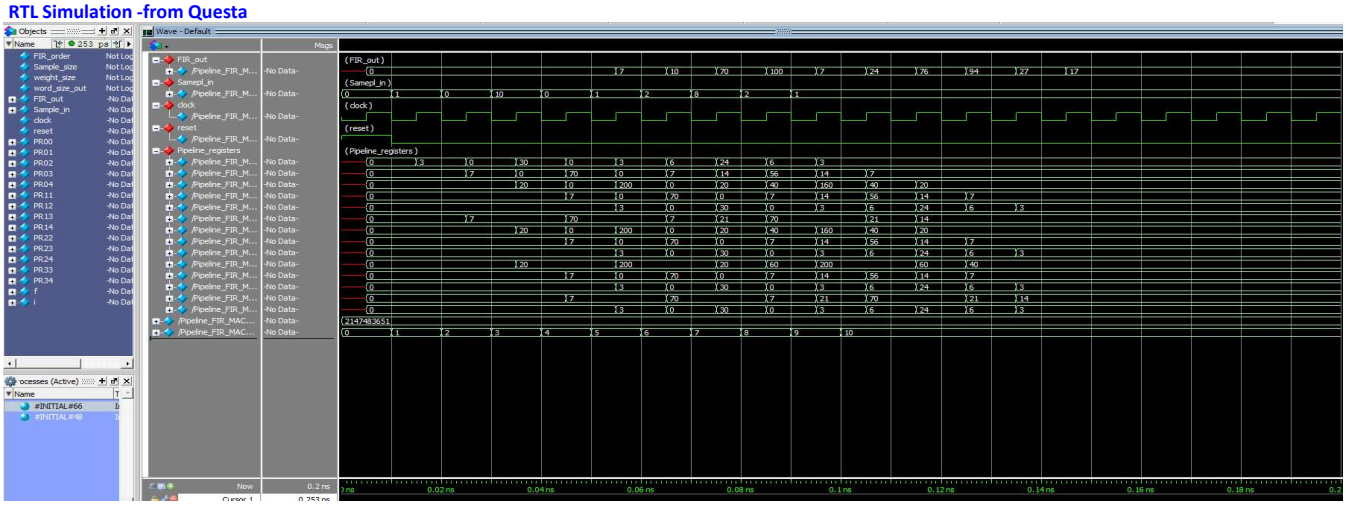


Testbench Verilog HDL Code


```
/*-------------------------------------------------------------------------------------------------
---------
Name Ron Kalin / Lamin Jammeh
CLass: EE417 Summer 2024
FINAL PROJECT: FIR MAC module
Group: Ron Kalin/ Lamin Jammeh
Project Description: testbench for the FIR_MAC with Pipelining
-------------------------------------------------------------------------------------------------
--------*/

module Pipeline_FIR_MAC_tb ;

// Define the parameter sets for the design
parameter FIR_order          = 4;
parameter Sample_size        = 6;    // maximum sample value is 63
parameter weight_size        = 5;    // maximum value may be 31
parameter word_size_out      = 2 * Sample_size + 2;  // maximum possible output 63*31*(4+1)

// Define the wires and registers for the test bench
wire     [word_size_out -1:0]     FIR_out;
reg      [Sample_size -1:0]       Sample_in;
reg                               clock, reset;

// Define the unit under test UUT
Pipeline_FIR_MAC  UUT (FIR_out, Sample_in, clock, reset);

// Define Probes to observe the Pipeline register PR0
wire     [word_size_out -1:0]     PR00;    assign PR00 = UUT.datapath.PR0[0];
wire     [word_size_out -1:0]     PR01;    assign PR01 = UUT.datapath.PR0[1];
wire     [word_size_out -1:0]     PR02;    assign PR02 = UUT.datapath.PR0[2];
wire     [word_size_out -1:0]     PR03;    assign PR03 = UUT.datapath.PR0[3];
wire     [word_size_out -1:0]     PR04;    assign PR04 = UUT.datapath.PR0[4];

// Define Probes to observe the Pipeline register PR1
wire     [word_size_out -1:0]     PR11;    assign PR11 = UUT.datapath.PR1[1];
wire     [word_size_out -1:0]     PR12;    assign PR12 = UUT.datapath.PR1[2];
wire     [word_size_out -1:0]     PR13;    assign PR13 = UUT.datapath.PR1[3];
wire     [word_size_out -1:0]     PR14;    assign PR14 = UUT.datapath.PR1[4];

// Define Probes to observe the Pipeline register PR2
wire     [word_size_out -1:0]     PR22;    assign PR22 = UUT.datapath.PR2[2];
wire     [word_size_out -1:0]     PR23;    assign PR23 = UUT.datapath.PR2[3];
wire     [word_size_out -1:0]     PR24;    assign PR24 = UUT.datapath.PR2[4];

// Define Probes to observe the Pipeline register PR3
wire     [word_size_out -1:0]     PR33;    assign PR33 = UUT.datapath.PR3[3];
wire     [word_size_out -1:0]     PR34;    assign PR34 = UUT.datapath.PR3[4];

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

// Instantiate and toggle the reset signal
initial
   begin
      reset = 1;
      #10 reset = 0;
   end

// Integer for file handle
integer f;
integer i;

// Apply different input samples and observe the outputs
initial
   begin
      f = $fopen("output.txt", "w");
```

```
69          $fwrite(f, "\t\tTime\tSample_in\tFIR_out\n" );
70
71          // Apply the input samples and log the output
72          for (i = 0; i < 10; i = i + 1)
73             begin
74                case (i)
75                    0:  Sample_in = 0;
76                    1:  Sample_in = 1;
77                    2:  Sample_in = 0;
78                    3:  Sample_in = 10;
79                    4:  Sample_in = 0;
80                    5:  Sample_in = 1;
81                    6:  Sample_in = 2;
82                    7:  Sample_in = 8;
83                    8:  Sample_in = 2;
84                    9:  Sample_in = 1;
85                    10: Sample_in = 0;
86                    11: Sample_in = 63;
87                    12: Sample_in = 0;
88                    default: Sample_in = 0;
89                endcase
90                #10; // Wait for the output to settle
91                $fwrite(f, "%d\t   %d\t      %d\n", $time, Sample_in, FIR_out);
92             end
93
94             $fclose(f);
95             #100 $stop;
96          end
97
98    endmodule
99
```

RTL Simulation -from Questa



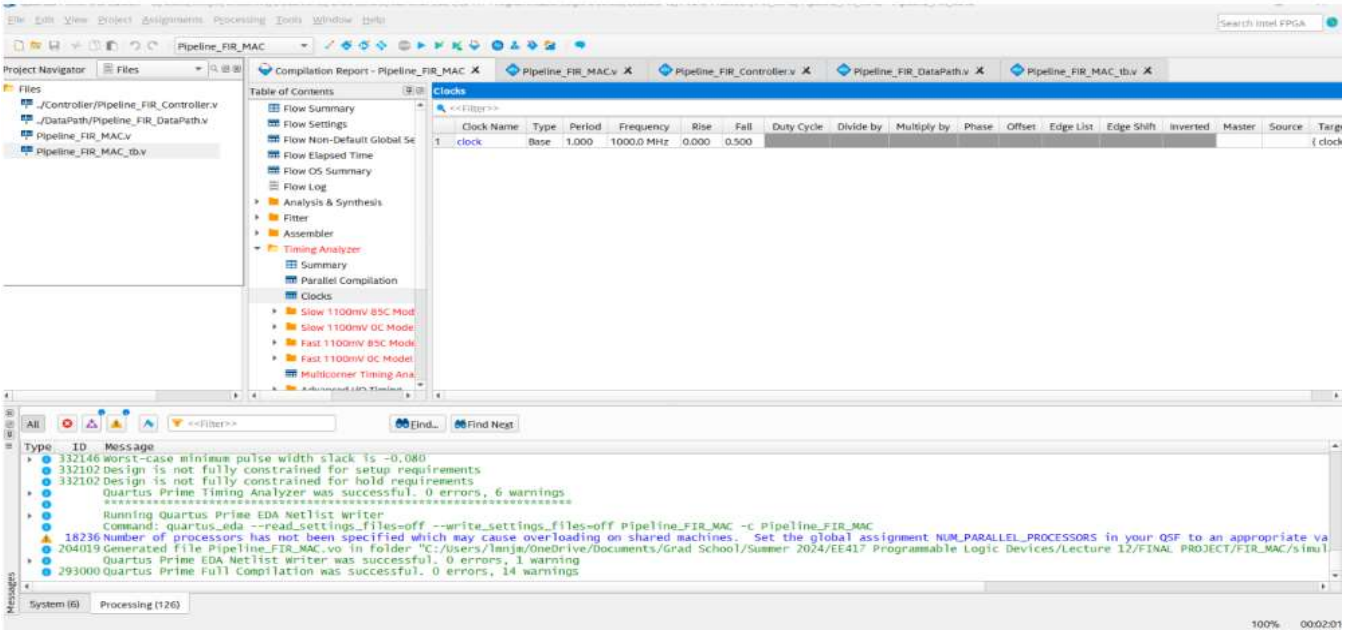
Testbench results from the text file

Time	Sample_in	FIR_out
10	0	0
20	1	0
30	0	0
40	10	0
50	0	0
60	1	7
70	2	10
80	8	70
90	2	100
100	1	7

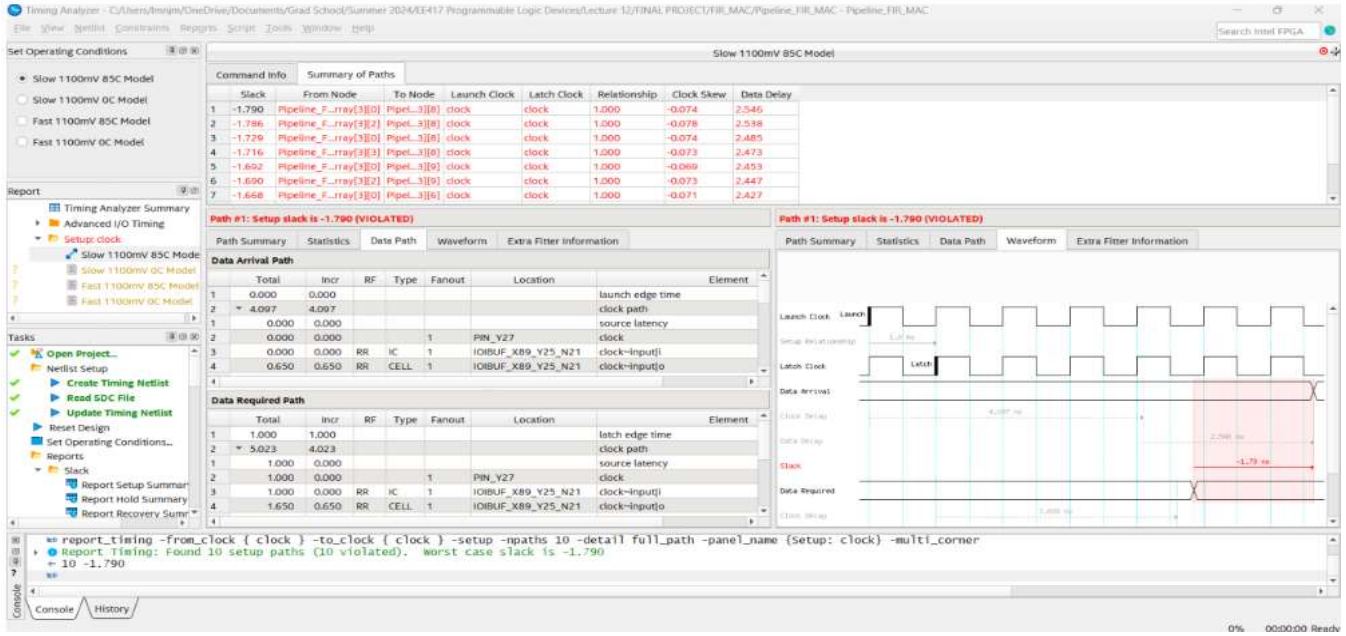
Timing Analysis Reports

Timing Analysis for the clock signal

@ default time when clock period is 1ns

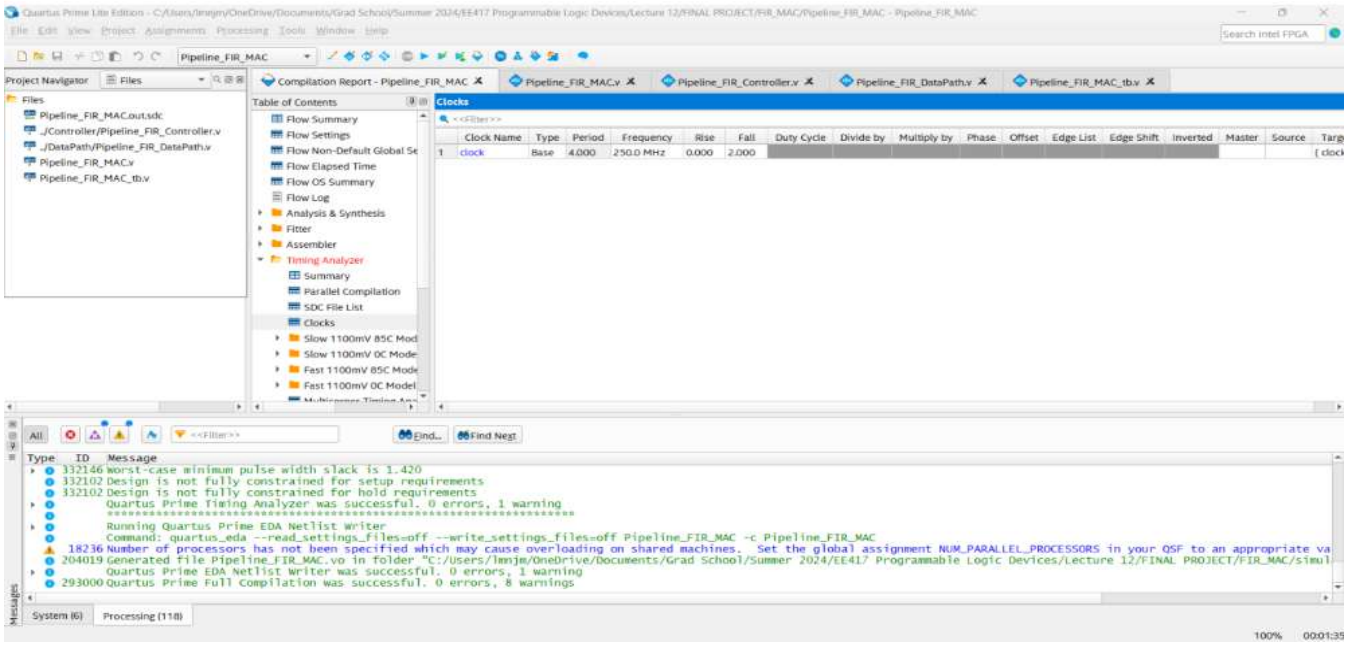


Time report at a slack of -1.790

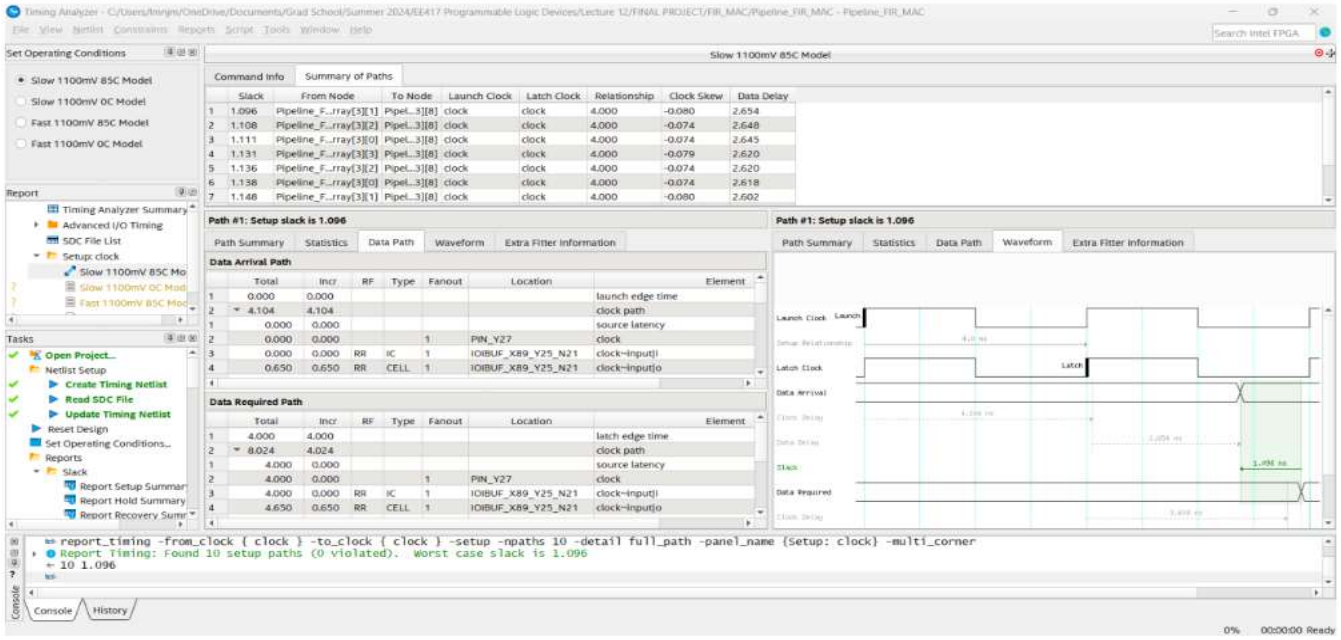


Timing Analysis Reports (cont'd)

@ default time when clock period is 4ns

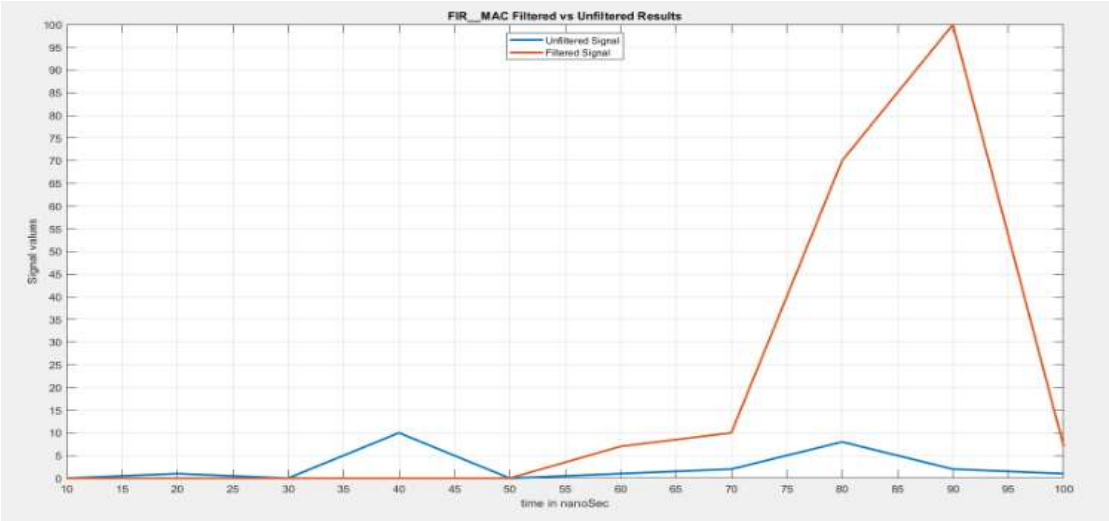


Time report at a slack of 1.096



Matlab Testnehc Data input and Results

```
% Name Lamin Jammeh
% CLass: EE417 Summer 2024
% Lesson 10 HW Question 3
% Group: Ron Kalin/ Lamin Jammeh
% Project Description: The Testbench results are ploted to show the filtered and unfiltered signals
%

%Step1 define the TestBench result
time = 10:10:100;
Sample_in = [0, 1, 0, 10, 0, 1, 2, 8, 2, 1];
FIR_out = [0, 0, 0, 0, 0, 7, 10, 70, 100, 7];

%Step2 plot the TestBench results
plot(time, Sample_in, 'LineWidth',2);
hold on;
plot(time, FIR_out, 'LineWidth',2);
grid on;
legend('Unfiltered Signal','Filtered Signal', 'Location','north');

title('FIR__MAC Filtered vs Unfiltered Results');
xticks(0:5:100);
yticks(0:5:100);
xlabel('time in nanoSec');
ylabel('Signal values');
hold off;
```



Conclusion

Reset high means Sample_input, internal registers, and output registers become zero.
When reset is low FIR filter is active. When filter is active, the actions happen on the positive clock edge.

FIR filter with pipeline registers were placed all adder inputs.
Simulation results show each pipelining register passes the values saved in its array to the next one.
The top two values are added together while the rest pass as is to the next register. The throughput (new output) is available at every clock cycle.
With the pipelining, the longest propagation delay for the combinational logic between the registers can be shortened, which will consequently allow reducing clock period and increasing clock frequency. With an output available at every clock cycle, the throughput of the module increases. The pipelining improves hardware utilization efficiency.

Sources

[1] Wikipedia "Multiply-accumulate operation"
[2] Ciletti, Michael D.. Advanced Digital Design with the Verilog HDL (p. 583). Pearson Education. Kindle Edition.