EE417 Advanced Digital Logic with Verilog HDL Date: 08/03/24

Group 3: Lamin Jammeh/Ron Kalin

Lesson 12 Final Project

Finite Impulse Response Multiply-Accumulate FIR_MAC Project

Objective

In Digital Signal Processing (DSP), an input signal can be filtered using a Finite Impulse Response with Multiply-Accumulator (FIR_MAC) filter. The filter coefficient can be calculated using MATLAB using the fir1 function giving the design specifications (cut-off frequency, Max-frequency, filter order, and filter Type) are already determined. The filter function is used in MATLAB with a sample signal s to determine if the given coefficient can filter the sample signal s. The coefficients and the filter order are then used to implement the design of the FIR_MAC in DSP.

The Finite Impulse Response with Multiply-Accumulator (FIR_MAC) filter in this design will take in an input signal and multiply the signal with coefficient. The output of the multiply is temporarily stored in a pipeline register (PR) to reduce hardware idle time, thereby improving the latency of the clock signal. A similar PR is implemented at the input of the adder which performs the Accumulator section. The output of the Multiplier and the input of the adder and summed to delivered to input of the next adder on the next filter order. The number of adders and multiplexers is determined by the filter order of the design. To maintain the signal coherency a PR is added at the output of each Multiplier and input of each Accumulator [2].

The filter in this design has two submodules and a main or top module. The two submodules are the controller and the Datapath modules. The controller module enables the clock and reset signals. The Datapath looks for the posedge of the clock signal and the state of the reset signal. When reset is high; all registers and output of the FIR_MAC goes to zero, and at reset low the Datapath module start processing input signals on the next posedge of the clock. To make sure the time requirement for the hardware is met the clock period was change from 1.0ns to 4.0ns, this changes the slack time from negative to positive tog give the hardware some leeway in case of propagation delay[3].

Application

Processing analog signals can come with lots of challenges since the analog signal are prone to noise which can be very difficult and expensive to clean. These can lead to wrong data especially from biomedical devices. The application of FIR_MAC filters in digital signal processing suppresses the Powerline Interference (PLI)[1]. When FIR_MAC filters are implemented in FPGA devices, the filter can be used for multiple DSP applications without the need for hardware change. The filter coefficients and filter order can be changed at the programming level. This is why FIR_MAC filters are preferred in instrumentation and measurement devices.

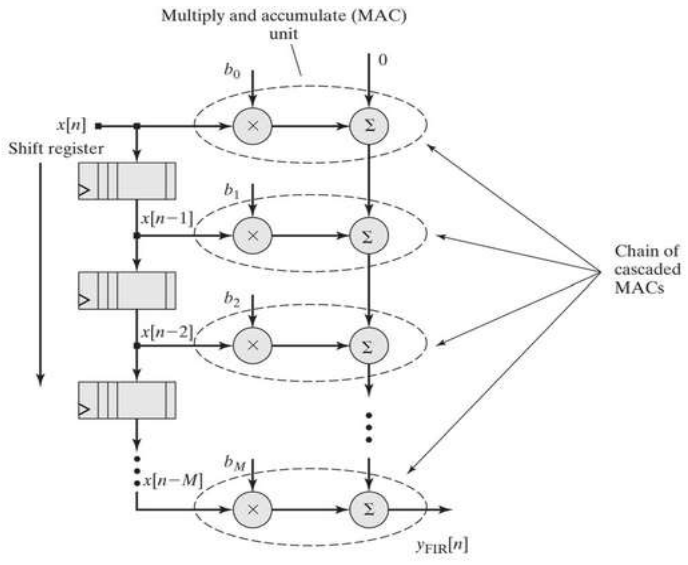


Figure 12.1 - FIR MAC diagram

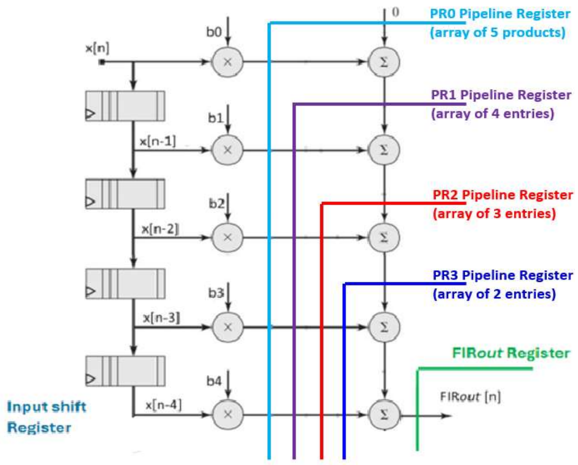

PR0 Pipeline Register (array of 5 products)
PR1 Pipeline Register (array of 4 entries)
PR2 Pipeline Register (array of 3 entries)
PR3 Pipeline Register (array of 2 entries)
FIRout Register
FIRout [n]
Input shift Register
x[n]
x[n-1]
x[n-2]
x[n-3]
x[n-4]
b0
b1
b2
b3
b4
0

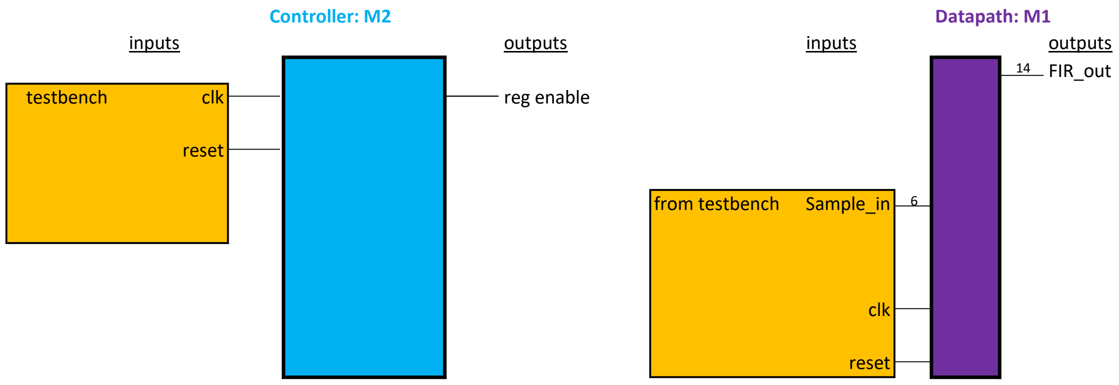
Figure 12.2 -
FIR MAC diagram, 4th order, showing pipelining cutlines

Design Methodology

A controller - datapath structure will be utilized with a reduced number of states vs. a FSM design. The design also implements pipeline to reduce Harware Idle time

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.

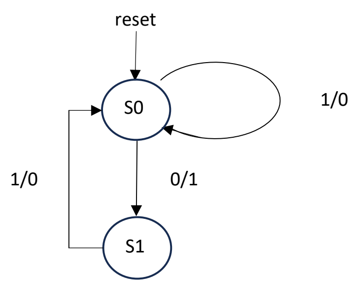
Block Diagram



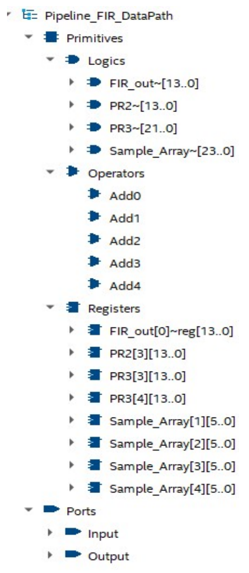
FSM (State Graph) for Controller

State Table

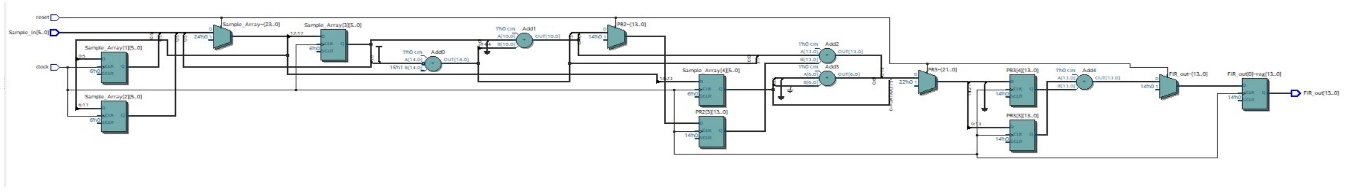
input	output	
reset	enable	state
1	0	S0
0	1	S1



Building Blocks of the Datapath - RTL Viewer



- Pipeline_FIR_DataPath
  - Primitives
    - Logics
      - FIR_out~[13..0]
      - PR2~[13..0]
      - PR3~[21..0]
      - Sample_Array~[23..0]
    - Operators
      - Add0
      - Add1
      - Add2
      - Add3
      - Add4
    - Registers
      - FIR_out[0]~reg[13..0]
      - PR2[3][13..0]
      - PR3[3][13..0]
      - PR3[4][13..0]
      - Sample_Array[1][5..0]
      - Sample_Array[2][5..0]
      - Sample_Array[3][5..0]
      - Sample_Array[4][5..0]
  - Ports
    - Input
    - Output



Design Verilog HDL Code -

Controller

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

```verilog
// ee417 Final Project L12A2
// Name: Ron Kalin / Lamin Jammeh, Date: 08-02-24  Group: Kalin/Jammeh
// Design: FIR filter chain of cascaded MACs, given 5-bit coefficients
// inputs are 6-bit positive values, output register is 14 parallel bits
// Project Description: Controller module drives the enable signal, which enables the
// computation of the FIR_MAC

module Pipeline_FIR_Controller  (
                                 input clock,
                                 input reset,
                                 output reg enable
                                );

// Control logic for the FIR filter
always @ (posedge clock or posedge reset)
begin
   if (reset)
      begin
         enable <= 0;
      end
      else
      begin
         enable <= 1;
      end
end

endmodule

```

Datapath

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

```verilog
/*---------------------------------------------------------------------------------------------
--------------
Name Lamin Jammeh / Ron Kalin
CLass: EE417 Summer 2024
FINAL PROJECT: Datapath
Group: Ron Kalin/ Lamin Jammeh
Project Description: Datapath module computesfilters input Sample by Multiplying and
Accumulating
---------------------------------------------------------------------------------------------
--------------*/

module Pipeline_FIR_DataPath  (FIR_out, Sample_in, clock, reset);

//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    = Sample_size + weight_size + 3; //log2(2^2 * 2^5 * (order+1))

//define output
output reg [word_size_out -1:0]  FIR_out;

//define inputs
input     [Sample_size -1:0]      Sample_in;
input                             clock, reset;

//define the filter coefficients
parameter   b0 = 5'd3;
parameter   b1 = 5'd7;
parameter   b2 = 5'd20;
parameter   b3 = 5'd7;
parameter   b4 = 5'd3;

reg          [Sample_size -1:0]   Sample_Array[1:FIR_order]; //5th coefficient multiplied by
Data_in
integer      k;

//define PR0 to PR3 as registers
reg       [word_size_out -1:0] PR0 [0:FIR_order];
reg       [word_size_out -1:0] PR1 [1:FIR_order];
reg       [word_size_out -1:0] PR2 [2:FIR_order];
reg       [word_size_out -1:0] PR3 [3:FIR_order];

//define the transition logic
always @ (posedge clock)
    if (reset == 1)
    /*----------------------------------------
    if reset is high do the following
    ****set all Pipeline registers to zero
    *************IR = 0      Input register
    *************PR[0:order-1] = 0    Pipeline register
    *************OR = 0      Output register
    ----------------------------------------*/
       begin
       //The input shift register
          for (k=1; k <= FIR_order; k = k + 1)
             Sample_Array[k] <= 0;

       //The pipeline register
          for (k = 0; k <= FIR_order; k = k + 1)
             PR0[k] <= 0;
       //The pipeline register
          for (k = 1; k <= FIR_order; k = k + 1)
             PR1[k] <= 0;
       //The pipeline register
          for (k = 2; k <= FIR_order; k = k + 1)
             PR2[k] <= 0;
       //The pipeline register
          for (k = 3; k <= FIR_order; k = k + 1)
             PR3[k] <= 0;
```

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

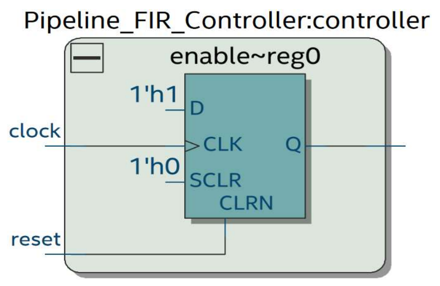
```verilog
        //The outpput register
            FIR_out  <= 0;
        end
    else
    /*---------------------------------------
    if reset is low do the following
    **************1 => move the Sample in into a cutset (Input register) to reduce idle time of the input
    **************2 => insert the PR at the input of the add and perform x[n] * b(n) and save in Pipeline register (PRn[n-1])
    **************3 => add all the PR registers at the input of the output register and save in the Output register
    ----------------------------------------*/
        begin
        //The input shift register
          Sample_Array[1] <= Sample_in;
          for (k = 2; k <= FIR_order; k = k + 1)
            Sample_Array[k] <= Sample_Array[k-1];

        //The pipeline register at PR0
           PR0[0] <= b0 * Sample_in;
           PR0[1] <= b1 * Sample_Array[1];
           PR0[2] <= b2 * Sample_Array[2];
           PR0[3] <= b3 * Sample_Array[3];
           PR0[4] <= b4 * Sample_Array[4];

        //The pipeline register at PR1
           PR1[1] <= b1 * Sample_Array[1] + PR0[1];
           PR1[2] <= b2 * Sample_Array[2];
           PR1[3] <= b3 * Sample_Array[3];
           PR1[4] <= b4 * Sample_Array[4];

        //The pipeline register at PR2
           PR2[2] <= b2 * Sample_Array[2] + PR1[2];
           PR2[3] <= b3 * Sample_Array[3];
           PR2[4] <= b4 * Sample_Array[4];

        //The pipeline register at PR3
           PR3[3] <= b3 * Sample_Array[3] + PR2[3];
           PR3[4] <= b4 * Sample_Array[4];

        //The outpput register
             FIR_out  <= PR3[3] + PR3[4];
        end
endmodule

```

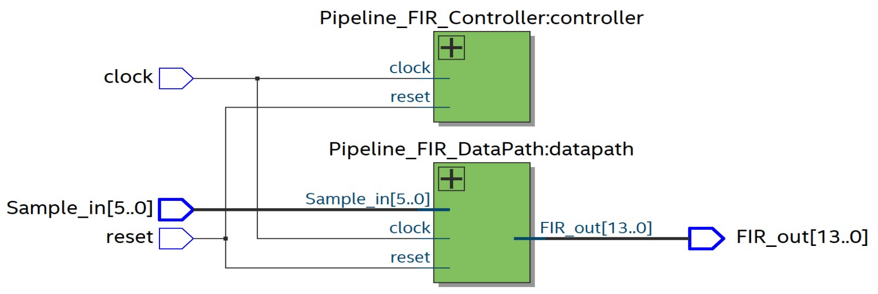
Top level module

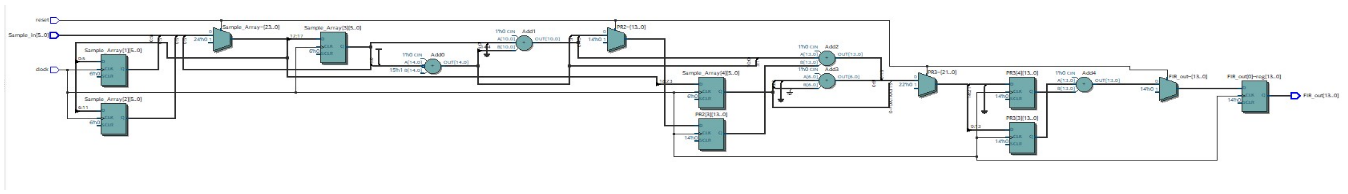
Date: August 02, 2024 Pipeline_FIR_MAC.v Project: Pipeline_FIR_DataPath

```verilog
// Name Lamin Jammeh / Ron Kalin
// CLass: EE417 Summer 2024
// FINAL PROJECT: FIR MAC module
// Group: Ron Kalin/ Lamin Jammeh
// Summary: Module combines Datapath with controller to form a FIR_MAC

module Pipeline_FIR_MAC  (FIR_out, Sample_in, clock, reset);

// Define the parameter sets for the design
parameter FIR_order        = 4;
parameter Sample_size      = 6;                        // Max sample value 2^n= 63
parameter weight_size      = 5;                        // Max value may be 31
parameter word_size_out    = Sample_size + weight_size + 3;   // log2(2^2 * 2^5 * (order+1))

//define the outputs
output [word_size_out - 1:0] FIR_out;

//define the inputs
input   [Sample_size - 1:0]    Sample_in;
input                          clock, reset;

// Internal signals
wire enable;

// Instantiate the DataPath module
Pipeline_FIR_DataPath #(FIR_order, Sample_size, weight_size, word_size_out) datapath (
    .FIR_out(FIR_out),
    .Sample_in(Sample_in),
    .clock(clock),
    .reset(reset)
);

// Instantiate the Controller module
Pipeline_FIR_Controller  controller (
    .clock(clock),
    .reset(reset),
    .enable(enable)
);

endmodule

```

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

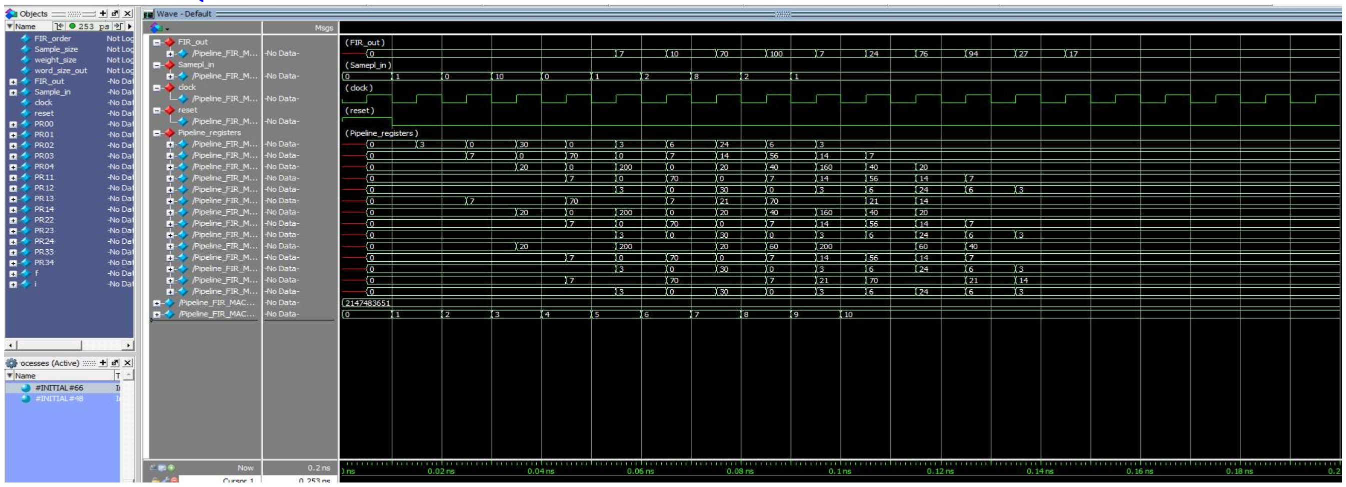


Testbench Verilog HDL Code

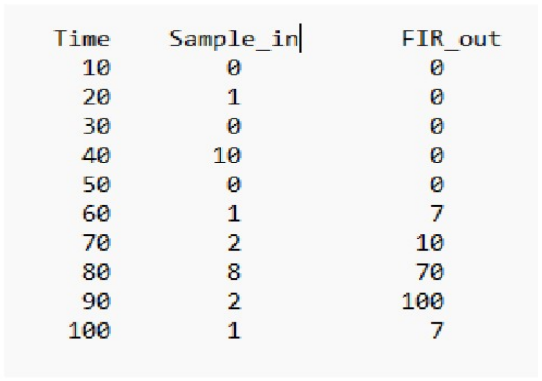
```
/*---------------------------------------------------------------------------------------------
---------
Name Ron Kalin / Lamin Jammeh
CLass: EE417 Summer 2024
FINAL PROJECT: FIR MAC module
Group: Ron Kalin/ Lamin Jammeh
Project Description: testbench for the FIR_MAC with Pipelining
---------------------------------------------------------------------------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

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");
```

```
        $fwrite(f, "\t\tTime\tSample_in\tFIR_out\n");

        // Apply the input samples and log the output
        for (i = 0; i < 10; i = i + 1)
           begin
              case (i)
                  0:  Sample_in = 0;
                  1:  Sample_in = 1;
                  2:  Sample_in = 0;
                  3:  Sample_in = 10;
                  4:  Sample_in = 0;
                  5:  Sample_in = 1;
                  6:  Sample_in = 2;
                  7:  Sample_in = 8;
                  8:  Sample_in = 2;
                  9:  Sample_in = 1;
                  10: Sample_in = 0;
                  11: Sample_in = 63;
                  12: Sample_in = 0;
                  default: Sample_in = 0;
              endcase
              #10; // Wait for the output to settle
              $fwrite(f, "%d\t   %d\t      %d\n", $time, Sample_in, FIR_out);
          end

          $fclose(f);
          #100 $stop;
      end

  endmodule
```

RTL Simulation -from Questa



Testbench results from the text file

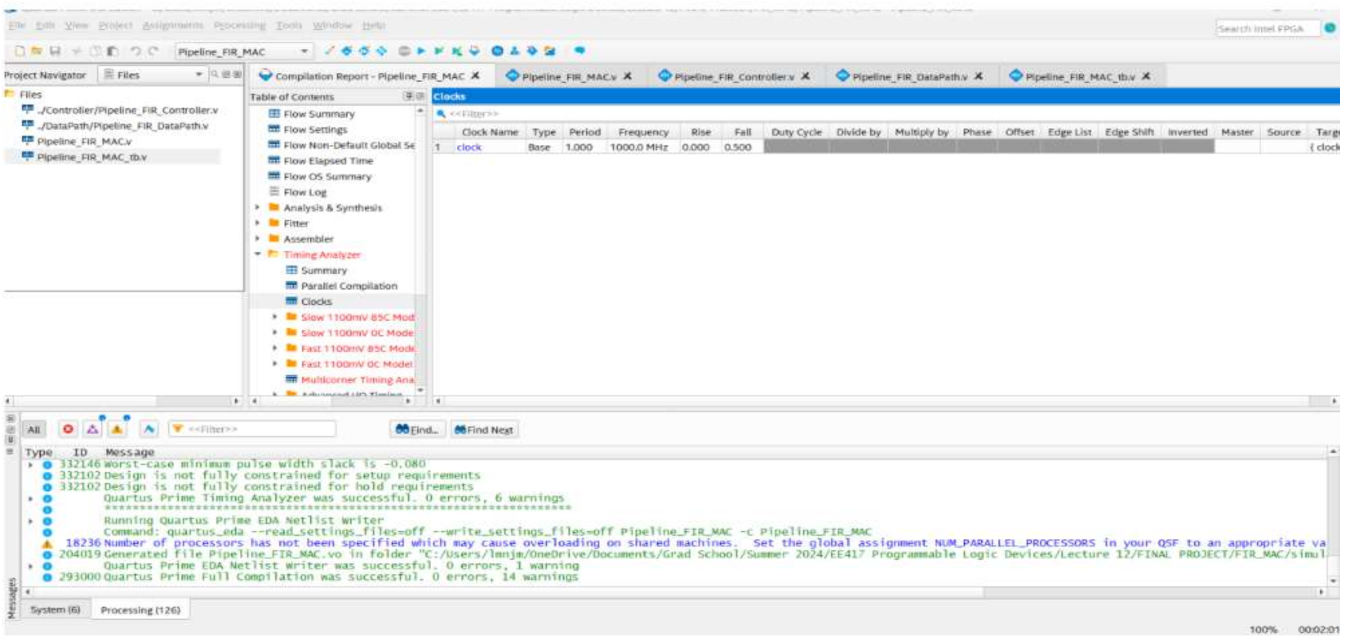


```
Time      Sample_in        FIR_out
  10          0               0
  20          1               0
  30          0               0
  40         10               0
  50          0               0
  60          1               7
  70          2              10
  80          8              70
  90 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

Timing Analysis Reports

Timing Analysis for the clock signal

@ default time when clock period is 1ns

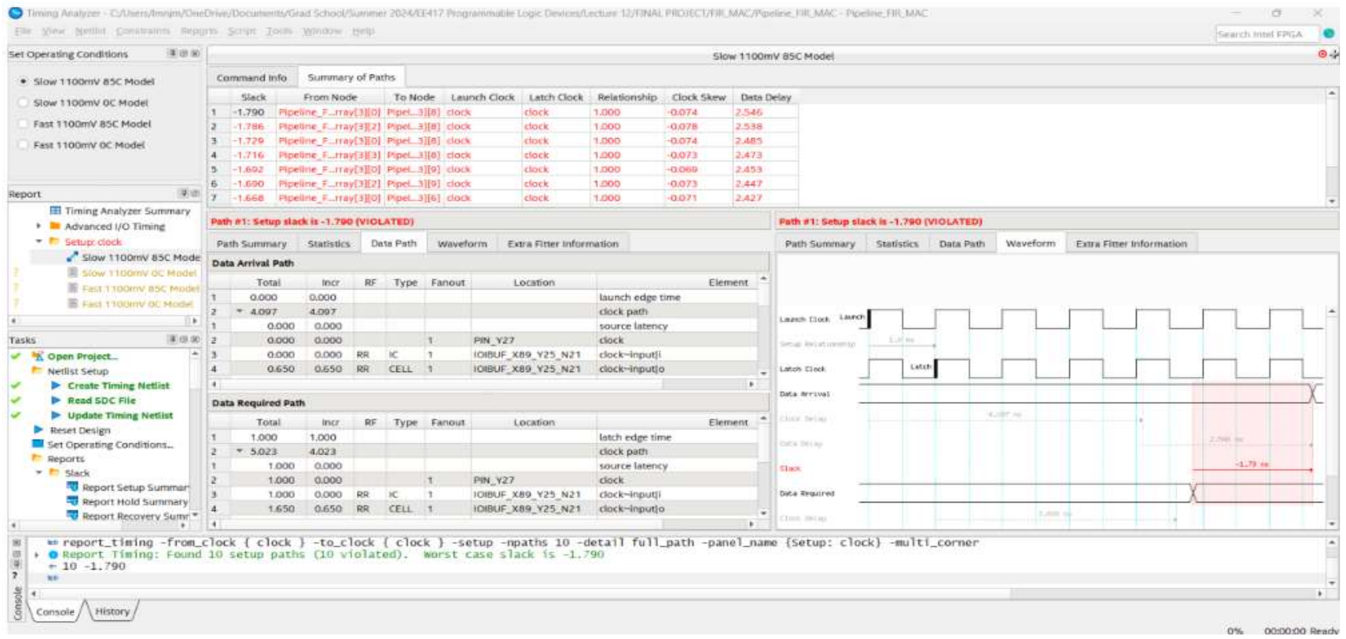


Clocks

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by	Phase	Offset	Edge List	Edge Shift	Inverted	Master	Source	Targ
1	clock	Base	1.000	1000.0 MHz	0.000	0.500											{ clock

```
Type  ID     Message
      332146 Worst-case minimum pulse width slack is -0.080
      332102 Design is not fully constrained for setup requirements
      332102 Design is not fully constrained for hold requirements
             Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
             ***************************************************************
             Running Quartus Prime EDA Netlist Writer
             Command: quartus_eda --read_settings_files=off --write_settings_files=off Pipeline_FIR_MAC -c Pipeline_FIR_MAC
      18236  Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate va
      204019 Generated file Pipeline_FIR_MAC.vo in folder "C:/Users/lmnjm/OneDrive/Documents/Grad School/Summer 2024/EE417 Programmable Logic Devices/Lecture 12/FINAL PROJECT/FIR_MAC/simul
             Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
      293000 Quartus Prime Full Compilation was successful. 0 errors, 14 warnings
```

Time report at a slack of -1.790



Slow 1100mV 85C Model — Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-1.790	Pipeline_F...rray[3][0]	Pipel...3][8]	clock	clock	1.000	-0.074	2.546
2	-1.786	Pipeline_F...rray[3][2]	Pipel...3][8]	clock	clock	1.000	-0.078	2.538
3	-1.729	Pipeline_F...rray[3][0]	Pipel...3][8]	clock	clock	1.000	-0.074	2.485
4	-1.716	Pipeline_F...rray[3][3]	Pipel...3][8]	clock	clock	1.000	-0.073	2.473
5	-1.692	Pipeline_F...rray[3][0]	Pipel...3][9]	clock	clock	1.000	-0.069	2.453
6	-1.690	Pipeline_F...rray[3][2]	Pipel...3][9]	clock	clock	1.000	-0.073	2.447
7	-1.668	Pipeline_F...rray[3][0]	Pipel...3][6]	clock	clock	1.000	-0.071	2.427

Path #1: Setup slack is -1.790 (VIOLATED)

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	4.097	4.097					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_Y27	clock
3	0.000	0.000	RR	IC	1	IOIBUF_X89_Y25_N21	clock~input|i
4	0.650	0.650	RR	CELL	1	IOIBUF_X89_Y25_N21	clock~input|o

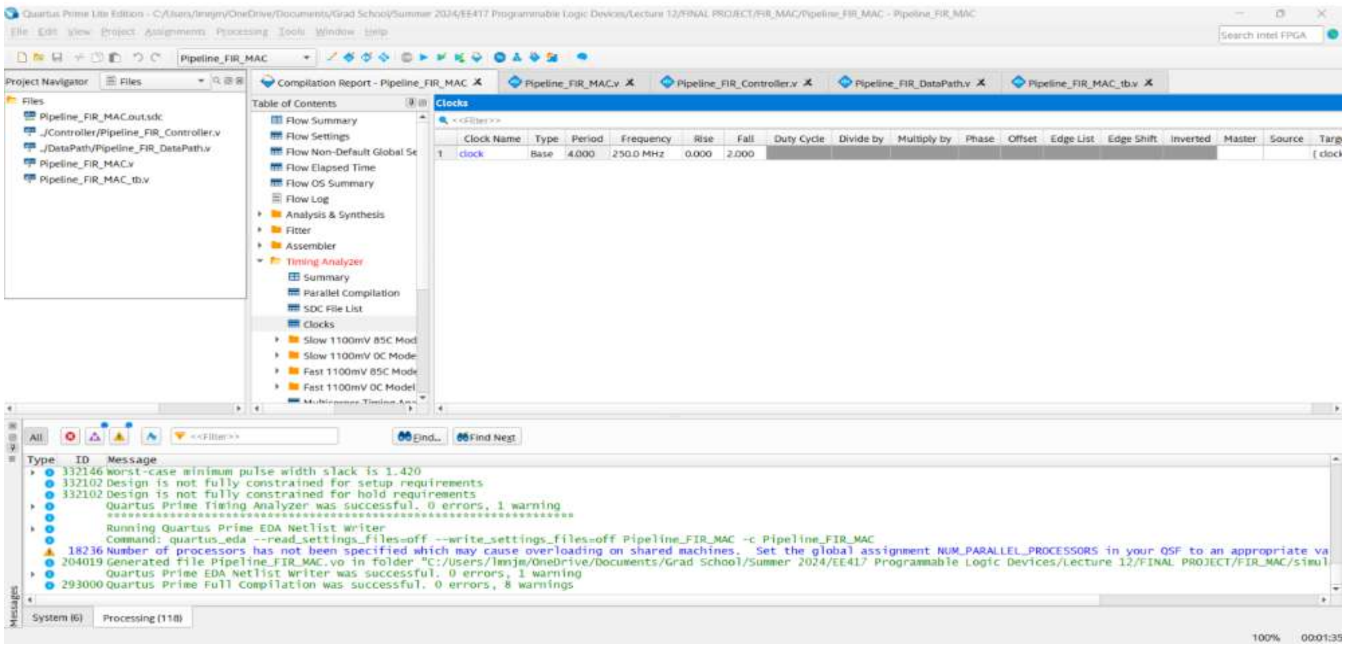
Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	1.000	1.000					latch edge time
2	5.023	4.023					clock path
1	1.000	0.000					source latency
2	1.000	0.000			1	PIN_Y27	clock
3	1.000	0.000	RR	IC	1	IOIBUF_X89_Y25_N21	clock~input|i
4	1.650	0.650	RR	CELL	1	IOIBUF_X89_Y25_N21	clock~input|o

```
report_timing -from_clock { clock } -to_clock { clock } -setup -npaths 10 -detail full_path -panel_name {Setup: clock} -multi_corner
Report Timing: Found 10 setup paths (10 violated).  Worst case slack is -1.790
← 10 -1.790
```

Timing Analysis Reports (cont'd)

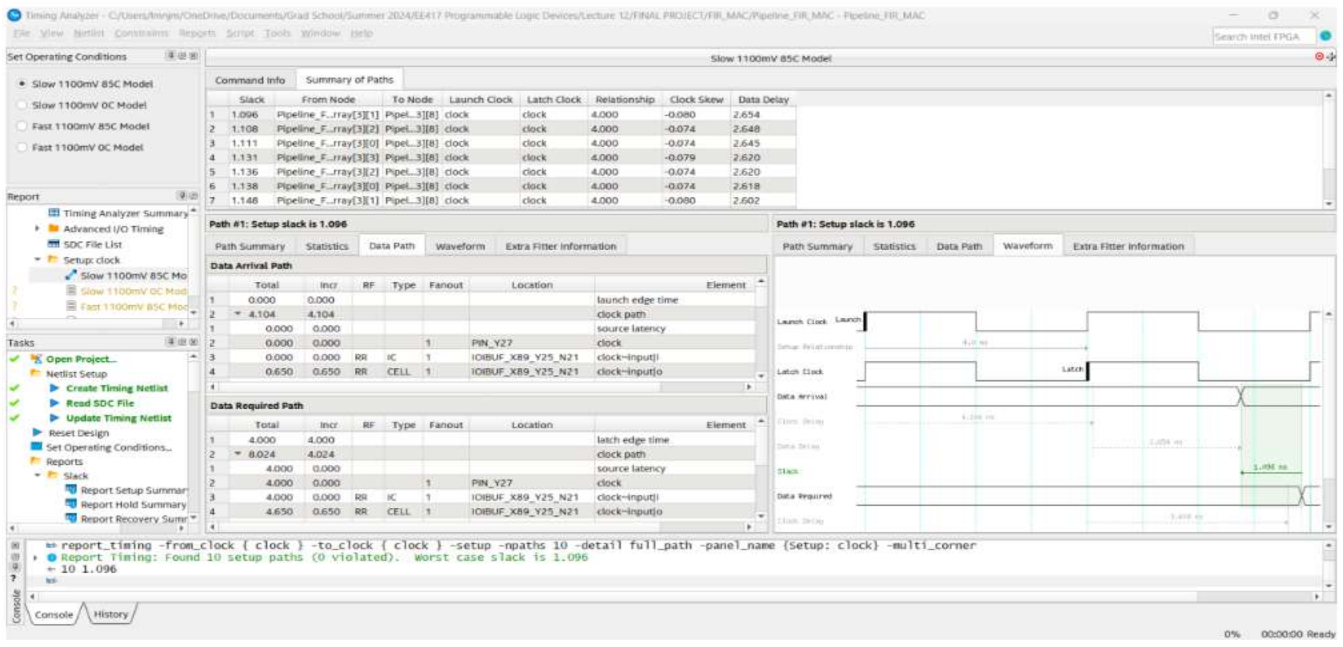
@ default time when clock period is 4ns



	Clock Name	Type	Period	Frequency	Rise	Fall
1	clock	Base	4.000	250.0 MHz	0.000	2.000

```
Type  ID    Message
      332146 Worst-case minimum pulse width slack is 1.420
      332102 Design is not fully constrained for setup requirements
      332102 Design is not fully constrained for hold requirements
             Quartus Prime Timing Analyzer was successful. 0 errors, 1 warning
             ***********************************************************************
             Running Quartus Prime EDA Netlist Writer
             Command: quartus_eda --read_settings_files=off --write_settings_files=off Pipeline_FIR_MAC -c Pipeline_FIR_MAC
       18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate va
      204019 Generated file Pipeline_FIR_MAC.vo in folder "C:/Users/lmnjm/OneDrive/Documents/Grad School/Summer 2024/EE417 Programmable Logic Devices/Lecture 12/FINAL PROJECT/FIR_MAC/simul
             Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
      293000 Quartus Prime Full Compilation was successful. 0 errors, 8 warnings
```

Time report at a slack of 1.096



Slow 1100mV 85C Model — Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	1.096	Pipeline_F...rray[3][1]	Pipel...3][8]	clock	clock	4.000	-0.080	2.654
2	1.108	Pipeline_F...rray[3][2]	Pipel...3][8]	clock	clock	4.000	-0.074	2.648
3	1.111	Pipeline_F...rray[3][0]	Pipel...3][8]	clock	clock	4.000	-0.074	2.645
4	1.131	Pipeline_F...rray[3][3]	Pipel...3][8]	clock	clock	4.000	-0.079	2.620
5	1.136	Pipeline_F...rray[3][2]	Pipel...3][8]	clock	clock	4.000	-0.074	2.620
6	1.138	Pipeline_F...rray[3][0]	Pipel...3][8]	clock	clock	4.000	-0.074	2.618
7	1.146	Pipeline_F...rray[3][1]	Pipel...3][8]	clock	clock	4.000	-0.080	2.602

Path #1: Setup slack is 1.096

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000					launch edge time
2	4.104	4.104					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_Y27	clock
3	0.000	0.000	RR	IC	1	IOIBUF_X89_Y25_N21	clock~input|i
4	0.650	0.650	RR	CELL	1	IOIBUF_X89_Y25_N21	clock~input|o

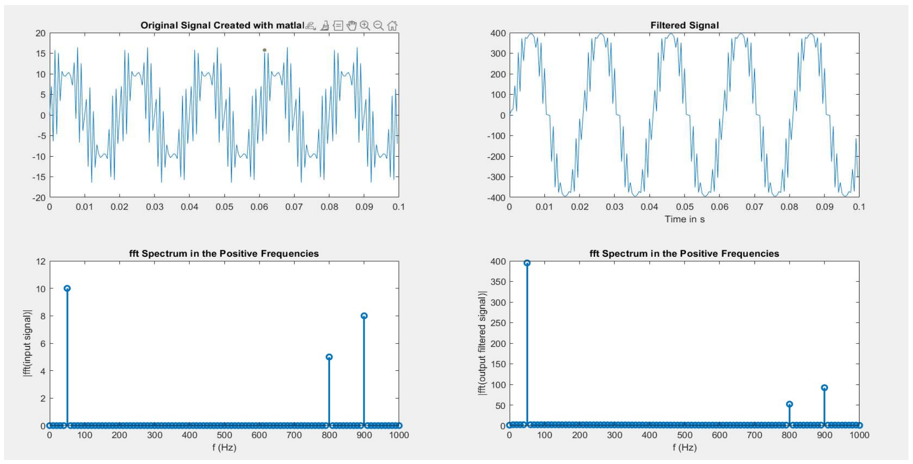
Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	4.000	4.000					latch edge time
2	8.024	4.024					clock path
1	4.000	0.000					source latency
2	4.000	0.000			1	PIN_Y27	clock
3	4.000	0.000	RR	IC	1	IOIBUF_X89_Y25_N21	clock~input|i
4	4.650	0.650	RR	CELL	1	IOIBUF_X89_Y25_N21	clock~input|o

```
report_timing -from_clock { clock } -to_clock { clock } -setup -npaths 10 -detail full_path -panel_name {Setup: clock} -multi_corner
Report Timing: Found 10 setup paths (0 violated).  Worst case slack is 1.096
10 1.096
```

```
%↙
---------------------------------------------------------------------------------------↙
--------------
% Name Lamin Jammeh
% CLass: EE417 Summer 2024
% Lesson 10 HW Question 3
% Group: Ron Kalin/ Lamin Jammeh
% Project Description: The verilog coeff are used for creating the filter
%↙
---------------------------------------------------------------------------------------↙
--------------

% Defining the parameters of the synthetic input signal:
% Creating a sine signal

fs   = 2000;     % the sampling frequency = 2KHz
fmax = fs/2;     % Maximum fundamental frequency that can be sampled by fs

f1 = 50;       % frequency f1 is 50Hz
f2 = 800;      % frequency f2 is 800Hz
f3 = 900;      % frequency f3 is 900Hz

A1 = 10;       % Amplitude of the sine wave s1
A2 = 5;        % Amplitude of the sine wave s2
A3 = 8;        % Amplitide of the sine wave s3

L = 200;       % the number of samples

t = (0:L-1)/fs;           % Generating the time vector for L samples

s = A1*sin(2*pi*t*f1) + A2*sin(2*pi*t*f2) + A3*sin(2*pi*t*f3);

%----------------------------------------------------------------------------
% Defining the filter parameters:
% Use the filter coeff from verilog and create Sampling signal in matlab

fc = 400;      % the filter cutoff frequency is 400Hz
Wn = fc/fmax;  % Wn = fc/fmax
order = 4;     % the order of the filter

%coeff = fir1(order,Wn,'low');   % low pass filter
%coeff = fir1(order,Wn,'high');   % high pass filter
coeff = [3, 7, 20, 7, 3];          % same coefficient used in verilog code

%----------------------------------------------------------------------------
%              Filtering the signal using 3 approaches
%----------------------------------------------------------------------------
% 1. Filtering the signal in MATLAB using the 'filter' function:
```

```
output_signal = filter(coeff,1,s);

% Plotting the input signal & the filtered signals in the time domain

subplot(2,2,1)
plot(t,s)
title("Original Signal Created with matlab")
subplot(2,2,2)
plot(t,output_signal)
title("Filtered Signal")
xlabel("Time in s")

%-------------------------------------------------------------------------
% Creating the fft for the signals:

f = fs/L*(0:(L/2));

Y = fft(s);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);

Y_out = fft(output_signal);
P2_out = abs(Y_out/L);
P1_out = P2_out(1:L/2+1);
P1_out(2:end-1) = 2*P1_out(2:end-1);



subplot(2,2,3)
stem(f,P1,"LineWidth",2)
title("fft Spectrum in the Positive Frequencies")
xlabel("f (Hz)")
ylabel("|fft(input signal)|")

subplot(2,2,4)
stem(f,P1_out,"LineWidth",2)
title("fft Spectrum in the Positive Frequencies")
xlabel("f (Hz)")
ylabel("|fft(output filtered signal)|")


%--------------------------------------------------------------------------
```



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]https://ieeexplore.ieee.org/document/7208065
[2] Ciletti, Michael D.. Advanced Digital Design with the Verilog HDL (Section 9.5). Pearson Education. Kindle Edition.
[3] Chttps://community.intel.com/t5/FPGA-Wiki/Timing-Constraints/ta-p/735562