```verilog
// ee417 lesson 10 Assignment 2 L10A2 & A3
// Name: Ron Kalin, Date: 07-18-24  Group: Kalin/Jammeh
// Design: FIR filter chain of cascaded MACs, given 4-bit coefficients
// inputs are 4-bit positive values, output register is 11 parallel bits
// top level module
module Pipeline_FIR (FIR_pipeline_A, FIR_pipeline_B, FIR_pipeline_C,
                     FIR_MAC, Sample_in, clock, reset);

parameter FIR_order    = 6;  // order of the filter
parameter sample_size  = 4;  // word size of input samples max 15
parameter weight_size  = 5;  // component of word_size_out max 31
parameter word_size_out = sample_size + weight_size + 3; // max possible output
15*31*(6+1)= 12digits

output  [word_size_out -1: 0] FIR_MAC;       //declare outputs
output  [word_size_out -1: 0] FIR_pipeline_A;
output  [word_size_out -1: 0] FIR_pipeline_B;
output  [word_size_out -1: 0] FIR_pipeline_C;

input   [sample_size -1: 0] Sample_in;    //declare inputs
input                       clock, reset;

wire    [word_size_out -1: 0] FIR_assign; //internal wire
//instantiate submodules
noPipeline_FIR M1 ( FIR_MAC, FIR_assign, Sample_in, clock, reset );
Pipeline_FIR_a M2 ( FIR_pipeline_A, Sample_in, clock, reset ); //pipe @ mult out
Pipeline_FIR_b M3 ( FIR_pipeline_B, Sample_in, clock, reset ); //pipe @ every sum in
Pipeline_FIR_c M4 ( FIR_pipeline_C, Sample_in, clock, reset ); //pipe @ every other sum in

endmodule

//multiply and accumulate MAC
module MAC (Acc_out, Sample_in, b, Acc_in );
parameter sample_size  = 4;  // word size of input samples max 15
parameter weight_size  = 5;  // component of word_size_out max 31
parameter word_size_out = sample_size + weight_size + 3; // max possible output
15*31*(6+1)=3255, 12 digits
output [word_size_out-1: 0] Acc_out;
input  [sample_size  -1: 0] Sample_in, b, Acc_in;

assign Acc_out = (Sample_in * b) + Acc_in;
endmodule

//FIR with no pipeline using combinational logic
module noPipeline_FIR ( FIR_out_MAC, FIR_out_assign, Sample_in, clock, reset );

parameter FIR_order    = 6;  // order of the filter
parameter sample_size  = 4;  // word size of input samples max 15
parameter weight_size  = 5;  // component of word_size_out max 31
parameter word_size_out = sample_size + weight_size + 3; // max possible output
15*31*(6+1)=3255, 12 digits

output reg [word_size_out -1: 0] FIR_out_MAC;       //declare outputs
output reg [word_size_out -1: 0] FIR_out_assign;

input  [sample_size -1: 0] Sample_in;    //declare inputs
input                      clock, reset;

//internal wires
wire   [word_size_out -1: 0] Acc0, Acc1, Acc2, Acc3, Acc4, Acc5, Acc6;
wire   [word_size_out -1: 0] comb_out;

//filter coefficients given
//b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2
parameter b0 = 4'd2;
parameter b1 = 4'd5;
parameter b2 = 4'd9;
parameter b3 = 4'd14;
parameter b4 = 4'd9;
parameter b5 = 4'd5;
parameter b6 = 4'd2;

reg [sample_size -1: 0] Sample_Array [1: FIR_order]; //7th coefficient multipled by Data_in
integer k;

MAC M0 ( Acc0, Sample_in,        b0, 0    ); //combinational logic
MAC M1 ( Acc1, Sample_Array [1], b1, Acc0 ); //combinational logic
MAC M2 ( Acc2, Sample_Array [2], b2, Acc1 ); //combinational logic
MAC M3 ( Acc3, Sample_Array [3], b3, Acc2 ); //combinational logic
MAC M4 ( Acc4, Sample_Array [4], b4, Acc3 ); //combinational logic
MAC M5 ( Acc5, Sample_Array [5], b5, Acc4 ); //combinational logic
MAC M6 ( Acc6, Sample_Array [6], b6, Acc5 ); //combinational logic

//alternate way to create combinational logic
assign comb_out = b0 * Sample_in
                + b1 * Sample_Array[1]
                + b2 * Sample_Array[2]
                + b3 * Sample_Array[3]
                + b4 * Sample_Array[4]
                + b5 * Sample_Array[5]
                + b6 * Sample_Array[6];

always @ (posedge clock)
   if (reset == 1) begin
     for (k = 1; k <= FIR_order; k=k+1)
          Sample_Array[k] <= 0; //set registers to zero
          FIR_out_MAC     <= 0;
          FIR_out_assign  <= 0;
   end
   else begin
          Sample_Array [1]  <= Sample_in;
            for (k = 2; k   <= FIR_order; k= k+1)
             Sample_Array[k]<= Sample_Array[k-1];
            FIR_out_assign  <= comb_out;
            FIR_out_MAC     <= Acc6;
   end
endmodule

// pipeline structure a: FIR filter w/ pipeline registers placed
// at multiplier outputs
module Pipeline_FIR_a ( FIR_out_pipeline, Sample_in, clock, reset);

parameter FIR_order    = 6;  // order of the filter
parameter sample_size  = 4;  // word size of input samples max 15
parameter weight_size  = 5;  // component of word_size_out max 31
parameter word_size_out = sample_size + weight_size + 3; // max possible output
15*31*(6+1)= 12digits
parameter product_size = sample_size + weight_size;

output reg [word_size_out -1: 0] FIR_out_pipeline; //declare outputs

input   [sample_size -1: 0] Sample_in;    //declare inputs
input                       clock, reset;

//filter coefficients given
//b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2
parameter b0 = 4'd2;
parameter b1 = 4'd5;
parameter b2 = 4'd9;
parameter b3 = 4'd14;
parameter b4 = 4'd9;
parameter b5 = 4'd5;
parameter b6 = 4'd2;

reg [sample_size -1: 0] Sample_Array [1: FIR_order]; //7th coefficient multipled by Data_in
integer k;

reg [product_size -1: 0] PR[0 : FIR_order];  // array format

always @ (posedge clock)
```

```
    if (reset == 1) begin
       // input shift register
       for (k = 1; k <= FIR_order; k = k +1)  //saves code lines
             Sample_Array[k] <= 0;
       // pipeline register
       for (k = 0; k <= FIR_order; k = k +1)  //saves code lines
             PR[k]   <= 0;
       // output register
       FIR_out_pipeline <= 0;
       end

   else begin
       // input shift register
       Sample_Array [1] <= Sample_in;
         for (k =2; k   <= FIR_order; k=k+1)
           Sample_Array [k] <= Sample_Array [k-1];
       // pipeline register
       PR[0] <= b0 * Sample_in;       // find products
       PR[1] <= b1 * Sample_Array[1];
       PR[2] <= b2 * Sample_Array[2];
       PR[3] <= b3 * Sample_Array[3];
       PR[4] <= b4 * Sample_Array[4];
       PR[5] <= b5 * Sample_Array[5];
       PR[6] <= b6 * Sample_Array[6];
       // output register, sum products
       FIR_out_pipeline <= PR[0] + PR[1] + PR[2] + PR[3] + PR[4] + PR[5] + PR[6];
       end
endmodule


// Alternative pipeline structure b: FIR filter w/ pipeline registers placed
// at adder inputs
module Pipeline_FIR_b ( FIR_out, Sample_in, clock, reset);

parameter FIR_order    = 6;  // order of the filter
parameter sample_size  = 4;  // word size of input samples max 15
parameter weight_size  = 5;  // component of word_size_out max 31
parameter word_size_out = sample_size + weight_size + 3; // max possible output 15*31*(6+1)= 12digits
parameter product_size = sample_size + weight_size;

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

input   [sample_size -1: 0] Sample_in;    //declare inputs
input                       clock, reset;

//filter coefficients given
//b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2
parameter b0 = 4'd2;
parameter b1 = 4'd5;
parameter b2 = 4'd9;
parameter b3 = 4'd14;
parameter b4 = 4'd9;
parameter b5 = 4'd5;
parameter b6 = 4'd2;

reg [sample_size -1: 0] Sample_Array [1: FIR_order]; //7th coefficient multipled by Data_in
integer k;

reg [product_size -1: 0] PR0 [0 : FIR_order];  // array format
reg [product_size -1: 0] PR1 [0 : FIR_order];
reg [product_size -1: 0] PR2 [0 : FIR_order];
reg [product_size -1: 0] PR3 [0 : FIR_order];
reg [product_size -1: 0] PR4 [0 : FIR_order];
reg [product_size -1: 0] PR5 [0 : FIR_order];

always @ (posedge clock)
    if (reset == 1) begin
       // input shift register
       for (k = 1; k <= FIR_order; k = k +1)
```

```
                Sample_Array[k] <= 0;
        // pipeline register PR0
        for (k = 0; k <= FIR_order; k = k +1)
                PR0[k]   <= 0;
        // pipeline register PR1
        for (k = 0; k <= FIR_order; k = k +1)
                PR1[k]   <= 0;
        // pipeline register PR2
        for (k = 0; k <= FIR_order; k = k +1)
                PR2[k]   <= 0;
        // pipeline register PR3
        for (k = 0; k <= FIR_order; k = k +1)
                PR3[k]   <= 0;
        // pipeline register PR4
        for (k = 0; k <= FIR_order; k = k +1)
                PR4[k]   <= 0;
        // pipeline register PR5
        for (k = 0; k <= FIR_order; k = k +1)
                PR5[k]   <= 0;
        // output register
        FIR_out <= 0;
        end

    else begin
        // input shift register
        Sample_Array [1] <= Sample_in;
          for (k =2; k      <= FIR_order; k=k+1)
            Sample_Array[k] <= Sample_Array[k-1];
        // pipeline register PR0
        PR0[0] <= b0 * Sample_in;       // find products
        PR0[1] <= b1 * Sample_Array[1];
        PR0[2] <= b2 * Sample_Array[2];
        PR0[3] <= b3 * Sample_Array[3];
        PR0[4] <= b4 * Sample_Array[4];
        PR0[5] <= b5 * Sample_Array[5];
        PR0[6] <= b6 * Sample_Array[6];

        // pipeline register PR1
        PR1[1] <= PR0[0] + PR0[1];
        PR1[2] <= PR0[2];
        PR1[3] <= PR0[3];
        PR1[4] <= PR0[4];
        PR1[5] <= PR0[5];
        PR1[6] <= PR0[6];

        // pipeline register PR2
        PR2[2] <= PR1[1] + PR1[2];
        PR2[3] <= PR1[3];
        PR2[4] <= PR1[4];
        PR2[5] <= PR1[5];
        PR2[6] <= PR1[6];

        // pipeline register PR3
        PR3[3] <= PR2[2] + PR2[3];
        PR3[4] <= PR2[4];
        PR3[5] <= PR2[5];
        PR3[6] <= PR2[6];

        // pipeline register PR4
        PR4[4] <= PR3[3] + PR3[4];
        PR4[5] <= PR3[5];
        PR4[6] <= PR3[6];

        // pipeline register PR5
        PR5[5] <= PR4[4] + PR4[5];
        PR5[6] <= PR4[6];

        // output register, sum products
        FIR_out<= PR5[5] + PR5[6];
        end
```

```
endmodule

// Alternative pipeline structure c: FIR filter w/ pipeline registers placed
// at every other adder input
module Pipeline_FIR_c ( FIR_out, Sample_in, clock, reset);

parameter FIR_order    = 6;  // order of the filter
parameter sample_size  = 4;  // word size of input samples max 15
parameter weight_size  = 5;  // component of word_size_out max 31
parameter word_size_out = sample_size + weight_size + 3; // max possible output
15*31*(6+1)= 12digits
parameter product_size = sample_size + weight_size;

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

input   [sample_size -1: 0] Sample_in;    //declare inputs
input                       clock, reset;

//filter coefficients given
//b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2
parameter b0 = 4'd2;
parameter b1 = 4'd5;
parameter b2 = 4'd9;
parameter b3 = 4'd14;
parameter b4 = 4'd9;
parameter b5 = 4'd5;
parameter b6 = 4'd2;

reg [sample_size -1: 0] Sample_Array [1: FIR_order]; //7th coefficient multipled by Data_in
integer k;

reg [product_size -1: 0] PR0 [0 : FIR_order];  // array format
reg [product_size -1: 0] PR1 [0 : FIR_order];
reg [product_size -1: 0] PR2 [0 : FIR_order];
/*reg [product_size -1: 0] PR3 [0 : FIR_order];
reg [product_size -1: 0] PR4 [0 : FIR_order];
reg [product_size -1: 0] PR5 [0 : FIR_order];*/

always @ (posedge clock)
    if (reset == 1) begin
       // input shift register
       for (k = 1; k <= FIR_order; k = k +1)
             Sample_Array[k] <= 0;
       // pipeline register PR0
       for (k = 0; k <= FIR_order; k = k +1)
             PR0[k]    <= 0;
       // pipeline register PR1
       for (k = 0; k <= FIR_order; k = k +1)
             PR1[k]    <= 0;
       // pipeline register PR2
       for (k = 0; k <= FIR_order; k = k +1)
             PR2[k]    <= 0;
       /*// pipeline register PR3
       for (k = 0; k <= FIR_order; k = k +1)
             PR3[k]    <= 0;
       // pipeline register PR4
       for (k = 0; k <= FIR_order; k = k +1)
             PR4[k]    <= 0;
       // pipeline register PR5
       for (k = 0; k <= FIR_order; k = k +1)
             PR5[k]    <= 0;*/
       // output register
       FIR_out <= 0;
       end

   else begin
       // input shift register
       Sample_Array [1] <= Sample_in;
         for (k =2; k      <= FIR_order; k=k+1)
           Sample_Array[k] <= Sample_Array[k-1];
```

```
        // pipeline register PR0, array of 7 products
        PR0[0] <= b0 * Sample_in;       // find products
        PR0[1] <= b1 * Sample_Array[1];
        PR0[2] <= b2 * Sample_Array[2];
        PR0[3] <= b3 * Sample_Array[3];
        PR0[4] <= b4 * Sample_Array[4];
        PR0[5] <= b5 * Sample_Array[5];
        PR0[6] <= b6 * Sample_Array[6];

        // pipeline register PR1, array of 5 entries
        PR1[2] <= PR0[0] + PR0[1] + PR0[2]; //+ PR1[1] + PR1[2];
        PR1[3] <= PR0[3];
        PR1[4] <= PR0[4];
        PR1[5] <= PR0[5];
        PR1[6] <= PR0[6];

        // pipeline register PR2, array of 3 entries
        PR2[4] <= PR1[2] + PR1[3] + PR1[4];
        PR2[5] <= PR1[5];
        PR2[6] <= PR1[6];

        // output register, sum products
        FIR_out<= PR2[4] + PR2[5] + PR2[6];
        end
endmodule

```