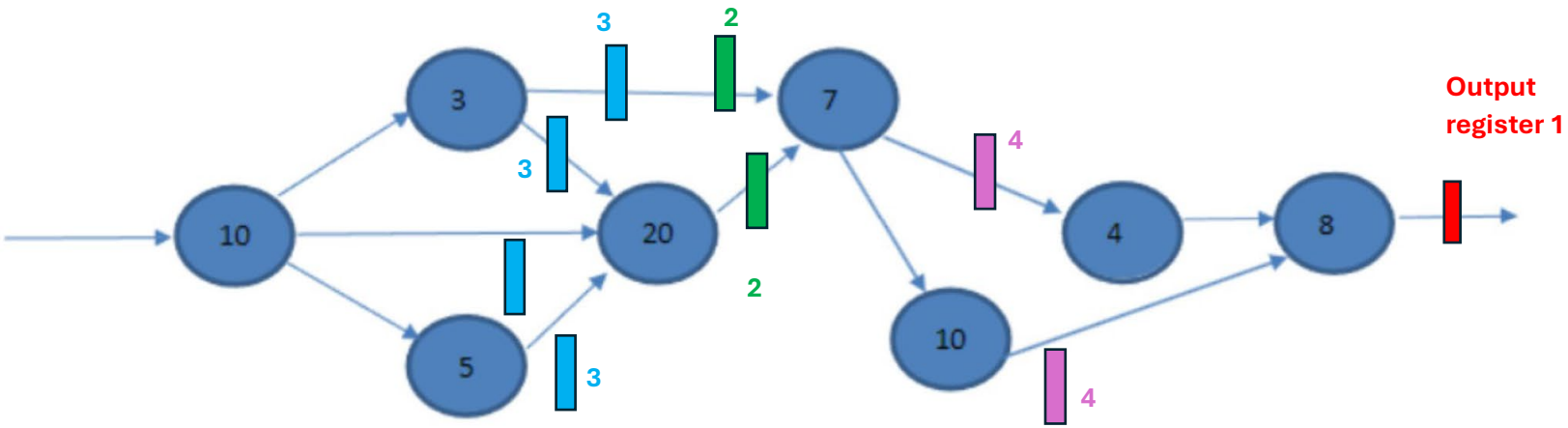
Lesson 10: Pipelining

Question 1:

The nodes of the DFG (Data Flow Graph) shown in figure has been annotated with propagation delays. Find the optimal placement of pipeline registers in the circuit. Make a list with the number of cut-sets and their locations and the corresponding latencies and throughput. What is the maximum clock frequency that can be used?

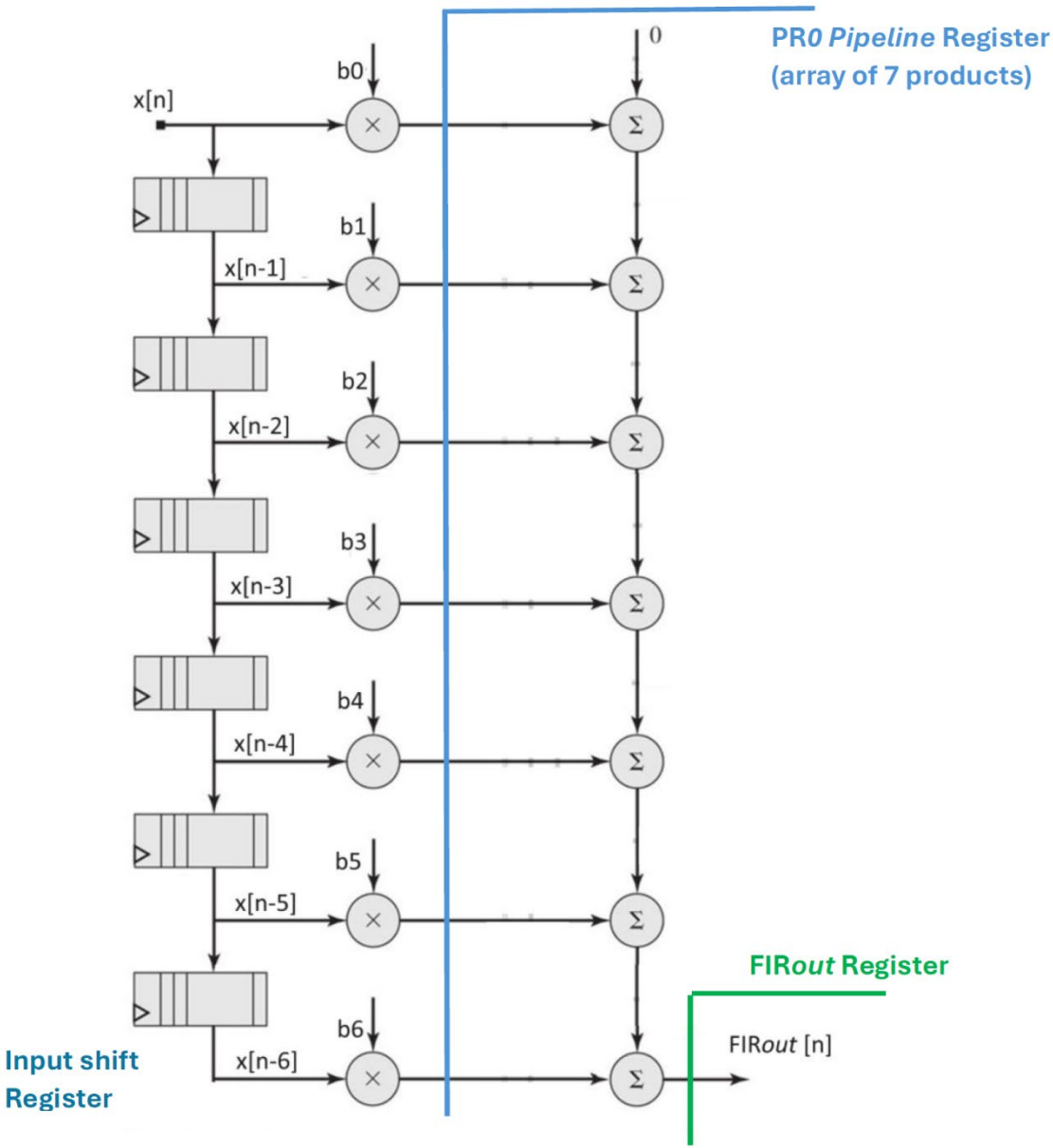


Cutset	Longest propagation delay	latency	Maximum clock & throughput
output register 1	60ns	60ns	16.667 MHz
Pipeline register 2	35ns	70ns	28.57MHz
Pipeline register 3	25ns	75ns	40MHz
Pipeline Register 4	20ns	80ns	50MHz

The maximum frequency that can be implemented is 50MHz.

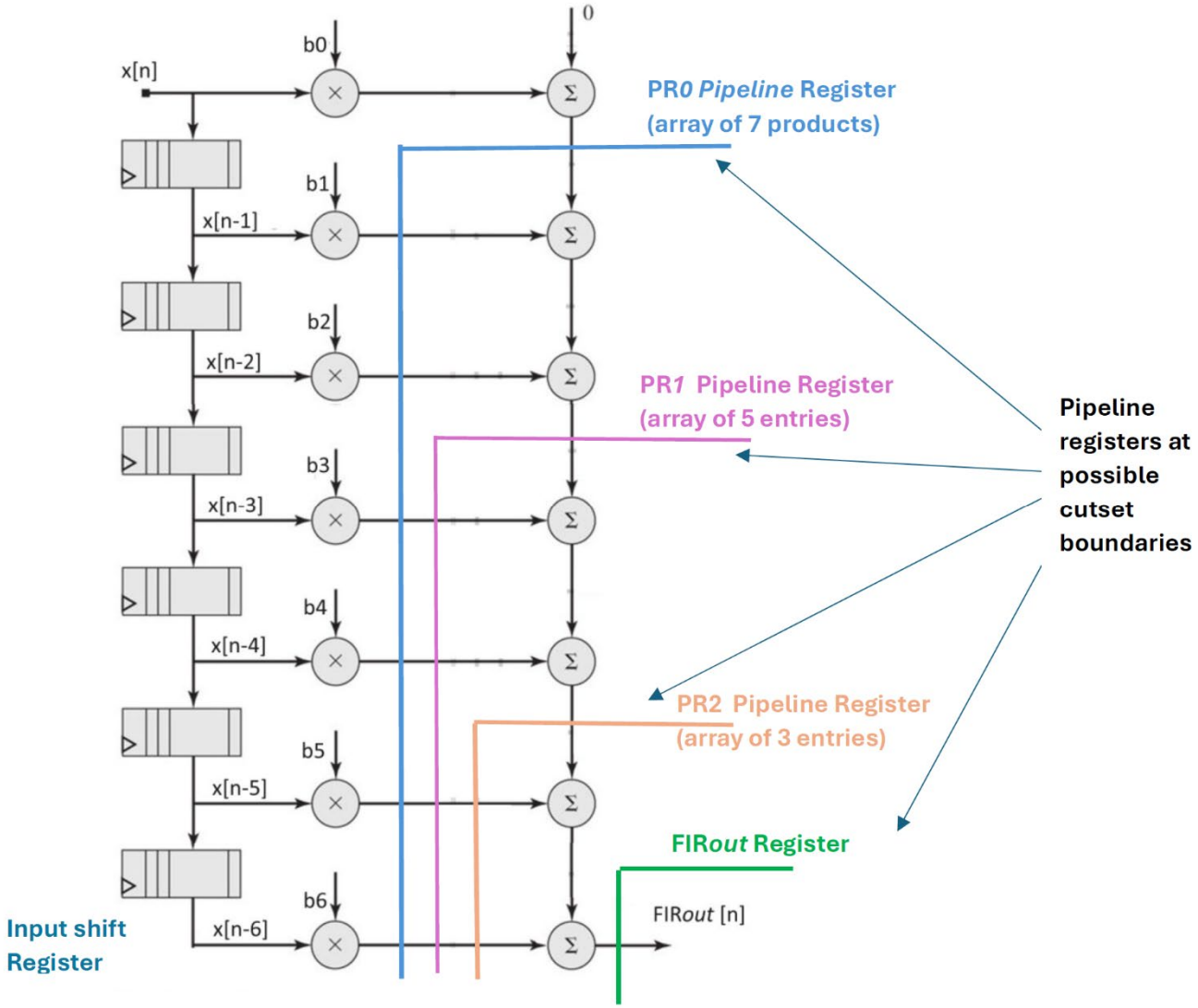
Assignment Question 2:

Consider the following 6th order FIR filter. The filter coefficients are given in 4bit numbers as follows: b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2. The input samples are also 4 bits wide with positive values. The output of the FIR filter is assigned 11 parallel bits. The output FIRout should be a registered output. It is required to add one pipeline register PR0 as shown in figure. Design the module and verify its functionality and the data coherence and compare it to a design with the same FIR filter coefficients but without any pipelining. Comment on the latency and throughput. (see example posted for thourough guidelines)



Assignment Question 3:

Consider the following 6th order FIR filter. The filter coefficients are given in 4bit numbers as follows: b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2. The input samples are also 4 bits wide with positive values. The output of the FIR filter is assigned 11 parallel bits. The output FIRout should be a registered output.
It is required to add pipeline registers PR0, PR1, PR2 as shown in figure. Design the module and verify its functionality and the data coherence and compare it to a design with the same FIR filter coefficients but without any pipelining. Comment on the latency and throughput. (see example posted for thorough guidelines)



```
/*-----------------------------------------------------------------------------------------
                                  Lesson 10: Pipelining
-----------------------------------------------------------------------------------------
Assignment 2:   FIR_pipeline_multiplier
--------------
Consider the following 6th order FIR filter. The filter coefficients
are given in 4bit numbers as follows:
b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2. The input samples
are also 4 bits wide with positive values. The output of the FIR filter is assigned
11 parallel bits. The output FIRout should be a registered output.
It is required to add one pipeline register PR0 as shown in figure. Design the
module and verify its functionality and the data coherence and compare it to a
design with the same FIR filter coefficients but without any pipelining.
Comment on the latency and throughput. (see example posted for thourough guidelines)

Assignment 3: FIR_pipeline_3_Adders
-------------
Consider the following 6th order FIR filter. The filter coefficients are given in
4bit numbers as follows: b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2.
The input samples are also 4 bits wide with positive values. The output of the FIR filter
is assigned 11 parallel bits. The output FIRout should be a registered output.
It is required to add pipeline registers PR0, PR1, PR2 as shown in figure.
Design the module and verify its functionality and the data coherence and compare
it to a design with the same FIR filter coefficients but without any pipelining.
Comment on the latency and throughput. (see example posted for thourough guidelines)
-----------------------------------------------------------------------------------------*/

module FIR_pipeline ( FIR_A, FIR_B, FIR_C, Sample_in, clock, reset);

parameter FIR_order     = 6;
parameter sample_size   = 4;                              // maximum sample value is 15
parameter weight_size   = 4;                              // maximum value may be 15
parameter word_size_out = 11;                             // log2(15*15*7)

//output  [word_size_out -1: 0] FIR_assign;
output  [word_size_out -1: 0] FIR_A;
output  [word_size_out -1: 0] FIR_B;
output  [word_size_out -1: 0] FIR_C;

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


FIR_pipeline_none       U1 ( FIR_A, Sample_in, clock, reset);
FIR_pipeline_multiplier U2 ( FIR_B, Sample_in, clock, reset);
FIR_pipeline_3adders    U3 ( FIR_C, Sample_in, clock, reset);

endmodule
```

```
module FIR_pipeline_none ( FIR_out, Sample_in, clock, reset);

parameter FIR_order     = 6;
parameter sample_size   = 4;                    // maximum sample value is 15
parameter weight_size   = 4;                    // maximum value may be 15
parameter word_size_out = 11;                   // log2(15*15*(6+1)

output reg [word_size_out -1: 0] FIR_out;

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

reg        [word_size_out -1: 0] comb_out;
wire       [weight_size   -1: 0] coefficients [0: FIR_order];

// Filter coefficients
parameter b0 = 4'd2;       assign coefficients [0] = b0;
parameter b1 = 4'd5;       assign coefficients [1] = b1;
parameter b2 = 4'd9;       assign coefficients [2] = b2;
parameter b3 = 4'd14;      assign coefficients [3] = b3;
parameter b4 = 4'd9;       assign coefficients [4] = b4;
parameter b5 = 4'd5;       assign coefficients [5] = b5;
parameter b6 = 4'd2;       assign coefficients [6] = b6;

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

integer k, m, n, c;

// Create the combinational logic
always @ (Sample_in or Sample_Array)
      begin
          comb_out = coefficients [0] * Sample_in;
      for (m=1; m<= FIR_order; m=m+1)
          comb_out = comb_out + coefficients[m] * Sample_Array[m];
      end

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;
         FIR_out              <= 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           <= comb_out;
         end

endmodule
```

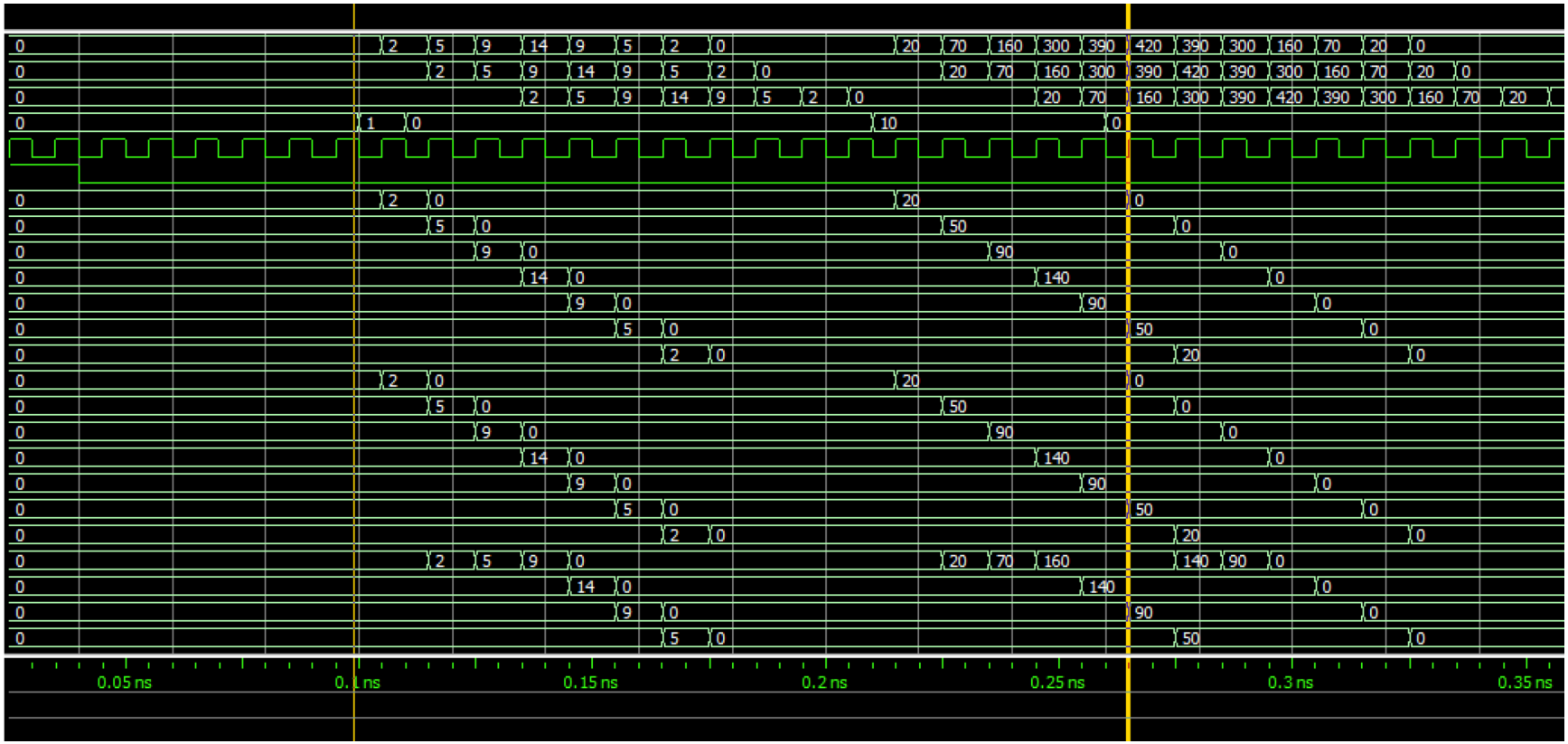
```
//-------------------------------------------------------------------------------------------
// (a) Alternative pipeline structures for FIR filter with pipeline registers placed
//     at the outputs of the multipliers
//-------------------------------------------------------------------------------------------

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

    parameter FIR_order     = 6;
    parameter sample_size   = 4;                                  // maximum sample value is 15
    parameter weight_size   = 4;                                  // maximum value may be 15
    parameter word_size_out = 11;                                 // log2(2^4*2^4*(order+1))

    parameter product_size  = sample_size + weight_size;

    output reg [word_size_out -1   : 0] FIR_out;
    input      [sample_size -1     : 0] Sample_in;
    input                               clock, reset;
    wire       [weight_size -1     : 0] coefficients [0: FIR_order];

    // Filter coefficients
    parameter b0 = 4'd2;       assign coefficients [0] = b0;
    parameter b1 = 4'd5;       assign coefficients [1] = b1;
    parameter b2 = 4'd9;       assign coefficients [2] = b2;
    parameter b3 = 4'd14;      assign coefficients [3] = b3;
    parameter b4 = 4'd9;       assign coefficients [4] = b4;
    parameter b5 = 4'd5;       assign coefficients [5] = b5;
    parameter b6 = 4'd2;       assign coefficients [6] = b6;

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

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

    always @ (posedge clock)
         if (reset == 1) begin
             // The input shift register
             for (k = 1; k <= FIR_order; k = k+1)    // to save on code lines
                  Sample_Array[k] <= 0;
             // The pipeline register
             for (k = 0; k <= FIR_order; k = k+1)    // to save on code lines
                  PR[k]    <= 0;
             // The output register
             FIR_out       <= 0;
             end

        else 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
                 PR[0] <= coefficients [0] * Sample_in;
             for (n = 1; n<= FIR_order; n = n+1)
                 PR[n] <= coefficients [n] * Sample_Array[n];

             // The output register
              FIR_out  <= PR[0] + PR[1] + PR[2] + PR[3] + PR[4] + PR[5] + PR[6];
             end

        endmodule
```

```
//-------------------------------------------------------------------------------------------
//  (b) Alternative pipeline structures for FIR filter with pipeline registers placed
//      at the inputs of the adders.
//-------------------------------------------------------------------------------------------

   module FIR_pipeline_3adders ( FIR_out, Sample_in, clock, reset);

parameter FIR_order     = 6;
parameter sample_size   = 4;                              // maximum sample value is 15
parameter weight_size   = 4;                              // maximum value may be 15
parameter word_size_out = 11;                             // log2(2^6*2^5*(order+1))

output reg [word_size_out -1: 0] FIR_out;

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

wire       [weight_size   -1: 0] coefficients [0: FIR_order];

// Filter coefficients
parameter b0 = 4'd2;       assign coefficients [0] = b0;
parameter b1 = 4'd5;       assign coefficients [1] = b1;
parameter b2 = 4'd9;       assign coefficients [2] = b2;
parameter b3 = 4'd14;      assign coefficients [3] = b3;
parameter b4 = 4'd9;       assign coefficients [4] = b4;
parameter b5 = 4'd5;       assign coefficients [5] = b5;
parameter b6 = 4'd2;       assign coefficients [6] = b6;

reg [sample_size -1 : 0] Sample_Array [1: FIR_order];   // 5th coefficient multiplied by Data_in
integer k, n, c, m;

reg [word_size_out -1: 0] PR0 [0 : FIR_order];         // To accomodate the accumulation
reg [word_size_out -1: 0] PR1 [1 : FIR_order];
reg [word_size_out -1: 0] PR2 [2 : FIR_order];


always @ (posedge clock)
     if (reset == 1) begin

          // The input shift register
          for (k = 1; k <= FIR_order; k = k+1)    // to save on code lines
               Sample_Array[k] <= 0;

          // The pipeline register PR0
          for (k = 0; k <= FIR_order; k = k+1)    // to save the 7 products
               PR0[k]    <= 0;
          // The pipeline register PR1
          for (k = 2; k <= FIR_order; k = k+1)    // to save the 5 entries
               PR1[k]    <= 0;
          // The pipeline register PR2
          for (k = 4; k <= FIR_order; k = k+1)    // to save the 3 entries
               PR2[k]    <= 0;
          // The output register
          FIR_out        <= 0;
          end
```

```
    else 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 PR0
          PR0[0] <= coefficients [0] * Sample_in;
             for (n = 1; n<= FIR_order; n= n+1)
             PR0[n] <= coefficients [n] * Sample_Array[n];

          // Pipeline Register PR1
          PR1[2] <= PR0[0] + PR0[1] + PR0[2];
          for (m = 3; m<= FIR_order; m= m+1)
             PR1[m] <= PR0[m];


          // Pipleline Register PR2
          PR2[4] <= PR1[2] + PR1[3] + PR1[4];
          for (c = 5; c<= FIR_order; c= c+1)
             PR2[c] <= PR1[c];

          // The output register
          FIR_out  <= PR2[4] + PR2[5] + PR2[6];
          end

endmodule
```

```
module FIR_pipeline_tb ();

parameter FIR_order     = 6;
parameter sample_size   = 4;                                  // maximum sample value is 15
parameter weight_size   = 4;                                  // maximum value may be 15
parameter word_size_out = sample_size + weight_size + 3;  // maximum possibe output 15*15*(6+1)
parameter product_size  = sample_size + weight_size;

wire [word_size_out -1: 0] FIR_A;
wire [word_size_out -1: 0] FIR_B;
wire [word_size_out -1: 0] FIR_C;

reg      [sample_size   -1: 0] Sample_in;
reg                            clock, reset;

FIR_pipeline UUT ( FIR_A, FIR_B, FIR_C, Sample_in, clock, reset);


// Probes to observe the pipleine register at the end of the multiplier
wire [product_size -1 : 0]  PR0;     assign PR0 = UUT.U2.PR[0];
wire [product_size -1 : 0]  PR1;     assign PR1 = UUT.U2.PR[1];
wire [product_size -1 : 0]  PR2;     assign PR2 = UUT.U2.PR[2];
wire [product_size -1 : 0]  PR3;     assign PR3 = UUT.U2.PR[3];
wire [product_size -1 : 0]  PR4;     assign PR4 = UUT.U2.PR[4];
wire [product_size -1 : 0]  PR5;     assign PR5 = UUT.U2.PR[5];
wire [product_size -1 : 0]  PR6;     assign PR6 = UUT.U2.PR[6];


// Probes to observe the pipleine register PR0 at the input of adder 1
wire [product_size -1 : 0]  PR00;     assign PR00 = UUT.U3.PR0[0];
wire [product_size -1 : 0]  PR01;     assign PR01 = UUT.U3.PR0[1];
wire [product_size -1 : 0]  PR02;     assign PR02 = UUT.U3.PR0[2];
wire [product_size -1 : 0]  PR03;     assign PR03 = UUT.U3.PR0[3];
wire [product_size -1 : 0]  PR04;     assign PR04 = UUT.U3.PR0[4];
wire [product_size -1 : 0]  PR05;     assign PR05 = UUT.U3.PR0[5];
wire [product_size -1 : 0]  PR06;     assign PR06 = UUT.U3.PR0[6];

// Probes to observe the pipleine register PR1 at the input of adder 3
wire [product_size -1 : 0]  PR12;     assign PR12 = UUT.U3.PR1[2];
wire [product_size -1 : 0]  PR13;     assign PR13 = UUT.U3.PR1[3];
wire [product_size -1 : 0]  PR14;     assign PR14 = UUT.U3.PR1[4];
wire [product_size -1 : 0]  PR15;     assign PR15 = UUT.U3.PR1[5];
wire [product_size -1 : 0]  PR16;     assign PR16 = UUT.U3.PR1[6];

// Probes to observe the pipleine register PR2 at the input of adder 5
wire [product_size -1 : 0]  PR24;     assign PR24 = UUT.U3.PR2[4];
wire [product_size -1 : 0]  PR25;     assign PR25 = UUT.U3.PR2[5];
wire [product_size -1 : 0]  PR26;     assign PR26 = UUT.U3.PR2[6];

initial
begin
clock = 0;
forever
#5 clock = ~clock;
end
```

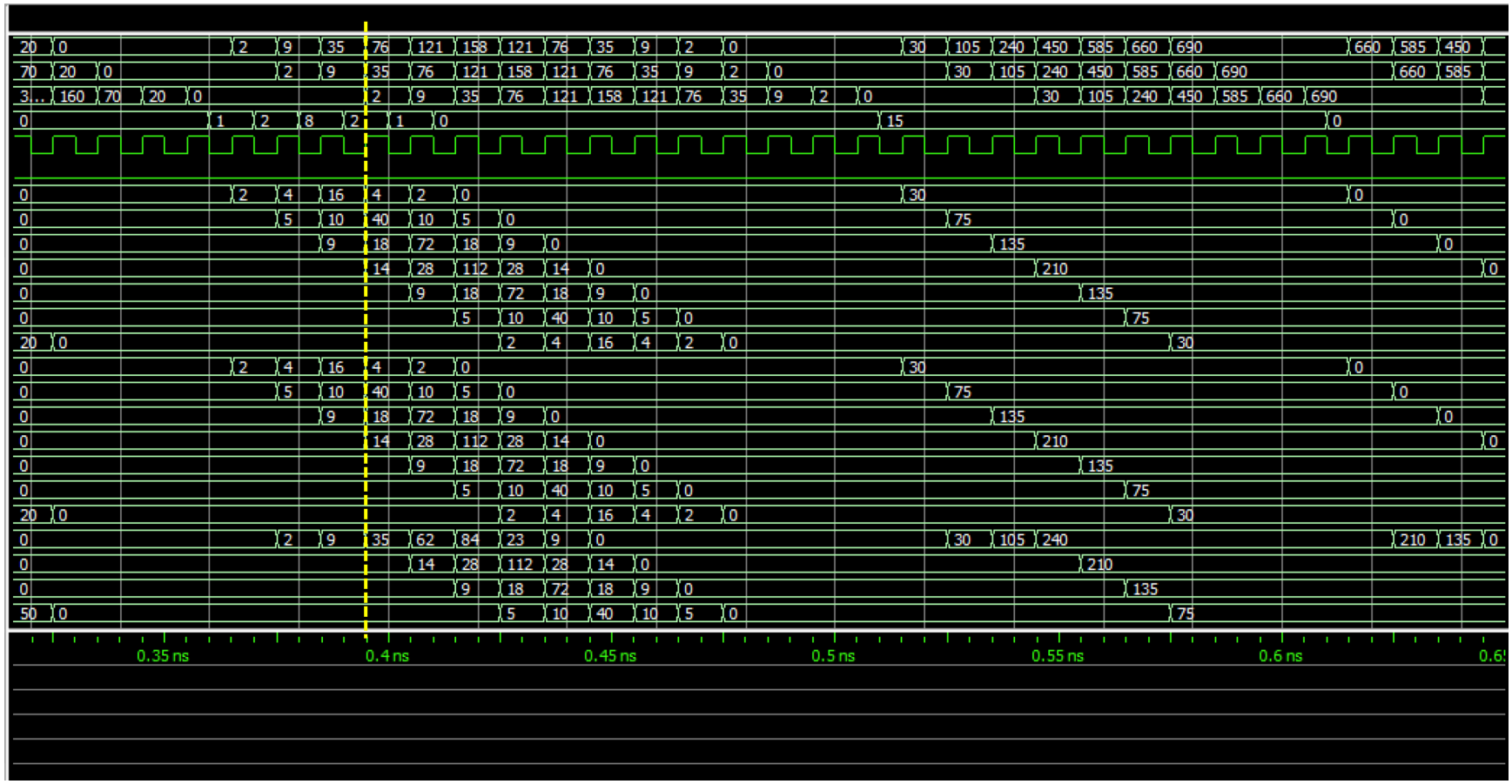
```
initial
begin
reset = 1;
#40 reset = 0;
end

initial
begin
Sample_in = 0;    #100
Sample_in = 1;    #10      // impulse response
Sample_in = 0;    #100
Sample_in = 10;   #50      // same input over 5 clock cycles
Sample_in = 0;    #100
Sample_in = 1;    #10
Sample_in = 2;    #10
Sample_in = 8;    #10
Sample_in = 2;    #10
Sample_in = 1;    #10
Sample_in = 0;    #100
Sample_in = 63;   #100
Sample_in = 0;
end

initial
   #800  $stop;

endmodule
```



The simulation results show that the 3 filters (noPipline, pipeline registers at the multipliers outputs, and the module with pipeline registers at the inputs of every other adder) have the same output values just at different latencies. The module with one pipeline register cutset shows a latency of one clock cycle, while the design with 3 pipeline register cutsets has a latency of 3 clock cycles.

When an impulse response is implemented (Sample_in equal to 1 over one clock cycle and zero otherwise – cursor on the left) we see that the output of the FIR filter shows the FIR coefficients. This is true for the 3 designs and different latencies.

The other test was keeping the input at 10 over 5 clock cycles. In this case we see how the output is accumulating the products of Sample_in and the FIR coefficients. The value 10 was used to make it easier to track the values. At the 2nd curser on the right, we see the accumulated value at FIR_out (420 = 0+50+90+140+90+50+0), (390 = 20+50+90+140+90+0+0), (160 = 20+50+90+0+0+0+0) showing the different latencies.



We see here how the internal pipeline register holds the products of the coefficients and the Sample_in.
The sum of the aligned products corresponds to the FIR_out with different latencies for the different designs.
(4+5=9), (16+10+9=35) and so on. We verify the coherency of data with the pipeline registers.

```
// Filter coefficients
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;
```