

Lesson 10 Pipelining

Assignment Question 3: (40 points)

Consider the following 6th order FIR filter (Figure 10.1). The filter coefficients are given in 4bit numbers as follows:

$b_0 = 2$, $b_1 = 5$, $b_2 = 9$, $b_3 = 14$, $b_4 = 9$, $b_5 = 5$, and $b_6 = 2$. 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 PRO, PR1, PR2 as shown in Figure 10.3. This design shall heretofore be known as design C. 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)

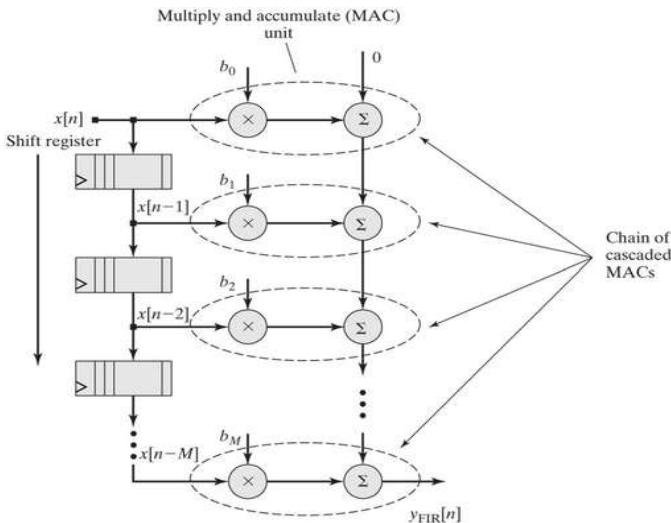


Figure 10.1

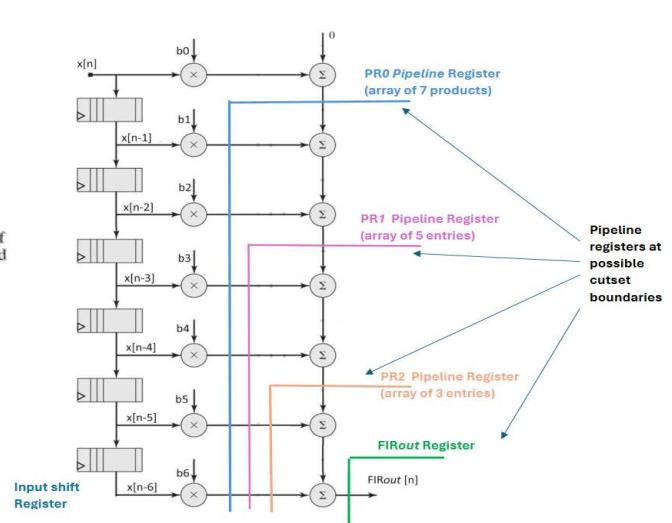


Figure 10.3

```

1 // ee417 lesson 10 Assignment 2 L10A2
2 // Name: Ron Kalin, Date: 07-18-24 Group: Kalin/Jammeh
3 // Design: FIR filter chain of cascaded MACs, given 4-bit coefficients
4 // inputs are 4-bit positive values, output register is 11 parallel bits
5 // top level module
6 module Pipeline_FIR (FIR_pipeline_A, FIR_pipeline_B, FIR_pipeline_C,
7                     FIR_MAC, Sample_in, clock, reset);
8
9 parameter FIR_order      = 6; // order of the filter
10 parameter sample_size     = 4; // word size of input samples max 15
11 parameter weight_size     = 5; // component of word_size_out max 31
12 parameter word_size_out   = sample_size + weight_size + 3; // max possible output
15*31*(6+1)= 12digits
13
14 output [word_size_out -1: 0] FIR_MAC;           //declare outputs
15 output [word_size_out -1: 0] FIR_pipeline_A;
16 output [word_size_out -1: 0] FIR_pipeline_B;
17 output [word_size_out -1: 0] FIR_pipeline_C;
18
19 input  [sample_size -1: 0] Sample_in;          //declare inputs
20 input    clock, reset;
21
22 wire   [word_size_out -1: 0] FIR_assign; //internal wire
23 //instantiate submodules
24 noPipeline_FIR M1 ( FIR_MAC, FIR_assign, Sample_in, clock, reset );
25 Pipeline_FIR_a M2 ( FIR_pipeline_A, Sample_in, clock, reset ); //pipe @ mult out
26 Pipeline_FIR_b M3 ( FIR_pipeline_B, Sample_in, clock, reset ); //pipe @ every sum in
27 Pipeline_FIR_c M4 ( FIR_pipeline_C, Sample_in, clock, reset ); //pipe @ every other sum in
28
29 endmodule
30
31 //multiply and accumulate MAC
32 module MAC (Acc_out, Sample_in, b, Acc_in);
33 parameter sample_size = 4; // word size of input samples max 15
34 parameter weight_size = 5; // component of word_size_out max 31
35 parameter word_size_out = sample_size + weight_size + 3; // max possible output
15*31*(6+1)=3255, 12 digits
36 output [word_size_out-1: 0] Acc_out;
37 input  [sample_size -1: 0] Sample_in, b, Acc_in;
38
39 assign Acc_out = (Sample_in * b) + Acc_in;
40 endmodule
41
42 //FIR with no pipeline using combinational logic
43 module noPipeline_FIR ( FIR_out_MAC, FIR_out_assign, Sample_in, clock, reset );
44
45 parameter FIR_order      = 6; // order of the filter
46 parameter sample_size     = 4; // word size of input samples max 15
47 parameter weight_size     = 5; // component of word_size_out max 31
48 parameter word_size_out   = sample_size + weight_size + 3; // max possible output
15*31*(6+1)=3255, 12 digits
49
50 output reg [word_size_out -1: 0] FIR_out_MAC;           //declare outputs
51 output reg [word_size_out -1: 0] FIR_out_assign;
52
53 input  [sample_size -1: 0] Sample_in;          //declare inputs
54 input    clock, reset;
55
56 //internal wires
57 wire   [word_size_out -1: 0] Acc0, Acc1, Acc2, Acc3, Acc4, Acc5, Acc6;
58 wire   [word_size_out -1: 0] comb_out;
59
60 //filter coefficients given
61 //b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2
62 parameter b0 = 4'd2;
63 parameter b1 = 4'd5;
64 parameter b2 = 4'd9;
65 parameter b3 = 4'd14;
66 parameter b4 = 4'd9;
67 parameter b5 = 4'd5;

```

```

68     parameter b6 = 4'd2;
69
70     reg [sample_size -1: 0] Sample_Array [1: FIR_order]; //7th coefficient multiplied by Data_in
71     integer k;
72
73     MAC M0 ( Acc0, Sample_in,          b0, 0 ); //combinational logic
74     MAC M1 ( Acc1, Sample_Array [1], b1, Acc0 ); //combinational logic
75     MAC M2 ( Acc2, Sample_Array [2], b2, Acc1 ); //combinational logic
76     MAC M3 ( Acc3, Sample_Array [3], b3, Acc2 ); //combinational logic
77     MAC M4 ( Acc4, Sample_Array [4], b4, Acc3 ); //combinational logic
78     MAC M5 ( Acc5, Sample_Array [5], b5, Acc4 ); //combinational logic
79     MAC M6 ( Acc6, Sample_Array [6], b6, Acc5 ); //combinational logic
80
81     //alternate way to create combinational logic
82     assign comb_out = b0 * Sample_in
83             + b1 * Sample_Array [1]
84             + b2 * Sample_Array [2]
85             + b3 * Sample_Array [3]
86             + b4 * Sample_Array [4]
87             + b5 * Sample_Array [5]
88             + b6 * Sample_Array [6];
89
90     always @ (posedge clock)
91         if (reset == 1) begin
92             for (k = 1; k <= FIR_order; k=k+1)
93                 Sample_Array [k] <= 0; //set registers to zero
94                 FIR_out_MAC <= 0;
95                 FIR_out_assign <= 0;
96         end
97     else begin
98             Sample_Array [1] <= Sample_in;
99             for (k = 2; k <= FIR_order; k= k+1)
100                Sample_Array [k]<= Sample_Array [k-1];
101                FIR_out_assign <= comb_out;
102                FIR_out_MAC <= Acc6;
103        end
104    endmodule
105
106 // pipeline structure a: FIR filter w/ pipeline registers placed
107 // at multiplier outputs
108 module Pipeline_FIR_a ( FIR_out_pipeline, Sample_in, clock, reset);
109
110     parameter FIR_order      = 6; // order of the filter
111     parameter sample_size     = 4; // word size of input samples max 15
112     parameter weight_size     = 5; // component of word_size_out max 31
113     parameter word_size_out   = sample_size + weight_size + 3; // max possible output
114     15*31*(6+1)= 12digits
115     parameter product_size   = sample_size + weight_size;
116
117     output reg [word_size_out -1: 0] FIR_out_pipeline; //declare outputs
118
119     input  [sample_size -1: 0] Sample_in; //declare inputs
120     input          clock, reset;
121
122     //filter coefficients given
123     //b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2
124     parameter b0 = 4'd2;
125     parameter b1 = 4'd5;
126     parameter b2 = 4'd9;
127     parameter b3 = 4'd14;
128     parameter b4 = 4'd9;
129     parameter b5 = 4'd5;
130     parameter b6 = 4'd2;
131
132     reg [sample_size -1: 0] Sample_Array [1: FIR_order]; //7th coefficient multiplied by Data_in
133     integer k;
134
135     reg [product_size -1: 0] PR[0 : FIR_order]; // array format
136
137     always @ (posedge clock)

```

```

137      if (reset == 1) begin
138          // input shift register
139          for (k = 1; k <= FIR_order; k = k +1) //saves code lines
140              Sample_Array [k] <= 0;
141          // pipeline register
142          for (k = 0; k <= FIR_order; k = k +1) //saves code lines
143              PR[k] <= 0;
144          // output register
145          FIR_out_pipeline <= 0;
146      end
147
148      else begin
149          // input shift register
150          Sample_Array [1] <= Sample_in;
151          for (k = 2; k <= FIR_order; k=k+1)
152              Sample_Array [k] <= Sample_Array [k-1];
153          // pipeline register
154          PR[0] <= b0 * Sample_in;           // find products
155          PR[1] <= b1 * Sample_Array [1];
156          PR[2] <= b2 * Sample_Array [2];
157          PR[3] <= b3 * Sample_Array [3];
158          PR[4] <= b4 * Sample_Array [4];
159          PR[5] <= b5 * Sample_Array [5];
160          PR[6] <= b6 * Sample_Array [6];
161          // output register, sum products
162          FIR_out_pipeline <= PR[0] + PR[1] + PR[2] + PR[3] + PR[4] + PR[5] + PR[6];
163      end
164  endmodule
165
166
167 // Alternative pipeline structure b: FIR filter w/ pipeline registers placed
168 // at adder inputs
169 module Pipeline_FIR_b ( FIR_out, Sample_in, clock, reset);
170
171 parameter FIR_order      = 6; // order of the filter
172 parameter sample_size     = 4; // word size of input samples max 15
173 parameter weight_size    = 5; // component of word_size_out max 31
174 parameter word_size_out = sample_size + weight_size + 3; // max possible output
175 parameter product_size   = sample_size + weight_size;
176
177 output reg [word_size_out -1: 0] FIR_out; //declare outputs
178
179 input  [sample_size -1: 0] Sample_in;    //declare inputs
180 input    clock, reset;
181
182 //filter coefficients given
183 //b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2
184 parameter b0 = 4'd2;
185 parameter b1 = 4'd5;
186 parameter b2 = 4'd9;
187 parameter b3 = 4'd14;
188 parameter b4 = 4'd9;
189 parameter b5 = 4'd5;
190 parameter b6 = 4'd2;
191
192 reg [sample_size -1: 0] Sample_Array [1: FIR_order]; //7th coefficient multiplied by Data_in
193 integer k;
194
195 reg [product_size -1: 0] PR0 [0 : FIR_order]; // array format
196 reg [product_size -1: 0] PR1 [0 : FIR_order];
197 reg [product_size -1: 0] PR2 [0 : FIR_order];
198 reg [product_size -1: 0] PR3 [0 : FIR_order];
199 reg [product_size -1: 0] PR4 [0 : FIR_order];
200 reg [product_size -1: 0] PR5 [0 : FIR_order];
201
202 always @ (posedge clock)
203     if (reset == 1) begin
204         // input shift register
205         for (k = 1; k <= FIR_order; k = k +1)

```

```
206          Sample_Array[k] <= 0;
207      // pipeline register PRO
208      for (k = 0; k <= FIR_order; k = k +1)
209          PRO[k] <= 0;
210      // pipeline register PR1
211      for (k = 0; k <= FIR_order; k = k +1)
212          PR1[k] <= 0;
213      // pipeline register PR2
214      for (k = 0; k <= FIR_order; k = k +1)
215          PR2[k] <= 0;
216      // pipeline register PR3
217      for (k = 0; k <= FIR_order; k = k +1)
218          PR3[k] <= 0;
219      // pipeline register PR4
220      for (k = 0; k <= FIR_order; k = k +1)
221          PR4[k] <= 0;
222      // pipeline register PR5
223      for (k = 0; k <= FIR_order; k = k +1)
224          PR5[k] <= 0;
225      // output register
226      FIR_out <= 0;
227      end
228
229  else begin
230      // input shift register
231      Sample_Array [1] <= Sample_in;
232      for (k = 2; k <= FIR_order; k=k+1)
233          Sample_Array[k] <= Sample_Array [k-1];
234      // pipeline register PRO
235      PRO[0] <= b0 * Sample_in;           // find products
236      PRO[1] <= b1 * Sample_Array [1];
237      PRO[2] <= b2 * Sample_Array [2];
238      PRO[3] <= b3 * Sample_Array [3];
239      PRO[4] <= b4 * Sample_Array [4];
240      PRO[5] <= b5 * Sample_Array [5];
241      PRO[6] <= b6 * Sample_Array [6];
242
243      // pipeline register PR1
244      PR1[1] <= PRO[0] + PRO[1];
245      PR1[2] <= PRO[2];
246      PR1[3] <= PRO[3];
247      PR1[4] <= PRO[4];
248      PR1[5] <= PRO[5];
249      PR1[6] <= PRO[6];
250
251      // pipeline register PR2
252      PR2[2] <= PR1[1] + PR1[2];
253      PR2[3] <= PR1[3];
254      PR2[4] <= PR1[4];
255      PR2[5] <= PR1[5];
256      PR2[6] <= PR1[6];
257
258      // pipeline register PR3
259      PR3[3] <= PR2[2] + PR2[3];
260      PR3[4] <= PR2[4];
261      PR3[5] <= PR2[5];
262      PR3[6] <= PR2[6];
263
264      // pipeline register PR4
265      PR4[4] <= PR3[3] + PR3[4];
266      PR4[5] <= PR3[5];
267      PR4[6] <= PR3[6];
268
269      // pipeline register PR5
270      PR5[5] <= PR4[4] + PR4[5];
271      PR5[6] <= PR4[6];
272
273      // output register, sum products
274      FIR_out<= PR5[5] + PR5[6];
275      end
```

```

276    endmodule
277
278 // Alternative pipeline structure c: FIR filter w/ pipeline registers placed
279 // at every other adder input
280 module Pipeline_FIR_c ( FIR_out, Sample_in, clock, reset);
281
282 parameter FIR_order      = 6; // order of the filter
283 parameter sample_size     = 4; // word size of input samples max 15
284 parameter weight_size     = 5; // component of word_size_out max 31
285 parameter word_size_out   = sample_size + weight_size + 3; // max possible output
286 //15*31*(6+1)= 12digits
287 parameter product_size   = sample_size + weight_size;
288
289 output reg [word_size_out -1: 0] FIR_out; //declare outputs
290
291 input  [sample_size -1: 0] Sample_in; //declare inputs
292           clock, reset;
293
294 //filter coefficients given
295 //b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2
296 parameter b0 = 4'd2;
297 parameter b1 = 4'd5;
298 parameter b2 = 4'd9;
299 parameter b3 = 4'd14;
300 parameter b4 = 4'd9;
301 parameter b5 = 4'd5;
302 parameter b6 = 4'd2;
303
304 reg [sample_size -1: 0] Sample_Array [1: FIR_order]; //7th coefficient multiplied by Data_in
305 integer k;
306
307 reg [product_size -1: 0] PR0 [0 : FIR_order]; // array format
308 reg [product_size -1: 0] PR1 [0 : FIR_order];
309 reg [product_size -1: 0] PR2 [0 : FIR_order];
310 /*reg [product_size -1: 0] PR3 [0 : FIR_order];
311 reg [product_size -1: 0] PR4 [0 : FIR_order];
312 reg [product_size -1: 0] PR5 [0 : FIR_order];*/
313
314 always @ (posedge clock)
315   if (reset == 1) begin
316     // input shift register
317     for (k = 1; k <= FIR_order; k = k +1)
318       Sample_Array[k] <= 0;
319     // pipeline register PR0
320     for (k = 0; k <= FIR_order; k = k +1)
321       PR0[k] <= 0;
322     // pipeline register PR1
323     for (k = 0; k <= FIR_order; k = k +1)
324       PR1[k] <= 0;
325     // pipeline register PR2
326     for (k = 0; k <= FIR_order; k = k +1)
327       PR2[k] <= 0;
328     /*// pipeline register PR3
329     for (k = 0; k <= FIR_order; k = k +1)
330       PR3[k] <= 0;
331     // pipeline register PR4
332     for (k = 0; k <= FIR_order; k = k +1)
333       PR4[k] <= 0;
334     // pipeline register PR5
335     for (k = 0; k <= FIR_order; k = k +1)
336       PR5[k] <= 0; */
337     // output register
338     FIR_out <= 0;
339   end
340 else begin
341   // input shift register
342   Sample_Array [1] <= Sample_in;
343   for (k =2; k <= FIR_order; k=k+1)
344     Sample_Array[k] <= Sample_Array[k-1];

```

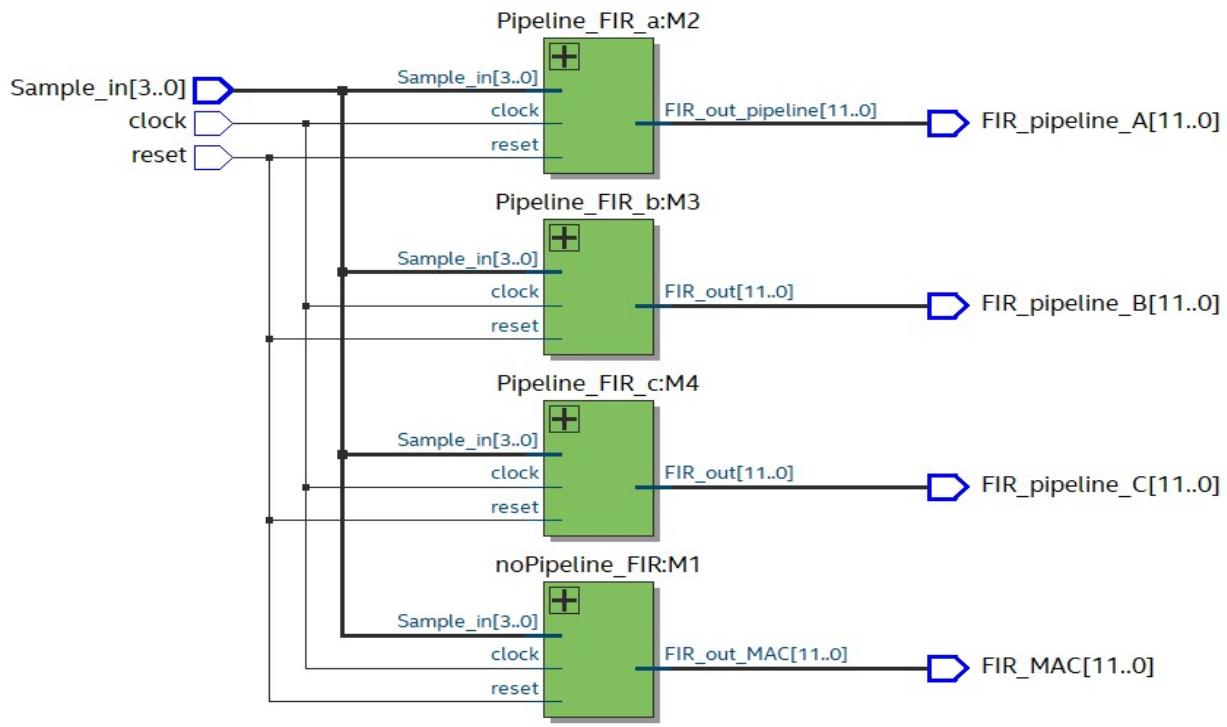
Date: July 20, 2024

Pipeline_FIR.v

Project: Pipeline_FIR

```
345      // pipeline register PRO, array of 7 products
346      PRO[0] <= b0 * Sample_in;           // find products
347      PRO[1] <= b1 * Sample_Array[1];
348      PRO[2] <= b2 * Sample_Array[2];
349      PRO[3] <= b3 * Sample_Array[3];
350      PRO[4] <= b4 * Sample_Array[4];
351      PRO[5] <= b5 * Sample_Array[5];
352      PRO[6] <= b6 * Sample_Array[6];
353
354      // pipeline register PR1, array of 5 entries
355      PR1[2] <= PRO[0] + PRO[1] + PRO[2]; //+ PR1[1] + PR1[2];
356      PR1[3] <= PRO[3];
357      PR1[4] <= PRO[4];
358      PR1[5] <= PRO[5];
359      PR1[6] <= PRO[6];
360
361      // pipeline register PR2, array of 3 entries
362      PR2[4] <= PR1[2] + PR1[3] + PR1[4];
363      PR2[5] <= PR1[5];
364      PR2[6] <= PR1[6];
365
366      // output register, sum products
367      FIR_out <= PR2[4] + PR2[5] + PR2[6];
368      end
369  endmodule
370
```

RTL Viewer



Testbench Verilog HDL Code

Date: July 20, 2024

Pipeline_FIR_tb.v

Project: Pipeline_FIR

```

1 //Name Ron Kalin Class: EE417 Summer 2024, Date: 07/19/24
2 //Lesson 10 HW Question 02 & 03
3 //Group: Ron Kalin/ Lamin Jammeh
4 //Project Description: test-bench for pipeline FIR filter
5 module Pipeline_FIR_tb ();
6
7 parameter FIR_order      = 6; // order of the filter
8 parameter sample_size     = 4; // word size of input samples max 15
9 parameter weight_size     = 5; // component of word_size_out max 31
10 parameter word_size_out = sample_size + weight_size + 3; // max possible output 15*31*(6+1)=
12digits
11 parameter product_size   = sample_size + weight_size; //max 9
12
13 wire [word_size_out -1: 0] FIR_MAC;           //declare outputs
14 wire [word_size_out -1: 0] FIR_pipeline_A;
15 wire [word_size_out -1: 0] FIR_pipeline_B;
16 wire [word_size_out -1: 0] FIR_pipeline_C;
17 //wire [word_size_out -1: 0] FIR_out_pipeline; //pipeline_a
18 //wire [word_size_out -1: 0] FIR_out; //pipeline_b
19
20 reg [sample_size -1: 0] Sample_in; //declare inputs
21 reg                      clock, reset;
22 //instantiate unit under test
23 Pipeline_FIR UUT (FIR_pipeline_A, FIR_pipeline_B, FIR_pipeline_C,
24                      FIR_MAC, Sample_in, clock, reset);
25 //Pipeline_FIR_a UUT ( FIR_out_pipeline, Sample_in, clock, reset);
26 //Pipeline_FIR_b UUT ( FIR_out, Sample_in, clock, reset);
27 wire [word_size_out -1: 0] FIR_assign; assign FIR_assign = UUT.FIR_assign; //probe for
no pipeline assign
28 // Probes to observe pipeline register PRO -b
29 wire [product_size -1: 0] PR00; assign PR00 = UUT.M3.PRO[0];
30 wire [product_size -1: 0] PR01; assign PR01 = UUT.M3.PRO[1];
31 wire [product_size -1: 0] PR02; assign PR02 = UUT.M3.PRO[2];
32 wire [product_size -1: 0] PR03; assign PR03 = UUT.M3.PRO[3];
33 wire [product_size -1: 0] PR04; assign PR04 = UUT.M3.PRO[4];
34 wire [product_size -1: 0] PR05; assign PR05 = UUT.M3.PRO[5];
35 wire [product_size -1: 0] PR06; assign PR06 = UUT.M3.PRO[6];
36
37 // Probes to observe pipeline register PR1 -b
38 wire [product_size -1: 0] PR11; assign PR11 = UUT.M3.PR1[1];
39 wire [product_size -1: 0] PR12; assign PR12 = UUT.M3.PR1[2];
40 wire [product_size -1: 0] PR13; assign PR13 = UUT.M3.PR1[3];
41 wire [product_size -1: 0] PR14; assign PR14 = UUT.M3.PR1[4];
42 wire [product_size -1: 0] PR15; assign PR15 = UUT.M3.PR1[5];
43 wire [product_size -1: 0] PR16; assign PR16 = UUT.M3.PR1[6];
44
45 // Probes to observe pipeline register PR2 -b
46 wire [product_size -1: 0] PR22; assign PR22 = UUT.M3.PR2[2];
47 wire [product_size -1: 0] PR23; assign PR23 = UUT.M3.PR2[3];
48 wire [product_size -1: 0] PR24; assign PR24 = UUT.M3.PR2[4];
49 wire [product_size -1: 0] PR25; assign PR25 = UUT.M3.PR2[5];
50 wire [product_size -1: 0] PR26; assign PR26 = UUT.M3.PR2[6];
51
52 // Probes to observe pipeline register PR3 -b
53 wire [product_size -1: 0] PR33; assign PR33 = UUT.M3.PR3[3];
54 wire [product_size -1: 0] PR34; assign PR34 = UUT.M3.PR3[4];
55 wire [product_size -1: 0] PR35; assign PR35 = UUT.M3.PR3[5];
56 wire [product_size -1: 0] PR36; assign PR36 = UUT.M3.PR3[6];
57
58 // Probes to observe pipeline register PR4 -b
59 wire [product_size -1: 0] PR44; assign PR44 = UUT.M3.PR4[4];
60 wire [product_size -1: 0] PR45; assign PR45 = UUT.M3.PR4[5];
61 wire [product_size -1: 0] PR46; assign PR46 = UUT.M3.PR4[6];
62
63 // Probes to observe pipeline register PR5 -b
64 wire [product_size -1: 0] PR55; assign PR55 = UUT.M3.PR5[5];
65 wire [product_size -1: 0] PR56; assign PR56 = UUT.M3.PR5[6];
66
67 // Probes to observe pipeline register PRO -c
68 wire [product_size -1: 0] PR00c; assign PR00c = UUT.M4.PRO[0];

```

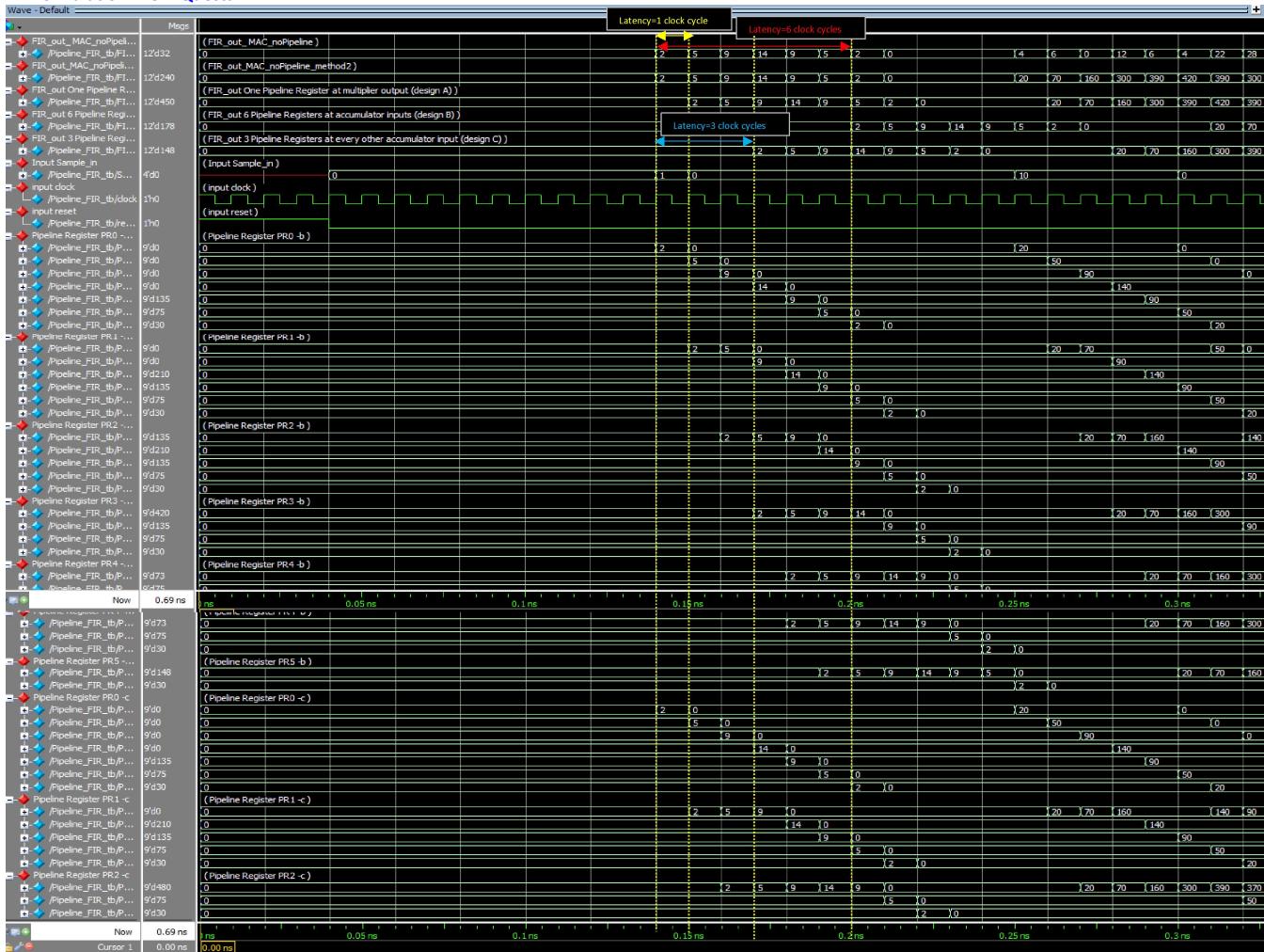
Date: July 20, 2024

Pipeline_FIR_tb.v

Project: Pipeline_FIR

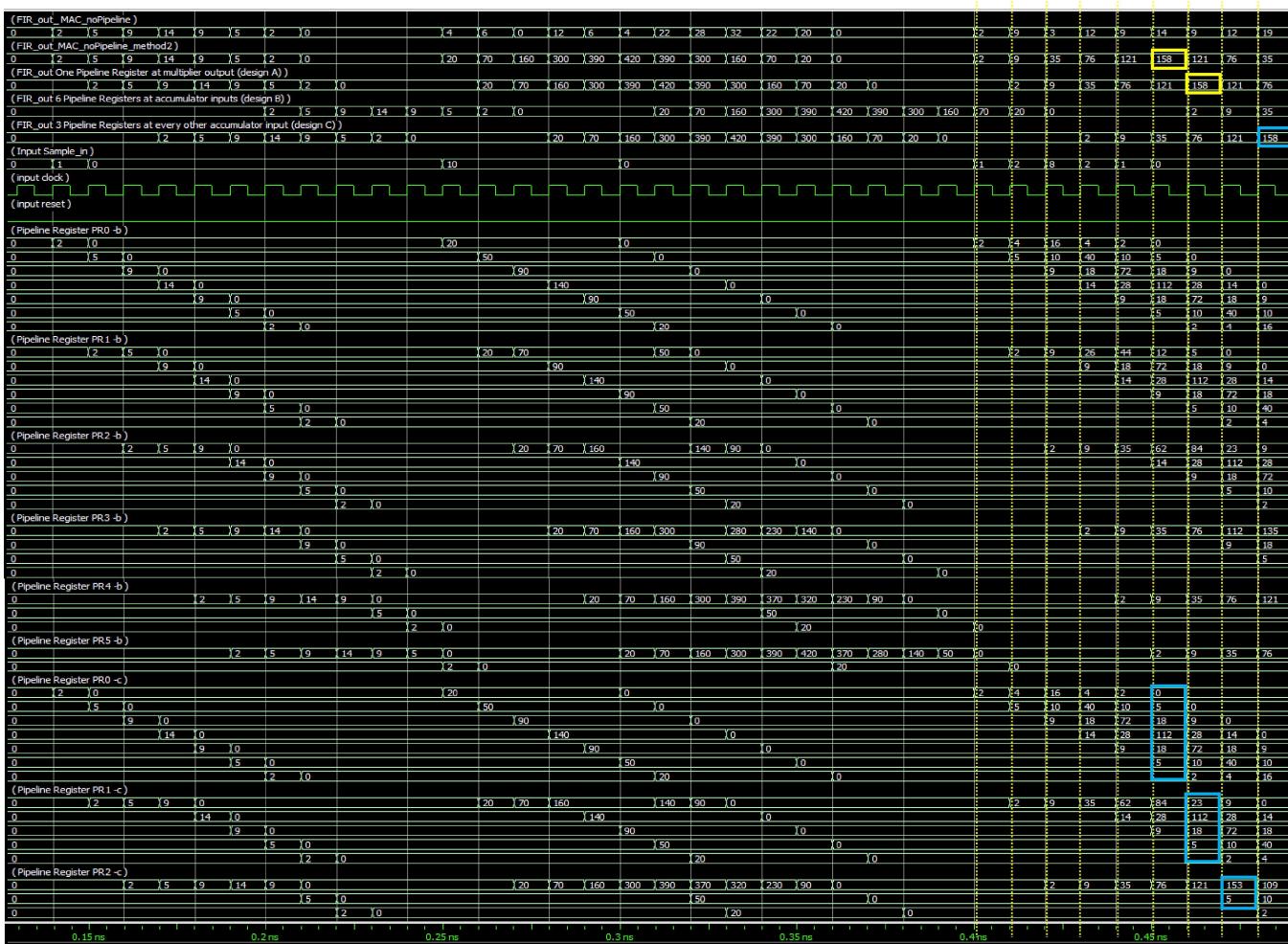
```
69  wire [product_size -1: 0] PR01c; assign PR01c = UUT.M4.PRO[1];
70  wire [product_size -1: 0] PR02c; assign PR02c = UUT.M4.PRO[2];
71  wire [product_size -1: 0] PR03c; assign PR03c = UUT.M4.PRO[3];
72  wire [product_size -1: 0] PR04c; assign PR04c = UUT.M4.PRO[4];
73  wire [product_size -1: 0] PR05c; assign PR05c = UUT.M4.PRO[5];
74  wire [product_size -1: 0] PR06c; assign PR06c = UUT.M4.PRO[6];
75
76 // Probes to observe pipeline register PR1 -c
77 wire [product_size -1: 0] PR12c; assign PR12c = UUT.M4.PR1[2];
78 wire [product_size -1: 0] PR13c; assign PR13c = UUT.M4.PR1[3];
79 wire [product_size -1: 0] PR14c; assign PR14c = UUT.M4.PR1[4];
80 wire [product_size -1: 0] PR15c; assign PR15c = UUT.M4.PR1[5];
81 wire [product_size -1: 0] PR16c; assign PR16c = UUT.M4.PR1[6];
82
83 // Probes to observe pipeline register PR2 -c
84 wire [product_size -1: 0] PR24c; assign PR24c = UUT.M4.PR2[4];
85 wire [product_size -1: 0] PR25c; assign PR25c = UUT.M4.PR2[5];
86 wire [product_size -1: 0] PR26c; assign PR26c = UUT.M4.PR2[6];
87
88 //clock cycle
89 always //#5 clock = clock;
90 begin
91   clock = 1;
92   forever #5 clock = ~clock;
93 end
94
95 initial begin
96   reset = 1; //reset everything
97   #40 reset = 0; //4 clock cycles, allow to run
98   //end
99
100 //test stimulus
101 Sample_in = 0; #100
102 Sample_in = 1; #10 //impulse response
103 Sample_in = 0; #100
104 Sample_in = 10; #50 //same input over 5 clock cycles.
105 Sample_in = 0; #100
106 Sample_in = 1; #10
107 Sample_in = 2; #10
108 Sample_in = 8; #10
109 Sample_in = 2; #10
110 Sample_in = 1; #10
111 Sample_in = 0; #100
112 Sample_in = 15; #100
113 Sample_in = 0; #40 $stop; //stop simulation, close debug window to view waveform
114 end
115
116 //display results
117 always @(posedge clock)
118 begin
119 $display("FIR_MAC: %u, FIR_pipeline_A: %u, FIR_pipeline_B: %u, FIR_pipeline_C: %u" ,
120         FIR_MAC, FIR_pipeline_A, FIR_pipeline_B, FIR_pipeline_C);
121 //$display("FIR_out_pipeline: %h", FIR_out_pipeline );
122 //$display("FIR_out: %h", FIR_out );
123 end
124 endmodule
```

RTL Simulation -from Questa



Matching results between all 4 designs (no, A, B, C) confirms data coherency for the pipelined design.

The latency between the design with no pipeline can be seen, design a (yellow), design c (blue), and design b (red).



Matching results between design a (yellow) and design c (blue). Design c can be seen that the registers add up to the total output at the top.

Output Table

```

# FIR_Mac: 000, FIR_pipeline_A: 046, FIR_pipeline_B: 000, FIR_pipeline_C: 000
# FIR_Mac: 00c, FIR_pipeline_A: 0a0, FIR_pipeline_B: 000, FIR_pipeline_C: 014
# FIR_Mac: 006, FIR_pipeline_A: 12c, FIR_pipeline_B: 000, FIR_pipeline_C: 046
# FIR_Mac: 004, FIR_pipeline_A: 186, FIR_pipeline_B: 000, FIR_pipeline_C: 0a0
# FIR_Mac: 016, FIR_pipeline_A: 1a4, FIR_pipeline_B: 014, FIR_pipeline_C: 12c
# FIR_Mac: 01c, FIR_pipeline_A: 186, FIR_pipeline_B: 046, FIR_pipeline_C: 186
# FIR_Mac: 020, FIR_pipeline_A: 12c, FIR_pipeline_B: 0a0, FIR_pipeline_C: 1a4
# FIR_Mac: 016, FIR_pipeline_A: 0a0, FIR_pipeline_B: 12c, FIR_pipeline_C: 186
# FIR_Mac: 014, FIR_pipeline_A: 046, FIR_pipeline_B: 186, FIR_pipeline_C: 12c
# FIR_Mac: 000, FIR_pipeline_A: 014, FIR_pipeline_B: 1a4, FIR_pipeline_C: 0a0
# FIR_Mac: 000, FIR_pipeline_A: 000, FIR_pipeline_B: 186, FIR_pipeline_C: 046
# FIR_Mac: 000, FIR_pipeline_A: 000, FIR_pipeline_B: 12c, FIR_pipeline_C: 014
# FIR_Mac: 000, FIR_pipeline_A: 000, FIR_pipeline_B: 0a0, FIR_pipeline_C: 000
# FIR_Mac: 002, FIR_pipeline_A: 000, FIR_pipeline_B: 046, FIR_pipeline_C: 000
# FIR_Mac: 009, FIR_pipeline_A: 002, FIR_pipeline_B: 014, FIR_pipeline_C: 000
# FIR_Mac: 003, FIR_pipeline_A: 009, FIR_pipeline_B: 000, FIR_pipeline_C: 000
# FIR_Mac: 00c, FIR_pipeline_A: 023, FIR_pipeline_B: 000, FIR_pipeline_C: 002
# FIR_Mac: 009, FIR_pipeline_A: 04c, FIR_pipeline_B: 000, FIR_pipeline_C: 009
# FIR_Mac: 00e, FIR_pipeline_A: 079, FIR_pipeline_B: 000, FIR_pipeline_C: 023
# FIR_Mac: 009, FIR_pipeline_A: 09e, FIR_pipeline_B: 002, FIR_pipeline_C: 04c
# FIR_Mac: 00c, FIR_pipeline_A: 079, FIR_pipeline_B: 009, FIR_pipeline_C: 079
# FIR_Mac: 013, FIR_pipeline_A: 04c, FIR_pipeline_B: 023, FIR_pipeline_C: 09e
# FIR_Mac: 009, FIR_pipeline_A: 023, FIR_pipeline_B: 04c, FIR_pipeline_C: 079
# FIR_Mac: 002, FIR_pipeline_A: 009, FIR_pipeline_B: 079, FIR_pipeline_C: 04c
# FIR_Mac: 000, FIR_pipeline_A: 002, FIR_pipeline_B: 09e, FIR_pipeline_C: 023
# FIR_Mac: 000, FIR_pipeline_A: 000, FIR_pipeline_B: 079, FIR_pipeline_C: 009
# FIR_Mac: 000, FIR_pipeline_A: 000, FIR_pipeline_B: 04c, FIR_pipeline_C: 002
# FIR_Mac: 000, FIR_pipeline_A: 000, FIR_pipeline_B: 023, FIR_pipeline_C: 000
# FIR_Mac: 00e, FIR_pipeline_A: 000, FIR_pipeline_B: 009, FIR_pipeline_C: 000
# FIR_Mac: 009, FIR_pipeline_A: 01e, FIR_pipeline_B: 002, FIR_pipeline_C: 000
# FIR_Mac: 000, FIR_pipeline_A: 069, FIR_pipeline_B: 000, FIR_pipeline_C: 000
# FIR_Mac: 002, FIR_pipeline_A: 0f0, FIR_pipeline_B: 000, FIR_pipeline_C: 01e
# FIR_Mac: 009, FIR_pipeline_A: 1c2, FIR_pipeline_B: 000, FIR_pipeline_C: 069
# FIR_Mac: 004, FIR_pipeline_A: 249, FIR_pipeline_B: 000, FIR_pipeline_C: 0f0
# FIR_Mac: 022, FIR_pipeline_A: 294, FIR_pipeline_B: 01e, FIR_pipeline_C: 1c2
# FIR_Mac: 022, FIR_pipeline_A: 2b2, FIR_pipeline_B: 069, FIR_pipeline_C: 049
# FIR_Mac: 022, FIR_pipeline_A: 2b2, FIR_pipeline_B: 0f0, FIR_pipeline_C: 094
# FIR_Mac: 022, FIR_pipeline_A: 2b2, FIR_pipeline_B: 1c2, FIR_pipeline_C: 0b2
# FIR_Mac: 024, FIR_pipeline_A: 2b2, FIR_pipeline_B: 049, FIR_pipeline_C: 0b2
# FIR_Mac: 029, FIR_pipeline_A: 294, FIR_pipeline_B: 094, FIR_pipeline_C: 0b2
# FIR_Mac: 022, FIR_pipeline_A: 249, FIR_pipeline_B: 0b2, FIR_pipeline_C: 0b2

```

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.

This report contains both assignment 2 (design a), and assignment 3 (design c).

A FIR with no pipeline was used, along with pipeline structure (a) FIR filter with pipeline registers placed at multiplier outputs, and two alternative pipeline structures. (b) for FIR filter with pipeline registers placed all adder inputs, and (C) with pipelines positioned at every other adder input.

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.

If the minimum clock period was measured in each design, then the throughput could also be measured.