

```

1 // ee417 lesson 10 Assignment 2 L10A2 & A3
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
13                               15*31*(6+1)= 12digits
14
15 output [word_size_out -1: 0] FIR_MAC; //declare outputs
16 output [word_size_out -1: 0] FIR_pipeline_A;
17 output [word_size_out -1: 0] FIR_pipeline_B;
18 output [word_size_out -1: 0] FIR_pipeline_C;
19
20 input [sample_size -1: 0] Sample_in; //declare inputs
21 input clock, reset;
22
23 wire [word_size_out -1: 0] FIR_assign; //internal wire
24 //instantiate submodules
25 noPipeline_FIR M1 ( FIR_MAC, FIR_assign, Sample_in, clock, reset );
26 Pipeline_FIR_a M2 ( FIR_pipeline_A, Sample_in, clock, reset ); //pipe @ mult out
27 Pipeline_FIR_b M3 ( FIR_pipeline_B, Sample_in, clock, reset ); //pipe @ every sum in
28 Pipeline_FIR_c M4 ( FIR_pipeline_C, Sample_in, clock, reset ); //pipe @ every other sum in
29
30 endmodule
31
32 //multiply and accumulate MAC
33 module MAC (Acc_out, Sample_in, b, Acc_in );
34 parameter sample_size  = 4; // word size of input samples max 15
35 parameter weight_size  = 5; // component of word_size_out max 31
36 parameter word_size_out = sample_size + weight_size + 3; // max possible output
37                               15*31*(6+1)=3255, 12 digits
38 output [word_size_out-1: 0] Acc_out;
39 input [sample_size -1: 0] Sample_in, b, Acc_in;
40
41 assign Acc_out = (Sample_in * b) + Acc_in;
42 endmodule
43
44 //FIR with no pipeline using combinational logic
45 module noPipeline_FIR ( FIR_out_MAC, FIR_out_assign, Sample_in, clock, reset );
46
47 parameter FIR_order    = 6; // order of the filter
48 parameter sample_size  = 4; // word size of input samples max 15
49 parameter weight_size  = 5; // component of word_size_out max 31
50 parameter word_size_out = sample_size + weight_size + 3; // max possible output
51                               15*31*(6+1)=3255, 12 digits
52
53 output reg [word_size_out -1: 0] FIR_out_MAC; //declare outputs
54 output reg [word_size_out -1: 0] FIR_out_assign;
55
56 input [sample_size -1: 0] Sample_in; //declare inputs
57 input clock, reset;
58
59 //internal wires
60 wire [word_size_out -1: 0] Acc0, Acc1, Acc2, Acc3, Acc4, Acc5, Acc6;
61 wire [word_size_out -1: 0] comb_out;
62
63 //filter coefficients given
64 //b0 = 2, b1 = 5, b2 = 9, b3 = 14, b4 = 9, b5 = 5, and b6 = 2
65 parameter b0 = 4'd2;
66 parameter b1 = 4'd5;
67 parameter b2 = 4'd9;
68 parameter b3 = 4'd14;
69 parameter b4 = 4'd9;
70 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 // Alternative pipeline structure b: FIR filter w/ pipeline registers placed
167 // at adder inputs
168 module Pipeline_FIR_b ( FIR_out, Sample_in, clock, reset);
169
170     parameter FIR_order      = 6; // order of the filter
171     parameter sample_size    = 4; // word size of input samples max 15
172     parameter weight_size     = 5; // component of word_size_out max 31
173     parameter word_size_out   = sample_size + weight_size + 3; // max possible output
174     // 15*31*(6+1)= 12digits
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 PR0
208     for (k = 0; k <= FIR_order; k = k +1)
209         PR0[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 PR0
235     PR0[0] <= b0 * Sample_in;           // find products
236     PR0[1] <= b1 * Sample_Array[1];
237     PR0[2] <= b2 * Sample_Array[2];
238     PR0[3] <= b3 * Sample_Array[3];
239     PR0[4] <= b4 * Sample_Array[4];
240     PR0[5] <= b5 * Sample_Array[5];
241     PR0[6] <= b6 * Sample_Array[6];
242
243     // pipeline register PR1
244     PR1[1] <= PR0[0] + PR0[1];
245     PR1[2] <= PR0[2];
246     PR1[3] <= PR0[3];
247     PR1[4] <= PR0[4];
248     PR1[5] <= PR0[5];
249     PR1[6] <= PR0[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 input  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];

```

```
345 // pipeline register PR0, array of 7 products
346 PR0[0] <= b0 * Sample_in; // find products
347 PR0[1] <= b1 * Sample_Array[1];
348 PR0[2] <= b2 * Sample_Array[2];
349 PR0[3] <= b3 * Sample_Array[3];
350 PR0[4] <= b4 * Sample_Array[4];
351 PR0[5] <= b5 * Sample_Array[5];
352 PR0[6] <= b6 * Sample_Array[6];
353
354 // pipeline register PR1, array of 5 entries
355 PR1[2] <= PR0[0] + PR0[1] + PR0[2]; //+ PR1[1] + PR1[2];
356 PR1[3] <= PR0[3];
357 PR1[4] <= PR0[4];
358 PR1[5] <= PR0[5];
359 PR1[6] <= PR0[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
```