

PHASE ONE REPORT

ELEC 374

Nick Lipski

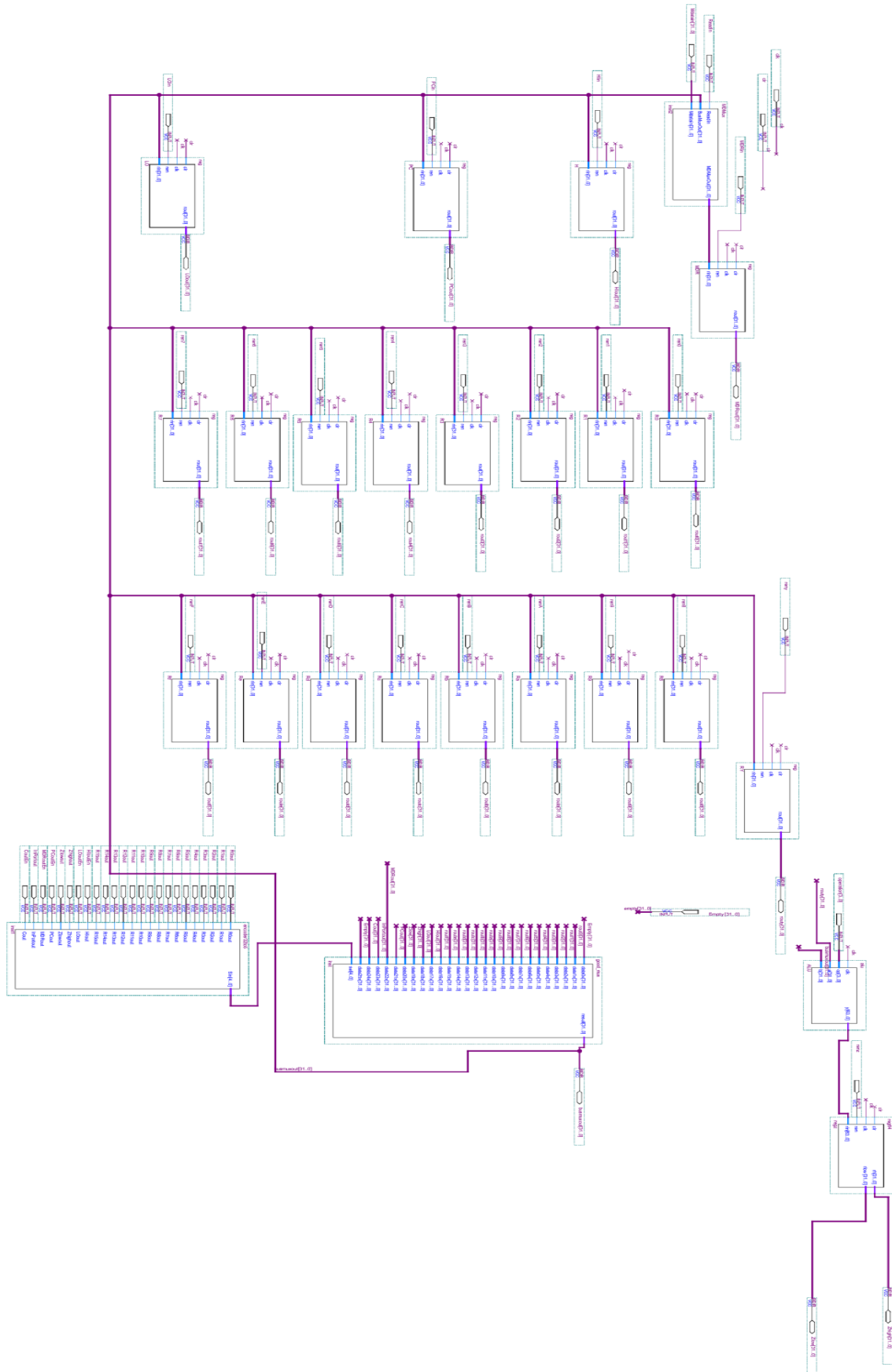
Temi Ogunsanya

Schematic of Phase 1 project

Date: March 05, 2017

Electrical

Project: Electrical



Code for Schematic's code (ELEC374.vhd)

```
1      -- Copyright (C) 1991-2013 Altera Corporation
2      -- Your use of Altera Corporation's design tools, logic functions
3      -- and other software and tools, and its AMPP partner logic
4      -- functions, and any output files from any of the foregoing
5      -- (including device programming or simulation files), and any
6      -- associated documentation or information are expressly subject
7      -- to the terms and conditions of the Altera Program License
8      -- Subscription Agreement, Altera MegaCore Function License
9      -- Agreement, or other applicable license agreement, including,
10     -- without limitation, that your use is for the sole purpose of
11     -- programming logic devices manufactured by Altera and sold by 12 -- Altera or its authorized distributors. Please
12     refer to the 13 -- applicable agreement for further details.
14
15     -- PROGRAM      "Quartus II 64-Bit"
16     -- VERSION      "Version 13.0.1 Build 232 06/12/2013 Service Pack 1 SJ Full Version"
17     -- CREATED      "Sat Mar 04 15:01:59 2017"
18
19     LIBRARY ieee;
20     USE ieee.std_logic_1164.all;
21
22     LIBRARY work;
23
24     ENTITY ELEC374 IS
25         PORT
26         (
27             ren0 : IN  STD_LOGIC;
28             ren1 : IN  STD_LOGIC;
29             ren2 : IN  STD_LOGIC;
30             ren3 : IN  STD_LOGIC;
31             ren4 : IN  STD_LOGIC;
32             ren5 : IN  STD_LOGIC;
33             ren6 : IN  STD_LOGIC;
34             ren7 : IN  STD_LOGIC;
35             ren8 : IN  STD_LOGIC;
36             ren9 : IN  STD_LOGIC;
37             renA : IN  STD_LOGIC;
38             renB : IN  STD_LOGIC;
39             renC : IN  STD_LOGIC;
40             renD : IN  STD_LOGIC;
41             renE : IN  STD_LOGIC;
42             renF : IN  STD_LOGIC;
43             clr  : IN  STD_LOGIC;
44             clk  : IN  STD_LOGIC;
45             R0out : IN  STD_LOGIC;
46             R1out : IN  STD_LOGIC;
47             R2out : IN  STD_LOGIC;
48             R3out : IN  STD_LOGIC;
49             R4out : IN  STD_LOGIC;
50             R5out : IN  STD_LOGIC;
51             R6out : IN  STD_LOGIC;
52             R7out : IN  STD_LOGIC;
53             R8out : IN  STD_LOGIC;
```

```
R9out : IN STD_LOGIC;  
R10out : IN STD_LOGIC;  
R11out : IN STD_LOGIC;  
R12out : IN STD_LOGIC;  
R13out : IN STD_LOGIC;  
R14out : IN STD_LOGIC;  
R15out : IN STD_LOGIC;  
Zhighout : IN STD_LOGIC;  
Zlowout : IN STD_LOGIC;
```

```

63      InPortout : IN STD_LOGIC;
64      Hlin : IN STD_LOGIC;
65      PCin : IN STD_LOGIC;
66      LOin : IN STD_LOGIC;
67      MDRin : IN STD_LOGIC;
68      HIoutEn : IN STD_LOGIC;
69      LOoutEn : IN STD_LOGIC;
70      PCoutEn : IN STD_LOGIC;
71      MDRoutEn : IN STD_LOGIC;
72      CoutEn : IN STD_LOGIC;
73      reny : IN STD_LOGIC;
74      renz : IN STD_LOGIC;
75      ReadIn : IN STD_LOGIC;
76      busmuxout : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
77      Empty : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 78      HIout : INOUT
STD_LOGIC_VECTOR(31 DOWNTO 0); 79      LOout : INOUT STD_LOGIC_VECTOR(31
DOWNTO 0); 80      Mdatain : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 81      MDRout :
INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
82      operation : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
83      PCout : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 84      rout0 : INOUT
STD_LOGIC_VECTOR(31 DOWNTO 0); 85      rout1 : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 86
rout2 : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 87      rout3 : INOUT
STD_LOGIC_VECTOR(31 DOWNTO 0); 88      rout4 : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 89
rout5 : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 90      rout6 : INOUT
STD_LOGIC_VECTOR(31 DOWNTO 0); 91      rout7 : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 92
rout8 : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 93      rout9 : INOUT
STD_LOGIC_VECTOR(31 DOWNTO 0); 94      routa : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 95
routb : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 96      routc : INOUT
STD_LOGIC_VECTOR(31 DOWNTO 0); 97      routd : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 98
route : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 99      routf : INOUT
STD_LOGIC_VECTOR(31 DOWNTO 0); 100      routy : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 101
Zhigh : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0); 102      Zlow : INOUT
STD_LOGIC_VECTOR(31 DOWNTO 0)
103      );
104      END ELEC374;
105
106  ARCHITECTURE bdf_type OF ELEC374 IS
107
108      COMPONENT alu
109      PORT(clk : IN STD_LOGIC;
110      a : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 111      b : IN STD_LOGIC_VECTOR(31 DOWNTO
0); 112      op : IN STD_LOGIC_VECTOR(3 DOWNTO 0); 113      y : OUT
STD_LOGIC_VECTOR(63 DOWNTO 0)
114      );
115  END COMPONENT;
116
117  COMPONENT reg
118      PORT(clr : IN STD_LOGIC;
119      clk : IN STD_LOGIC;

```

```

120         ren : IN STD_LOGIC;
121         rin : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 122         rout : OUT STD_LOGIC_VECTOR(31
            DOWNTO 0)
123     );
124 END COMPONENT;
125
126 COMPONENT good_mux
127 PORT(data0x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 128 data10x : IN STD_LOGIC_VECTOR(31
    DOWNTO 0); 129 data11x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 130 data12x : IN
    STD_LOGIC_VECTOR(31 DOWNTO 0); 131 data13x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
132 data14x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 133 data15x : IN
    STD_LOGIC_VECTOR(31 DOWNTO 0); 134 data16x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
135 data17x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 136 data18x : IN
    STD_LOGIC_VECTOR(31 DOWNTO 0); 137 data19x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
138 data1x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 139 data20x : IN
    STD_LOGIC_VECTOR(31 DOWNTO 0); 140 data21x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
141 data22x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 142 data23x : IN
    STD_LOGIC_VECTOR(31 DOWNTO 0); 143 data24x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
144 data25x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 145 data2x : IN STD_LOGIC_VECTOR(31
    DOWNTO 0); 146 data3x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 147 data4x : IN
    STD_LOGIC_VECTOR(31 DOWNTO 0); 148 data5x : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
149 data6x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 150 data7x : IN STD_LOGIC_VECTOR(31
    DOWNTO 0); 151 data8x : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 152 data9x : IN
    STD_LOGIC_VECTOR(31 DOWNTO 0); 153 sel : IN STD_LOGIC_VECTOR(4 DOWNTO 0); 154
    result : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
155 );
156 END COMPONENT;
157
158 COMPONENT encoder32to5
159 PORT(R0out : IN STD_LOGIC;
160 R1out : IN STD_LOGIC;
161 R2out : IN STD_LOGIC;
162 R3out : IN STD_LOGIC;
163 R4out : IN STD_LOGIC;
164 R5out : IN STD_LOGIC;
165 R6out : IN STD_LOGIC;
166 R7out : IN STD_LOGIC;
167 R8out : IN STD_LOGIC;
168 R9out : IN STD_LOGIC;
169 R10out : IN STD_LOGIC;
170 R11out : IN STD_LOGIC;
171 R12out : IN STD_LOGIC;
172 R13out : IN STD_LOGIC;
173 R14out : IN STD_LOGIC;
174 R15out : IN STD_LOGIC;
175 HIout : IN STD_LOGIC;
176 LOout : IN STD_LOGIC;
177 Zhighout : IN STD_LOGIC;
178 Zlowout : IN STD_LOGIC;

```

```

179         PCout : IN STD_LOGIC;
180         MDRout : IN STD_LOGIC;
181         InPortout : IN STD_LOGIC;
182         Cout : IN STD_LOGIC;
183         Sin : OUT STD_LOGIC_VECTOR(4 DOWNTO 0)
184     );
185 END COMPONENT;
186
187 COMPONENT mdmux
188     PORT(ReadIn : IN STD_LOGIC;
189         BusMuxOut : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
190         Mdatain : IN STD_LOGIC_VECTOR(31 DOWNTO 0); 191         MDMuxOut : OUT
191             STD_LOGIC_VECTOR(31 DOWNTO 0)
192     );
193 END COMPONENT;
194
195 COMPONENT reg64
196     PORT(clr : IN STD_LOGIC;
197         clk : IN STD_LOGIC;
198         ren : IN STD_LOGIC;
199         rin : IN STD_LOGIC_VECTOR(63 DOWNTO 0); 200         rh : OUT STD_LOGIC_VECTOR(31
200             DOWNTO 0); 201         rlow : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
201     );
202 END COMPONENT;
203
204
205 SIGNAL      Cout : STD_LOGIC_VECTOR(31 DOWNTO 0);
206 SIGNAL      InPortout0 : STD_LOGIC;
207 SIGNAL      InPortout1 : STD_LOGIC;
208 SIGNAL      InPortout10 : STD_LOGIC;
209 SIGNAL      InPortout11 : STD_LOGIC;
210 SIGNAL      InPortout12 : STD_LOGIC;
211 SIGNAL      InPortout13 : STD_LOGIC;
212 SIGNAL      InPortout14 : STD_LOGIC;
213 SIGNAL      InPortout15 : STD_LOGIC;
214 SIGNAL      InPortout16 : STD_LOGIC;
215 SIGNAL      InPortout17 : STD_LOGIC;
216 SIGNAL      InPortout18 : STD_LOGIC;
217 SIGNAL      InPortout19 : STD_LOGIC; 218 SIGNAL InPortout2 : STD_LOGIC;
219 SIGNAL      InPortout20 : STD_LOGIC;
220 SIGNAL      InPortout21 : STD_LOGIC;
221 SIGNAL      InPortout22 : STD_LOGIC;
222 SIGNAL      InPortout23 : STD_LOGIC;
223 SIGNAL      InPortout24 : STD_LOGIC;
224 SIGNAL      InPortout25 : STD_LOGIC;
225 SIGNAL      InPortout26 : STD_LOGIC;
226 SIGNAL      InPortout27 : STD_LOGIC;
227 SIGNAL      InPortout28 : STD_LOGIC;

```

```

228     SIGNAL      InPortout29 : STD_LOGIC; 229     SIGNAL InPortout3 : STD_LOGIC;
230     SIGNAL      InPortout30 : STD_LOGIC;
231     SIGNAL      InPortout31 : STD_LOGIC;
232     SIGNAL      InPortout4 : STD_LOGIC;
233     SIGNAL      InPortout5 : STD_LOGIC;
234     SIGNAL      InPortout6 : STD_LOGIC;
235     SIGNAL      InPortout7 : STD_LOGIC;
236     SIGNAL      InPortout8 : STD_LOGIC;
237     SIGNAL      InPortout9 : STD_LOGIC;
238     SIGNAL      SYNTHESIZED_WIRE_0 : STD_LOGIC_VECTOR(4 DOWNT0 0);
239     SIGNAL      SYNTHESIZED_WIRE_1 : STD_LOGIC_VECTOR(31 DOWNT0 0); 240     SIGNAL
        SYNTHESIZED_WIRE_2 : STD_LOGIC_VECTOR(63 DOWNT0 0); 241
242     SIGNAL      GDFX_TEMP_SIGNAL_0 : STD_LOGIC_VECTOR(31 DOWNT0 0);
243
244     BEGIN
245
246     GDFX_TEMP_SIGNAL_0 <= (InPortout31 & InPortout30 & InPortout29 & InPortout28 & InPortout27
        & InPortout26 & InPortout25 & InPortout24 & InPortout23 & InPortout22 & InPortout21 &
        InPortout20 & InPortout19 & InPortout18 & InPortout17 & InPortout16 & InPortout15 &
        InPortout14 & InPortout13 & InPortout12 & InPortout11 & InPortout10 & InPortout9 &
        InPortout8 & InPortout7 & InPortout6 & InPortout5 & InPortout4 & InPortout3 & InPortout2 &
        InPortout1 & InPortout0);
247
248
249         b2v_ALU : alu
250         PORT MAP(clk => clk,
251             a => routy,
252             b => busmuxout,
253             op => operation,
254             y => SYNTHESIZED_WIRE_2);
255
256
257         b2v_H : reg
258         PORT MAP(clr => clr,
259             clk => clk,
260             ren => HIin,
261             rin => busmuxout,
262             rout => HIout);
263
264
265         b2v_inst : good_mux
266         PORT MAP(data0x => Empty,
267             data10x => rout9,
268             data11x => routa,
269             data12x => routb,
270             data13x => routc,
271             data14x => routd,
272             data15x => route,

```



```

273     data16x => routf,
274     data17x => HIout,
275     data18x => LOout,
276     data19x => Zhigh,
277     data1x => rout0,
278     data20x => Zlow,
279     data21x => PCout,
280     data22x => MDRout,
281     data23x => GDFX_TEMP_SIGNAL_0,
282     data24x => Cout,
283     data25x => Empty,
284     data2x => rout1,
285     data3x => rout2,
286     data4x => rout3,
287     data5x => rout4,
288     data6x => rout5,
289     data7x => rout6,
290     data8x => rout7,
291     data9x => rout8,
292     sel => SYNTHESIZED_WIRE_0,
293     result => busmuxout);
294
295
296     b2v_inst1 : encoder32to5
297     PORT MAP(R0out => R0out,
298     R1out => R1out,
299     R2out => R2out,
300     R3out => R3out,
301     R4out => R4out,
302     R5out => R5out,
303     R6out => R6out,
304     R7out => R7out,
305     R8out => R8out,
306     R9out => R9out,
307     R10out => R10out,
308     R11out => R11out,
309     R12out => R12out,
310     R13out => R13out,
311     R14out => R14out,
312     R15out => R15out,
313     HIout => HIoutEn,
314     LOout => LOoutEn,
315     Zhighout => Zhighout,
316     Zlowout => Zlowout, 317 PCout => PCoutEn,
318     MDRout => MDRoutEn,
319     InPortout => InPortout,

```

```

320      Cout => CoutEn,
321      Sin => SYNTHESIZED_WIRE_0);
322
323
324      b2v_inst2 : mdmux
325      PORT MAP(ReadIn => ReadIn,
326      BusMuxOut => busmuxout,
327      Mdatain => Mdatain,
328      MDMuxOut => SYNTHESIZED_WIRE_1);
329
330
331      b2v_LO : reg
332      PORT MAP(clr => clr,
333      clk => clk,
334      ren => LOin,
335      rin => busmuxout,
336      rout => LOout);
337
338
339      b2v_MDR : reg
340      PORT MAP(clr => clr,
341      clk => clk,
342      ren => MDRin,
343      rin => SYNTHESIZED_WIRE_1,
344      rout => MDRout);
345
346
347      b2v_PC : reg
348      PORT MAP(clr => clr,
349      clk => clk,
350      ren => PCin,
351      rin => busmuxout,
352      rout => PCout);
353
354
355      b2v_R0 : reg
356      PORT MAP(clr => clr,
357      clk => clk,
358      ren => ren0,
359      rin => busmuxout,
360      rout => rout0);
361
362
363      b2v_R1 : reg
364      PORT MAP(clr => clr,
365      clk => clk,
366      ren => ren1,
367      rin => busmuxout,

```

```
368         rout => rout1);
369
370
371     b2v_R2 : reg
372     PORT MAP(clr => clr,
373             clk => clk,
374             ren => ren2,
375             rin => busmuxout,
376             rout => rout2);
377
378
379     b2v_R3 : reg
380     PORT MAP(clr => clr,
381             clk => clk,
382             ren => ren3,
383             rin => busmuxout,
384             rout => rout3);
385
386
387     b2v_R4 : reg
388     PORT MAP(clr => clr,
389             clk => clk,
390             ren => ren4,
391             rin => busmuxout,
392             rout => rout4);
393
394
395     b2v_R5 : reg
396     PORT MAP(clr => clr,
397             clk => clk,
398             ren => ren5,
399             rin => busmuxout,
400             rout => rout5);
401
402
403     b2v_R6 : reg
404     PORT MAP(clr => clr,
405             clk => clk,
406             ren => ren6,
407             rin => busmuxout,
408             rout => rout6);
409
410
411     b2v_R7 : reg
412     PORT MAP(clr => clr,
413             clk => clk,
414             ren => ren7,
415             rin => busmuxout,
```

```

416         rout => rout7);
417
418
419         b2v_R8 : reg
420         PORT MAP(clr => clr,
421         clk => clk,
422         ren => ren8,
423         rin => busmuxout,
424         rout => rout8);
425
426
427         b2v_R9 : reg
428         PORT MAP(clr => clr,
429         clk => clk,
430         ren => ren9,
431         rin => busmuxout,
432         rout => rout9);
433
434
435         b2v_Ra : reg
436         PORT MAP(clr => clr,
437         clk => clk,
438         ren => renA,
439         rin => busmuxout,
440         rout => routa);
441
442
443         b2v_Rb : reg
444         PORT MAP(clr => clr,
445         clk => clk,
446         ren => renB,
447         rin => busmuxout,
448         rout => routb);
449
450
451         b2v_Rc : reg
452         PORT MAP(clr => clr,
453         clk => clk,
454         ren => renC,
455         rin => busmuxout,
456         rout => routc);
457
458
459         b2v_Rd : reg
460         PORT MAP(clr => clr,
461         clk => clk,
462         ren => renD,
463         rin => busmuxout,

```

```

464         rout => routd);
465
466
467         b2v_Re : reg
468         PORT MAP(clr => clr,
469         clk => clk,
470         ren => renE,
471         rin => busmuxout,
472         rout => route);
473
474
475         b2v_regz : reg64
476         PORT MAP(clr => clr,
477         clk => clk,
478         ren => renz,
479         rin => SYNTHESIZED_WIRE_2,
480         rh => Zhigh,
481         rlow => Zlow);
482
483
484         b2v_Rf : reg
485         PORT MAP(clr => clr,
486         clk => clk,
487         ren => renF,
488         rin => busmuxout,
489         rout => routf);
490
491
492         b2v_RY : reg
493         PORT MAP(clr => clr,
494         clk => clk,
495         ren => reny,
496         rin => busmuxout,
497         rout => routy);
498
499
500     END bdf_type;

```

Code for Booth Multiplier with Bit recording (bmul32.vhd)

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity badd32 is
5      port (a          : in std_logic_vector(2 downto 0); -- Booth multiplier
6           b          : in std_logic_vector(31 downto 0); -- multiplicand
7           sum_in : in std_logic_vector(31 downto 0); -- sum input
8           sum_out : out std_logic_vector(31 downto 0); -- sum output
9           prod   : out std_logic_vector(1 downto 0)); --
10             2 bits of product
11 end entity badd32;
12 library IEEE;
13 use IEEE.std_logic_1164.all;
14 entity add32 is -- simple 32 bit ripple carry adder
15     port(a : in std_logic_vector(31 downto 0); 15 b : in std_logic_vector(31
16     cin : in std_logic;
17         sum : out std_logic_vector(31 downto 0);
18         cout : out std_logic);
19 end entity add32;
20 library IEEE;
21 use IEEE.std_logic_1164.all;
22 entity bmul32 is -- 32-bit by 32-bit two's complement multiplier
23     port (a : in std_logic_vector(31 downto 0); -- multiplier
24          b : in std_logic_vector(31 downto 0); -- multiplicand
25          p : out std_logic_vector(63 downto 0)); -- product
26 end entity bmul32;
27 library IEEE;
28 use IEEE.std_logic_1164.all;
29 entity fadd is -- full adder stage, interface
30     port(a : in std_logic; 31 b : in
31     std_logic; 32 cin : in std_logic; 33 s : out
32     std_logic;
33         cout : out std_logic);
34 end entity fadd;
35
36
37 architecture circuits of badd32 is
38     subtype word is std_logic_vector(31 downto 0);
39     signal bb : word; 40 signal psum :
39     word; 41 signal b_bar : word;
40     signal two_b : word; 43 signal
41     two_b_bar : word;
42     signal cout : std_logic; 45 signal cin :
42     std_logic; 46 signal topbit : std_logic; 47
43     signal topout : std_logic; 48 signal nc1 :
44     std_logic;
45     begin -- circuits of badd32
46     b_bar <= not b;
47     two_b <= b(30 downto 0) & '0';
```

```

52         two_b_bar <= not two_b;
53         bb <= b when a="001" or a="010"           -- 5-input mux
54         else two_b when a="011"
55         else two_b_bar when a="100" -- cin=1 56     else b_bar when a="101" or a="110" -- cin=1
56             else x"00000000";
57         cin <= '1' when a="100" or a="101" or a="110"
58         else '0';
59         topbit <= b(31) when a="001" or a="010" or a="011"
60         else b_bar(31) when a="100" or a="101" or a="110"
61         else '0';
62
63
64     a1: entity WORK.add32 port map(sum_in, bb, cin, psum, cout);
65     a2: entity WORK.fadd port map(sum_in(31), topbit, cout, topout, nc1); 66
66     sum_out(29 downto 0) <= psum(31 downto 2);
67     sum_out(31) <= topout;
68     sum_out(30) <= topout;
69     prod <= psum(1 downto 0);
70     end architecture circuits; -- of badd32
71
72
73     architecture circuits of bmul32 is
74     signal zer : std_logic_vector(31 downto 0) := x"00000000";           -- zeros 75     signal mul0: std_logic_vector(2
75     subtype word is std_logic_vector(31 downto 0);
76     type ary is array(0 to 15) of word;
77     signal s : ary;-- temp sums
78     begin -- circuits of bmul32
79         mul0 <= a(1 downto 0) & '0';
80         a0: entity WORK.badd32 port map(
81             mul0,      b, zer,  s( 0), p( 1 downto 0));
82         a1: entity WORK.badd32 port map(
83             a(3 downto 1), b, s( 0), s( 1), p( 3 downto 2));
84         a2: entity WORK.badd32 port map(
85             a(5 downto 3), b, s( 1), s( 2), p( 5 downto 4));
86         a3: entity WORK.badd32 port map(
87             a(7 downto 5), b, s( 2), s( 3), p( 7 downto 6));
88         a4: entity WORK.badd32 port map(
89             a(9 downto 7), b, s( 3), s( 4), p( 9 downto 8));
90         a5: entity WORK.badd32 port map(
91             a(11 downto 9), b, s( 4), s( 5), p(11 downto 10));
92         a6: entity WORK.badd32 port map(
93             a(13 downto 11), b, s( 5), s( 6), p(13 downto 12));
94         a7: entity WORK.badd32 port map(
95             a(15 downto 13), b, s( 6), s( 7), p(15 downto 14));
96         a8: entity WORK.badd32 port map(
97             a(17 downto 15), b, s( 7), s( 8), p(17 downto 16));
98         a9: entity WORK.badd32 port map(
99             a(19 downto 17), b, s( 8), s( 9), p(19 downto 18));
100

```

```

101         a10: entity WORK.badd32 port map(
102             a(21 downto 19), b, s( 9), s(10), p(21 downto 20));
103         a11: entity WORK.badd32 port map(
104             a(23 downto 21), b, s(10), s(11), p(23 downto 22)); 105 a12: entity WORK.badd32 port
                map(
106 a(25 downto 23), b, s(11), s(12), p(25 downto 24)); 107 a13: entity WORK.badd32 port map(
108 a(27 downto 25), b, s(12), s(13), p(27 downto 26)); 109 a14: entity WORK.badd32 port map(
110             a(29 downto 27), b, s(13), s(14), p(29 downto 28));
111             a15: entity WORK.badd32 port map(
112                 a(31 downto 29), b, s(14), p(31 downto 30) , p(31 downto 30));
113             end architecture circuits; -- of bmul32
114
115
116
117
118     architecture circuits of fadd is
119     begin
120         s <= a xor b xor cin after 1 ps;
121         cout <= (a and b) or (a and cin) or (b and cin) after 1
                ps;
122         end architecture circuits; -- of fadd
123
124
125
126
127     architecture circuits of add32 is
128     signal c : std_logic_vector(0 to 30);
129     begin
130         a0: entity WORK.fadd port map(a(0),
                b(0), cin, sum(0), c(0));
131         stage: for I in 1 to 30 generate
132             as: entity WORK.fadd port map(a(I),
                b(I), c(I-1) , sum(I), c(I));
133         end generate stage;
134         a31: entity WORK.fadd port
                map(a(31), b(31), c(30) , sum(31),
                cout);
135         end architecture circuits; -- of

```


Code for a 32 bit register (reg.vhd)

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity reg is
5 port( clr: in std_logic;
6       clk: in std_logic;
7       ren: in std_logic;
8       rin: in std_logic_vector(31 downto 0);
9       rout: out std_logic_vector(31 downto 0)
10      );
11     end entity reg;
12
13     architecture behavior of reg is
14     begin
15         process (clk,clr,ren,rin)
16         begin
17             if (clr= '1') then
18                 rout <= x"00000000";
19             elsif ( clk='1') then
20                 if ren = '1' then
21                     rout <= rin;
22                 end if;
23             end if;
24         end process;
25     END behavior;
```

Code for a 32-5 Encoder (encoder32to5.vhd)

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity encoder32to5 is
5  port( R0out: in std_logic; 6      R1out: in std_logic; 7
      R2out: in std_logic; 8      R3out: in std_logic; 9      R4out:
      in std_logic;
10      R5out: in std_logic;
11      R6out: in std_logic;
12      R7out: in std_logic;
13      R8out: in std_logic;
14      R9out: in std_logic;
15      R10out: in std_logic;
16      R11out: in std_logic;
17      R12out: in std_logic;
18      R13out: in std_logic;
19      R14out: in std_logic;
20      R15out: in std_logic;
21      Hlout: in std_logic;
22      LOout: in std_logic;
23      Zhighout: in std_logic;
24      Zlowout: in std_logic;
25      PCout: in std_logic;
26      MDRout: in std_logic;
27      InPortout: in std_logic;
28  Cout: in std_logic; 29      Sin: out std_logic_vector(4 downto 0)
30      );
31  end entity encoder32to5;
32
33  architecture behavior of encoder32to5 is
34  begin
35
36
37      Sin <= "00001" when R0out = '1' else
38      "00010" when R1out = '1' else
39      "00011" when R2out = '1' else
40      "00100" when R3out = '1' else
41      "00101" when R4out = '1' else
42      "00110" when R5out = '1' else
43      "00111" when R6out = '1' else
44      "01000" when R7out = '1' else
45      "01001" when R8out = '1' else
46      "01010" when R9out = '1' else
```

```
47 "01011" when R10out = '1' else
48 "01100" when R11out = '1' else
49 "01101" when R12out = '1' else
50 "01110" when R13out = '1' else
51 "01111" when R14out = '1' else
52 "10000" when R15out = '1' else
53 "10001" when Hlout = '1' else
54 "10010" when LOout = '1' else
55 "10011" when Zhighout = '1' else
56 "10100" when Zlowout = '1' else
57 "10101" when PCout = '1' else
58 "10110" when MDRout = '1' else
59 "10111" when InPortout = '1' else
60 "11000" when Cout = '1' else
61 "11111";
62
```

```
63 end behavior;
```

Code for a 5 to32bit Multiplexer (good_mux.vhd)

```
2  library ieee;
3  use ieee.std_logic_1164.all; 4  use ieee.numeric_std.all;
5  entity good_mux is
6
7  port (data0x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 8  data1x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 9  data2x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 10 data3x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 11 data4x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 12 data5x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 13 data6x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 14 data7x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 15 data8x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 16 data9x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 17 data10x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 18 data11x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 19 data12x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 20 data13x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 21 data14x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 22 data15x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 23 data16x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 24 data17x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 25 data18x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 26 data19x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 27 data20x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 28 data21x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 29 data22x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 30 data23x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 31 data24x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 32 data25x : IN STD_LOGIC_VECTOR (31 DOWNT0
0); 33 sel : IN STD_LOGIC_VECTOR (4 DOWNT0
0); 34 result : OUT STD_LOGIC_VECTOR (31
DOWNT0 0)
35 ); 36 end good_mux;
37 -----38
39 architecture behavioral of good_mux is
40 begin
41 process (sel,data0x,data1x,data2x,data3x,data4x,data5x,data6x,data7x,data8x,data9x,
data10x,data11x,data12x,data13x,data14x,data15x,data16x,data17x,data18x,data19x,data20x,
data21x,data22x,data23x,data24x,data25x)
42 begin
43 case sel is
```

```

44         when "00001" => result <=      data1x; 45
        when "00010" => result <=      data2x; 46
        when "00011" => result <=      data3x; 47
        when "00100" => result <=      data4x; 48
        when "00101" => result <=      data5x; 49
        when "00110" => result <=      data6x; 50
        when "00111" => result <=      data7x; 51
        when "01000" => result <=      data8x; 52
        when "01001" => result <=      data9x; 53
        when "01010" => result <= data10x; 54      when
"01011" => result <= data11x; 55      when "01100" =>
result <= data12x; 56      when "01101" => result <=
data13x; 57      when "01110" => result <= data14x; 58
        when "01111" => result <= data15x; 59      when
"10000" => result <= data16x; 60      when "10001" =>
result <= data17x;

61         when "10010" => result <= data18x; 62
        when "10011" => result <= data19x; 63      when
"10100" => result <= data20x; 64      when "10101" =>
result <= data21x; 65      when "10110" => result <=
data22x; 66      when "10111" => result <= data23x; 67
        when "11000" => result <= data24x; 68  when others => result
<= data25x;

69     end case;
70     end process;

71 end behavioral;
72 -----

```

Code for an ALU (ALU.vhd)

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity alu is
6  port (
7      clk      : in std_logic;
8      op       : in std_logic_vector(3 downto 0);
9      a        : in std_logic_vector(31 downto 0);
10     b        : in std_logic_vector(31 downto 0);
11     y        : out std_logic_vector(63 downto 0)
12 );
13 end entity;
14
15 architecture behavioral of alu is
16
17     component bmul32
18     port( a : in std_logic_vector(31 downto 0); -- multiplier 19      b : in
19         std_logic_vector(31 downto 0); -- multiplicand 20      p : out std_logic_vector(63
20         downto 0) -- product
21     );
22     end component bmul32;
23
24     component CLAdder
25     port(Xin: in std_logic_vector(15 downto 0);
26         Yin: in std_logic_vector(15 downto 0);
27         Cin: in std_logic;
28         Sum: out std_logic_vector (15 downto 0);
29         Cout: out std_logic
30     );
31     end component CLAdder;
32
33     component divider
34     PORT
35     (
36         denom : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
37         numer  : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
38         quotient : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
```

```

39   remain : OUT STD_LOGIC_VECTOR (31 DOWNT0 0)
40       );
41       end component divider;
42
43
44
45   type op_type is (op_and, op_add, op_sub, op_nop, op_mul, op_div, op_or, op_shr, op_shl,
   op_ror, op_rol, op_neg, op_not);
46
47       signal enum_op : op_type;
48       signal c_in: std_logic := '0';
49       signal c_outH: std_logic;
50       signal c_outL: std_logic;
51   signal a_minus_b : std_logic_vector(32
   downto 0);
52   signal a_plus_b : std_logic_vector(32 downto
   0);
53   signal a_mul_b : std_logic_vector(63 downto
   0);
54   signal a_div_b : std_logic_vector(63 downto
   0);
55   signal a_shr_b : std_logic_vector(31 downto
   0);
56   signal a_shl_b : std_logic_vector(31 downto
   0);
57   signal a_ror_b : std_logic_vector(31 downto
   0);
58   signal a_rol_b : std_logic_vector(31 downto
   0);
59   signal reg      : std_logic_vector(32 downto
   0);
60   signal reg64    : std_logic_vector(63 downto
   0);
61   signal quotient : std_logic_vector(31 downto
   0);
62   signal remainder : std_logic_vector(31 downto 0);
63
64   begin
65
66       CLAhigh: CLAdder port map (a(15 downto 0),

```

```

67         b (15 downto 0),
68         c_in,
69         a_plus_b(15 downto 0),
70         c_outL
71     );
72
73     CLAlow: CLAdder port map (a(31 downto 16),
74         b (31 downto 16),
75         c_outL,
76         a_plus_b(31 downto 16),
77         c_outH
78     );
79     a_plus_b(32)<=c_outH;
80     booth: bmul32 port map (a,
81         b,
82         a_mul_b
83     );
84
85     div32: divider port map (b,
86         a,
87         quotient,
88         remainder
89     );
90
91     a_minus_b <= std_logic_vector(signed(a(a'high) & a) - signed(b(b'high) & b)); 92
92     a_div_b    <= remainder & quotient;
93     a_shr_b    <= to_stdlogicvector(to_bitvector(a) srl to_integer( unsigned(b)));
94     a_shl_b     <= to_stdlogicvector(to_bitvector(a) sll to_integer(
95         unsigned(b)));
96     a_ror_b     <= to_stdlogicvector(to_bitvector(a) ror to_integer( unsigned(b)));
97     a_rol_b     <= to_stdlogicvector(to_bitvector(a) rol to_integer( unsigned(b)));
98     process(op,a,b,clk)
99     begin
100     case op is
101         when "0000" => enum_op <= op_and;
102         when "0001" => enum_op <= op_add;
103         when "0010" => enum_op <= op_sub;
104         when "0011" => enum_op <= op_mul;

```



```

105         when "0100" => enum_op <= op_div;
106 when "0101" => enum_op <= op_or;
107         when "0110" => enum_op <= op_shr;
108         when "0111" => enum_op <= op_shl;
109         when "1000" => enum_op <= op_ror;
110         when "1001" => enum_op <= op_rol;
111         when "1010" => enum_op <= op_neg;
112         when "1011" => enum_op <= op_not;
113         when others => enum_op <= op_nop;
114     end case;
115 end process;

116
117 process(clk,a,b,op,a_plus_b,a_minus_b,a_mul_b,a_div_b,a_shl_b,a_shr_b,a_rol_b,a_ror_b)
118 begin
119     if clk='1' then
120         case enum_op is
121             when op_add      => reg64 <= (63 downto a_plus_b'length => '0') & a_plus_b;
122             when op_sub      => reg64 <= (63 downto a_minus_b'length => '0') &
a_minus_b;
123             when op_and      => reg64 <= (63 downto 32 => '0') & (a and b);
124             when op_mul      => reg64 <= (63 downto a_mul_b'length => '0') & a_mul_b;
125             when op_div      => reg64 <= (63 downto a_div_b'length => '0') & a_div_b;
126             when op_or       => reg64 <= (63 downto 32 => '0') & (a or b);
127             when op_shr      => reg64 <= (63 downto a_shr_b'length => '0') & a_shr_b;
128             when op_shl      => reg64 <= (63 downto a_shl_b'length => '0') & a_shl_b;
129             when op_ror      => reg64 <= (63 downto a_ror_b'length => '0') & a_ror_b;
130             when op_rol      => reg64 <= (63 downto a_rol_b'length => '0') & a_rol_b;
131             when op_neg      => reg64 <= (63 downto 32 => '0') &
std_logic_vector(unsigned(not (a)) + 1);
132             when op_not      => reg64 <= (63 downto 32 => '0') & (not(to_X01(a)));
133
134             when op_nop      =>
135                 reg(32) <= '0';
136         end case;
137     end if;
138 end process;

139 process(a_plus_b,a_minus_b,a_mul_b,a_div_b,a_shr_b,reg64)
140 begin

```

```
141         y <= reg64;  
142     end process; 143 end architecture;
```

Code for a 32 bit Carry Look Ahead Adder (CLAdder.vhd)

```
1  library ieee;
1  use ieee.std_logic_1164.all;
3
4
5      entity CLAdder is
6      port(Xin: in std_logic_vector(15 downto 0);
7      Yin: in std_logic_vector(15 downto 0);
8      Cin: in std_logic;
9      Sum: out std_logic_vector (15 downto 0);
10     Cout: out std_logic
11     );
12     end CLAdder;
13
14
15 architecture behaviour of CLAdder is
16
17     signal h_sum : std_logic_vector(15 downto 0);
18     signal G: std_logic_vector(15 downto 0);
19     signal P: std_logic_vector(15 downto 0);
20     signal CarryIn: std_logic_vector(15 downto 1);
21
22     begin
23         h_sum<= Xin xor Yin;
24         G <= Xin and Yin;
25         P <= Xin or Yin;
26
27         process (G,P, CarryIn, h_sum, Cin)
28         begin
29             CarryIn(1) <= G(0) or (P(0) and Cin);
30             inst: for i in 1 to 14 loop
31                 CarryIn(i+1) <= G(i) or (P(i) and CarryIn(i));
32             end loop;
33             Cout <= G(15) or (P(15) and CarryIn(15));
34         end process;
35         Sum(0)<= h_sum(0) xor Cin;
36         Sum(15 downto 1)<= h_sum(15 downto 1) xor CarryIn(15 downto 1);
37
38     end behaviour;
39
```

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6      entity MDMux is
7          port ( ReadIn: in std_logic;
8              BusMuxOut: in std_logic_vector(31 downto 0);
9              Mdatain: in std_logic_vector(31 downto 0);
10             MDMuxOut: out std_logic_vector(31 downto 0)
11         );
12     end MDMux;
13
14
15     architecture behavior of MDMux is
16     begin
17         process (ReadIn,BusMuxOut,Mdatain)
18         begin
19             if (ReadIn= '1') then
20                 MDMuxOut <= Mdatain;
21             else
22                 MDMuxOut <= BusMuxOut;
23             end if;
24         end process;
25     end architecture

```

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity reg64 is
5  port( clr:
in std_logic;
6
       clk: in
std_logic; 7
       ren: in
std_logic; 8
       rin: in
std_logic_ve
ctor(63
downto 0); 9
       rh,rlow:
out
std_logic_ve
ctor(31
downto 0)
10      );
11      end entity reg64;
12
13      architecture behavior of reg64 is
14      begin
15      process (clk,clr, rin, ren)
16      begin
17      if (clr = '1') then
18      rh <= x"00000000";
19      rlow <= x"00000000";
20      elsif ( clk='1') then
21      if ren = '1' then
22      rh <= rin(63 downto 32);
23      rlow <= rin(31 downto 0);
24      end if;
25      end if;
26
27      end process;
28      end behaviour

```

Code for a datapath (reg.vhd)

```
1. library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4 library work;
5
6   ENTITY testbench IS
7   END testbench;
8
9   ARCHITECTURE testbench_arch OF testbench IS
10    --initialization and declaration of inputs
11    signal ren0_tb:      std_logic;
12    signal ren1_tb:      std_logic;
13    signal ren2_tb:      std_logic;
14    signal ren3_tb:      std_logic;
15    signal ren4_tb:      std_logic;
16    signal ren5_tb:      std_logic;
17    signal ren6_tb:      std_logic;
18    signal ren7_tb:      std_logic;
19    signal ren8_tb:      std_logic;
20    signal ren9_tb:      std_logic;
21    signal renA_tb:      std_logic;
22    signal renB_tb:      std_logic;
23    signal renC_tb:      std_logic;
24    signal renD_tb:      std_logic;
25    signal renE_tb:      std_logic;
26    signal renF_tb: std_logic; 27 signal clr_tb : std_logic;
28 signal clk_tb : std_logic;
29 signal R0out_tb : std_logic;
30 signal R1out_tb : std_logic;
31 signal R2out_tb : std_logic;
32 signal R3out_tb : std_logic;
33 signal R4out_tb : std_logic;
34 signal R5out_tb : std_logic;
35 signal R6out_tb : std_logic;
36 signal R7out_tb : std_logic;
37 signal R8out_tb : std_logic;
38 signal R9out_tb : std_logic;
39 signal R10out_tb :std_logic;
40 signal R11out_tb :std_logic;
41 signal R12out_tb :std_logic;
42 signal R13out_tb :std_logic;
43 signal R14out_tb :std_logic;
44 signal R15out_tb :std_logic;
45 signal Zhighout_tb: std_logic; 46 signal Zlowout_tb :std_logic;
47 signal InPortout_tb: std_logic;
```

```

48  signal Hlin_tb      :std_logic;
49  signal PCin_tb      :std_logic;
50  signal LOin_tb      :std_logic;
51  signal Empty_tb     : STD_LOGIC_VECTOR(31 DOWNT0 0);
52  signal readin_tb    :std_logic;
53  signal MDRin_tb     :std_logic;
54  signal Mdatain_tb   :STD_LOGIC_VECTOR(31 DOWNT0 0);
55  signal HIoutEn_tb   :std_logic;
56  signal LOoutEn_tb   :std_logic;
57  signal PCoutEn_tb   :std_logic;
58  signal MDRoutEn_tb  :std_logic;
59  signal CoutEn_tb    :std_logic; 60  signal reny_tb :std_logic;
61  signal operation_tb : STD_LOGIC_VECTOR(3 DOWNT0 0);
62  signal renz_tb      :std_logic;
63  signal rout2_tb     :std_logic_vector(31 downto 0);
64  signal rout3_tb     :std_logic_vector(31 downto 0);
65  signal rout4_tb     :std_logic_vector(31 downto 0);
66  signal rout5_tb     :std_logic_vector(31 downto 0);
67  signal rout6_tb     :std_logic_vector(31 downto 0);
68  signal rout7_tb     :std_logic_vector(31 downto 0);
69  signal rout8_tb     :std_logic_vector(31 downto 0);
70  signal rout9_tb     :std_logic_vector(31 downto 0);
71  signal routa_tb     :std_logic_vector(31 downto 0);
72  signal routb_tb     :std_logic_vector(31 downto 0);
73  signal routc_tb     :std_logic_vector(31 downto 0);
74  signal routd_tb     :std_logic_vector(31 downto 0);
75  signal route_tb     :std_logic_vector(31 downto 0);
76  signal routf_tb     :std_logic_vector(31 downto 0);
77  signal rout0_tb     :std_logic_vector(31 downto 0);
78  signal rout1_tb     :std_logic_vector(31 downto 0); 79  signal busmuxout_tb :std_logic_vector(31 downto 0); 80
    signal PCout_tb    :std_logic_vector(31 downto 0);
81  signal Zhigh_tb     :std_logic_vector(31 downto 0);
82  signal LOout_tb     :std_logic_vector(31 downto 0);
83  signal HIout_tb     :std_logic_vector(31 downto 0);
84  signal MDRout_tb    :std_logic_vector(31 downto 0);
85  signal routy_tb     : STD_LOGIC_VECTOR(31 DOWNT0 0);
86  signal Zlow_tb      :std_logic_vector(31 downto 0);
87
88
89  TYPE State IS(default, Reg_load1, Reg_load2, Reg_load3, T0, T1, T2, T3, T4,
    T5);
90  SIGNAL Present_state: State := default; 91
92  component ELEC374
93      PORT
94      (
95          ren0      :      IN STD_LOGIC;
96          ren1      :      IN STD_LOGIC;

```

```

97      ren2      :      IN STD_LOGIC;
98      ren3      :      IN STD_LOGIC;
99      ren4      :      IN STD_LOGIC;
100     ren5      :      IN STD_LOGIC;
101     ren6      :      IN STD_LOGIC;
102     ren7      :      IN STD_LOGIC;
103     ren8      :      IN STD_LOGIC;
104     ren9      :      IN STD_LOGIC;
105     renA      :      IN STD_LOGIC;
106     renB      :      IN STD_LOGIC;
107     renC      :      IN STD_LOGIC;
108     renD      :      IN STD_LOGIC;
109     renE      :      IN STD_LOGIC;
110     renF : IN STD_LOGIC;
111     clr : IN STD_LOGIC;
112     clk : IN STD_LOGIC;
113     R0out : IN STD_LOGIC;
114     R1out  :      IN STD_LOGIC;
115     R2out  :      IN STD_LOGIC;
116     R3out  :      IN STD_LOGIC;
117     R4out  :      IN STD_LOGIC;
118     R5out  :      IN STD_LOGIC;
119     R6out  :      IN STD_LOGIC;
120     R7out  :      IN STD_LOGIC;
121     R8out  :      IN STD_LOGIC;
122     R9out  :      IN STD_LOGIC;
123     R10out :      IN STD_LOGIC;
124     R11out :      IN STD_LOGIC;
125     R12out :      IN STD_LOGIC;
126     R13out :      IN STD_LOGIC;
127     R14out :      IN STD_LOGIC;
128     R15out : IN STD_LOGIC; 129  Zhighout : IN STD_LOGIC; 130  Zlowout : IN STD_LOGIC;
131     InPortout      :      IN STD_LOGIC;
132     HIn      :      IN STD_LOGIC;
133     PCin      :      IN STD_LOGIC;
134
135     LOin      :      IN STD_LOGIC;
136     Empty      :      IN STD_LOGIC_VECTOR(31 DOWNTO 0);
137     ReadIn : IN STD_LOGIC;
138     MDRin      :      IN STD_LOGIC;
139     Mdatain :      IN STD_LOGIC_VECTOR(31 DOWNTO 0);
140     HIoutEn :      IN STD_LOGIC;
141     LOoutEn      :      IN STD_LOGIC;
142     PCoutEn : IN STD_LOGIC; 143  MDRoutEn : IN STD_LOGIC; 144  CoutEn : IN STD_LOGIC;
145     reny      :      IN STD_LOGIC;
146     operation:      IN STD_LOGIC_VECTOR(3 DOWNTO 0);

```



```

147         renz      :      IN STD_LOGIC;
148         rout2     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
149         rout3     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
150         rout4     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
151         rout5     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
152         rout6     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
153         rout7     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
154         rout8     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
155         rout9     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
156         routa     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
157         routb     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
158         routc     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
159         routd     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
160         route     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
161         routf     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
162         rout0     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
163         rout1     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
164         busmuxout  :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
165         PCout     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
166         Zhigh     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
167         Zlow      :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
168         LOout     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
169         HIout     :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
170         MDRout    :      INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
171         routy     : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0)
172
173     );
174     end component ELEC374;
175
176 BEGIN
177
178 DUT: ELEC374 PORT MAP(
179     ren0      =>ren0_tb,
180     ren1      =>ren1_tb,
181     ren2      =>ren2_tb,
182     ren3      =>ren3_tb,
183     ren4      =>ren4_tb,
184     ren5      =>ren5_tb,
185     ren6      =>ren6_tb,
186     ren7      =>ren7_tb,
187     ren8      =>ren8_tb,
188     ren9      =>ren9_tb,
189     renA      =>renA_tb,
190     renB      =>renB_tb,
191     renC      =>renC_tb,
192     renD      =>renD_tb,
193     renE      =>renE_tb,
194     renF      =>renF_tb,

```

```

195 clr                =>clr_tb,
196 clk                =>clk_tb,
197 R0out               =>R0out_tb,
198 R1out               =>R1out_tb,
199 R2out               =>R2out_tb,
200 R3out               =>R3out_tb,
201 R4out               =>R4out_tb,
202 R5out               =>R5out_tb,
203 R6out               =>R6out_tb,
204 R7out               =>R7out_tb,
205 R8out               =>R8out_tb,
206 R9out               =>R9out_tb,
207 R10out              =>R10out_tb,
208 R11out              =>R11out_tb,
209 R12out              =>R12out_tb,
210 R13out              =>R13out_tb,
211 R14out              =>R14out_tb,
212 R15out              =>R15out_tb,
213   Zhighout =>Zhighout_tb,
214   Zlowout =>Zlowout_tb,
215   InPortout      =>InPortout_tb,
216 HIn              =>HIn_tb,
217 PCin              =>PCin_tb,
218 LOin              =>LOin_tb,
219 Empty             =>Empty_tb,
220 ReadIn            =>readin_tb,
221 MDRin             =>MDRin_tb,
222 Mdatain           =>Mdatain_tb,
223 HIoutEn            =>HIoutEn_tb,
224 LOoutEn            =>LOoutEn_tb,
225 PCoutEn            =>PCoutEn_tb,
226   MDRoutEn =>MDRoutEn_tb,
227   CoutEn =>CoutEn_tb,
228   reny        =>reny_tb,
229   operation    =>operation_tb,
230 renz           =>renz_tb,
231 rout2          =>rout2_tb,
232 rout3          =>rout3_tb,
233 rout4          =>rout4_tb,
234 rout5          =>rout5_tb,
235 rout6          =>rout6_tb,
236 rout7          =>rout7_tb,
237 rout8          =>rout8_tb,
238 rout9          =>rout9_tb,
239 routa          =>routa_tb,
240 routb          =>routb_tb,
241 routc          =>routc_tb,
242 routd          =>routd_tb,
243 route          =>route_tb,
244 routf          =>routf_tb,
245 rout0          =>rout0_tb,

```

```

246 rout1          =>rout1_tb,
247   busmuxout     =>busmuxout_tb,
248   PCout         =>PCout_tb,
249   Zhigh         =>Zhigh_tb,
250   Zlow          =>Zlow_tb,
251   LOout         =>LOout_tb,
252   HIout         =>HIout_tb,
253   MDRout        =>MDRout_tb,
254   routy => routy_tb
255   );
256
257   clk_process : process
258   begin
259       clk_tb<='0','1' after 10 ns;
260       wait for 20 ns;
261   end process clk_process;
262
263   process (clk_tb)
264   begin
265       if (clk_tb'event and clk_tb ='1') then
266       case Present_state is
267       when Default =>
268           Present_state<=Reg_load1;
269       when reg_load1 =>
270           present_state<=Reg_load2;
271       when reg_load2 =>
272           present_state<=Reg_load3;
273       when reg_load3 =>
274           present_state<=T0;
275       when T0 =>
276           present_state<=T1;
277       when T1 =>
278           present_state<=T2;
279       when T2 =>
280           present_state<=T3;
281       when T3 =>
282           present_state<=T4;
283       when T4 =>
284           present_state<=T5;
285       when others =>
286           end case;
287       end if;
288   end process;
289
290   Process(present_state)
291   begin
292   CASE present_state IS
293   default =>
294       PCin_tb<='0'; Zlowout_tb<='0'; mdROUTEn_tb<='0';
295       R1Out_tb<='0';R2Out_tb<='0';R3Out_tb<='0';

```

```

297      PCin_tb <= '0'; MDRin_tb <= '0';
298      ren1_tb<='0'; readin_tb<='0';
299      clr_tb <= '1', '0' after 4 ns;
300      renz_tb <= '0';
301      ren0_tb <= '0';
302      ren1_tb <= '0';
303      ren2_tb <= '0';
304      ren3_tb <= '0';
305      ren4_tb <= '0';
306      ren5_tb <= '0';
307      ren6_tb <= '0';
308      ren7_tb <= '0';
309      ren8_tb <= '0';
310      ren9_tb <= '0';
311      renA_tb <= '0';
312      renB_tb <= '0';
313      renC_tb <= '0';
314      renD_tb <= '0';
315      renE_tb <= '0';
316      renF_tb <= '0';
317      R0out_tb <= '0';
318      R1out_tb <= '0';
319      R2out_tb <= '0';
320      R3out_tb <= '0';
321      R4out_tb <= '0';
322      R5out_tb <= '0';
323      R6out_tb <= '0';
324      R7out_tb <= '0';
325      R8out_tb <= '0';
326      R9out_tb <= '0';
327      R10out_tb <= '0';
328      R11out_tb <= '0';
329      R12out_tb <= '0';
330      R13out_tb <= '0';
331      R14out_tb <= '0';
332      R15out_tb <= '0';
333      Zhighout_tb <= '0';
334      InPortout_tb <= '0';
335      Hlin_tb <= '0';
336
337      Loin_tb <='0';
338      HIoutEn_tb <= '0';
339      LOoutEn_tb<='0';
340      PcoutEn_tb <= '0';
341      CoutEn_tb<='0'; 342      reny_tb <= '0';
343      operation_tb <= "0000";

```

```

344      Mdatain_tb<=x"00000012";
345
346      --routy_tb <= x"00000000";
347      empty_tb <= x"00000000";
348      operation_tb<="0000";
349
350
351
352      when reg_load1 =>
353          mdatain_tb<= x"00006666";
354          readin_tb<='1','0' after 10 ns;
355          mdrin_tb<='1','0' after 10 ns;
356          mdROUTen_tb<='1', '0' after 10 ns ;
357
358              ren1_tb<='1', '0' after 10 ns ;
359
360      when reg_load2 =>
361          mdatain_tb <= x"00000015";
362          readin_tb<='1','0' after 10 ns;
363          mdrin_tb<='1','0' after 10 ns;
364          mdROUTen_tb<='1', '0' after 10 ns ;
365
366
367              ren2_tb<='1', '0' after 10 ns ;
368
369      when reg_load3 =>
370          mdatain_tb <= x"00000002";
371          readin_tb<='1','0' after 10 ns;
372          mdrin_tb<='1','0' after 10 ns;
373          mdROUTen_tb<='1', '0' after 10 ns ;
374          ren3_tb<='1', '0' after 10 ns ;
375
376      when T0 =>
377          PCoutEn_tb<='1', '0' after 10 ns ;
378          renz_tb<='1', '0' after 10 ns ;
379
380
381      when T1 =>
382          Zlowout_tb<='1', '0' after 10 ns ; PCin_tb<='1', '0' after 10 ns ;
383          readin_tb<='1', '0' after 10 ns ; MDRin_tb<='1', '0' after 10 ns ;
384          --Mdatain_tb<=x"294c0000";
385
386      when T2 =>
387
388          MDRoutEn_tb<='1', '0' after 10 ns ;
389
390      when T3 =>
391          --
392          r2out_tb<='1', '0' after 10 ns;

```

```
393         reny_tb<='1', '0' after 10 ns ; --readin_tb<='1', '0' after 10 ns
        ;MDRin_tb<='1', '0' after 10 ns ;
394         when T4 =>
395             r3out_tb<='1', '0' after 10 ns ; renz_tb<='1', '0' after 10 ns ;
396             operation_tb<="0100";
397             when T5 =>
398                 zlowout_tb<='1', '0' after 10 ns ; ren1_tb<='1', '0' after 10 ns ;
                --operation_tb<="1111"; renz_tb<='0';
399             when others =>
400
401     end case;
402 end process;
```

