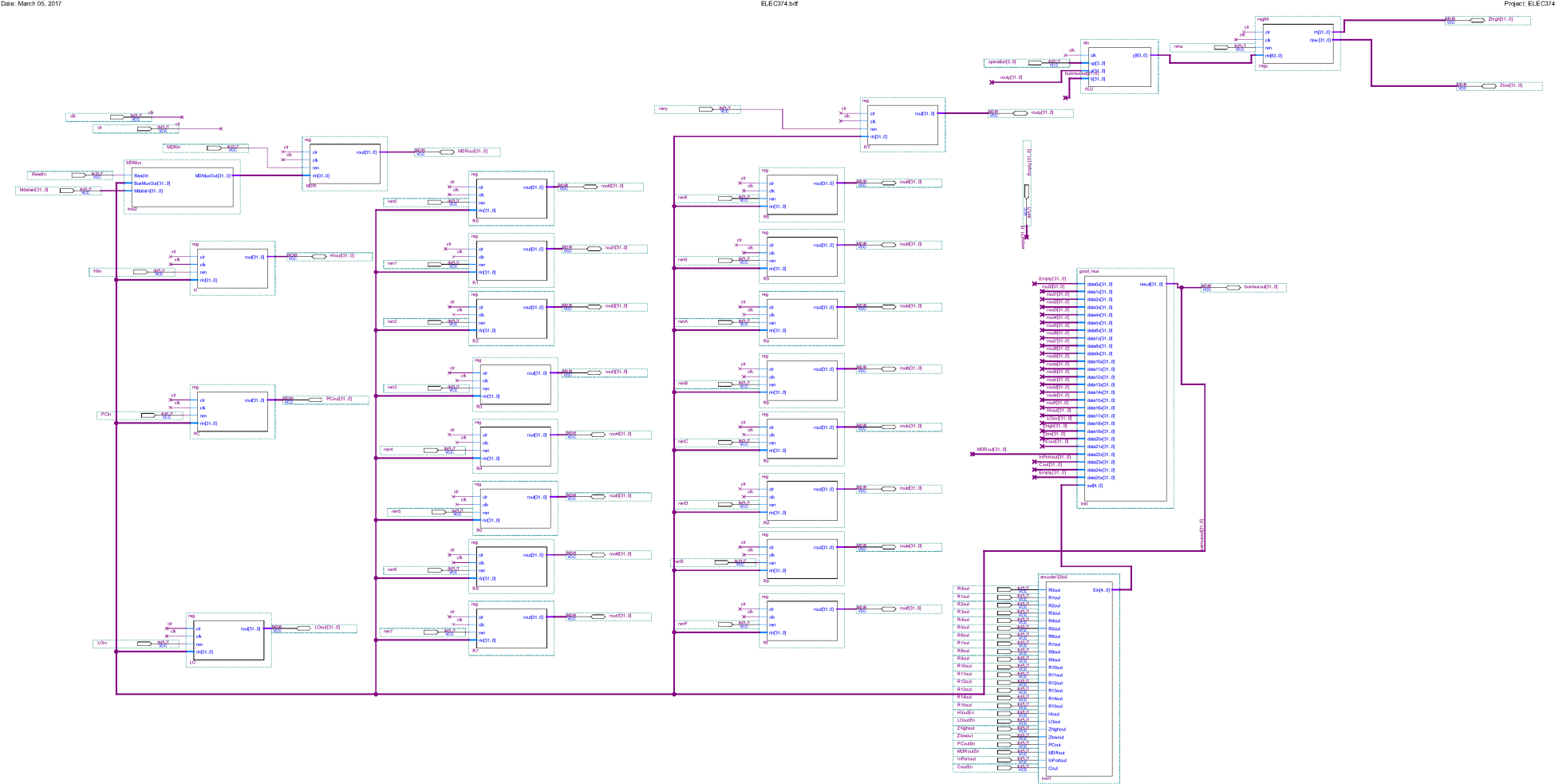
Temi sucks dick Schematic of Phase 1 project



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 refer to the 13 -- applicable agreement for further details.

14

1. -- PROGRAM "Quartus II 64-Bit"
2. -- VERSION "Version 13.0.1 Build 232 06/12/2013 Service Pack 1 SJ Full Version"
3. -- 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 | | |
| PORT  ( |  |  |
| ren0 : | IN | STD\_LOGIC; |
| ren1 : | IN | STD\_LOGIC; |
| ren2 : | IN | STD\_LOGIC; |
| ren3 : | IN | STD\_LOGIC; |
| ren4 : | IN | STD\_LOGIC; |
| ren5 : | IN | STD\_LOGIC; |
| ren6 : | IN | STD\_LOGIC; |
| ren7 : | IN | STD\_LOGIC; |
| ren8 : | IN | STD\_LOGIC; |
| ren9 : | IN | STD\_LOGIC; |
| renA : | IN | STD\_LOGIC; |
| renB : | IN | STD\_LOGIC; |
| renC : | IN | STD\_LOGIC; |
| renD : | IN | STD\_LOGIC; |
| renE : | IN | STD\_LOGIC; |
| renF : | IN | STD\_LOGIC; |
| clr : | IN | STD\_LOGIC; |
| clk : | IN | STD\_LOGIC; |
| R0out : | IN | STD\_LOGIC; |
| R1out : | IN | STD\_LOGIC; |
| R2out : | IN | STD\_LOGIC; |
| R3out : | IN | STD\_LOGIC; |
| R4out : | IN | STD\_LOGIC; |
| R5out : | IN | STD\_LOGIC; |
| R6out : | IN | STD\_LOGIC; |
| R7out : | IN | STD\_LOGIC; |
| 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; | | |

1. InPortout : IN STD\_LOGIC;
2. HIin : IN STD\_LOGIC;
3. PCin : IN STD\_LOGIC;
4. LOin : IN STD\_LOGIC;
5. MDRin : IN STD\_LOGIC;
6. HIoutEn : IN STD\_LOGIC;
7. LOoutEn : IN STD\_LOGIC;
8. PCoutEn : IN STD\_LOGIC;
9. MDRoutEn : IN STD\_LOGIC;
10. CoutEn : IN STD\_LOGIC;
11. reny : IN STD\_LOGIC;
12. renz : IN STD\_LOGIC;
13. ReadIn : IN STD\_LOGIC;
14. busmuxout : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
15. 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)

1. );
2. END ELEC374;

105

106 ARCHITECTURE bdf\_type OF ELEC374 IS

107

1. COMPONENT alu
2. PORT(clk : IN STD\_LOGIC;
3. 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

1. PORT(clr : IN STD\_LOGIC;
2. clk : IN STD\_LOGIC;
3. ren : IN STD\_LOGIC;
4. rin : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0); 122 rout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

123 );

# 124 END COMPONENT;

125

1. COMPONENT good\_mux
2. 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

1. COMPONENT encoder32to5
2. PORT(R0out : IN STD\_LOGIC;
3. R1out : IN STD\_LOGIC;
4. R2out : IN STD\_LOGIC;
5. R3out : IN STD\_LOGIC;
6. R4out : IN STD\_LOGIC;
7. R5out : IN STD\_LOGIC;
8. R6out : IN STD\_LOGIC;
9. R7out : IN STD\_LOGIC;
10. R8out : IN STD\_LOGIC;
11. R9out : IN STD\_LOGIC;
12. R10out : IN STD\_LOGIC;
13. R11out : IN STD\_LOGIC;
14. R12out : IN STD\_LOGIC;
15. R13out : IN STD\_LOGIC;
16. R14out : IN STD\_LOGIC;
17. R15out : IN STD\_LOGIC;
18. HIout : IN STD\_LOGIC;
19. LOout : IN STD\_LOGIC;
20. Zhighout : IN STD\_LOGIC;
21. Zlowout : IN STD\_LOGIC;
22. PCout : IN STD\_LOGIC;
23. MDRout : IN STD\_LOGIC;
24. InPortout : IN STD\_LOGIC;
25. Cout : IN STD\_LOGIC;
26. Sin : OUT STD\_LOGIC\_VECTOR(4 DOWNTO 0)
27. );

# 185 END COMPONENT;

186

## 187 COMPONENT mdmux

1. PORT(ReadIn : IN STD\_LOGIC;
2. BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);
3. Mdatain : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0); 191 MDMuxOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

192 );

# 193 END COMPONENT;

194

## 195 COMPONENT reg64

1. PORT(clr : IN STD\_LOGIC;
2. clk : IN STD\_LOGIC;
3. ren : IN STD\_LOGIC;
4. rin : IN STD\_LOGIC\_VECTOR(63 DOWNTO 0); 200 rh : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0); 201 rlow : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

202 );

# 203 END COMPONENT;

204

## 205 SIGNAL Cout : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

1. SIGNAL InPortout0 : STD\_LOGIC;
2. SIGNAL InPortout1 : STD\_LOGIC;
3. SIGNAL InPortout10 : STD\_LOGIC;
4. SIGNAL InPortout11 : STD\_LOGIC;
5. SIGNAL InPortout12 : STD\_LOGIC;
6. SIGNAL InPortout13 : STD\_LOGIC;
7. SIGNAL InPortout14 : STD\_LOGIC;
8. SIGNAL InPortout15 : STD\_LOGIC;
9. SIGNAL InPortout16 : STD\_LOGIC;
10. SIGNAL InPortout17 : STD\_LOGIC;
11. SIGNAL InPortout18 : STD\_LOGIC;
12. SIGNAL InPortout19 : STD\_LOGIC; 218 SIGNAL InPortout2 : STD\_LOGIC;
13. SIGNAL InPortout20 : STD\_LOGIC;
14. SIGNAL InPortout21 : STD\_LOGIC;
15. SIGNAL InPortout22 : STD\_LOGIC;
16. SIGNAL InPortout23 : STD\_LOGIC;
17. SIGNAL InPortout24 : STD\_LOGIC;
18. SIGNAL InPortout25 : STD\_LOGIC;
19. SIGNAL InPortout26 : STD\_LOGIC;
20. SIGNAL InPortout27 : STD\_LOGIC;
21. SIGNAL InPortout28 : STD\_LOGIC;
22. SIGNAL InPortout29 : STD\_LOGIC; 229 SIGNAL InPortout3 : STD\_LOGIC;
23. SIGNAL InPortout30 : STD\_LOGIC;
24. SIGNAL InPortout31 : STD\_LOGIC;
25. SIGNAL InPortout4 : STD\_LOGIC;
26. SIGNAL InPortout5 : STD\_LOGIC;
27. SIGNAL InPortout6 : STD\_LOGIC;
28. SIGNAL InPortout7 : STD\_LOGIC;
29. SIGNAL InPortout8 : STD\_LOGIC;
30. SIGNAL InPortout9 : STD\_LOGIC;
31. SIGNAL SYNTHESIZED\_WIRE\_0 : STD\_LOGIC\_VECTOR(4 DOWNTO 0);
32. SIGNAL SYNTHESIZED\_WIRE\_1 : STD\_LOGIC\_VECTOR(31 DOWNTO 0); 240 SIGNAL SYNTHESIZED\_WIRE\_2 : STD\_LOGIC\_VECTOR(63 DOWNTO 0); 241

242 SIGNAL GDFX\_TEMP\_SIGNAL\_0 : STD\_LOGIC\_VECTOR(31 DOWNTO 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

1. b2v\_ALU : alu
2. PORT MAP(clk => clk,
3. a => routy,
4. b => busmuxout,
5. op => operation,
6. y => SYNTHESIZED\_WIRE\_2);

255

256

1. b2v\_H : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => HIin,
5. rin => busmuxout,
6. rout => HIout);

263

264

1. b2v\_inst : good\_mux
2. PORT MAP(data0x => Empty,
3. data10x => rout9,
4. data11x => routa,
5. data12x => routb,
6. data13x => routc,
7. data14x => routd,
8. data15x => route,
9. data16x => routf,
10. data17x => HIout,
11. data18x => LOout,
12. data19x => Zhigh,
13. data1x => rout0,
14. data20x => Zlow,
15. data21x => PCout,
16. data22x => MDRout,
17. data23x => GDFX\_TEMP\_SIGNAL\_0,
18. data24x => Cout,
19. data25x => Empty,
20. data2x => rout1,
21. data3x => rout2,
22. data4x => rout3,
23. data5x => rout4,
24. data6x => rout5,
25. data7x => rout6,
26. data8x => rout7,
27. data9x => rout8,
28. sel => SYNTHESIZED\_WIRE\_0,
29. result => busmuxout);

294

295

1. b2v\_inst1 : encoder32to5
2. PORT MAP(R0out => R0out,
3. R1out => R1out,
4. R2out => R2out,
5. R3out => R3out,
6. R4out => R4out,
7. R5out => R5out,
8. R6out => R6out,
9. R7out => R7out,
10. R8out => R8out,
11. R9out => R9out,
12. R10out => R10out,
13. R11out => R11out,
14. R12out => R12out,
15. R13out => R13out,
16. R14out => R14out,
17. R15out => R15out,
18. HIout => HIoutEn,
19. LOout => LOoutEn,
20. Zhighout => Zhighout,
21. Zlowout => Zlowout, 317 PCout => PCoutEn,
22. MDRout => MDRoutEn,
23. InPortout => InPortout,
24. Cout => CoutEn,
25. Sin => SYNTHESIZED\_WIRE\_0);

322

323

1. b2v\_inst2 : mdmux
2. PORT MAP(ReadIn => ReadIn,
3. BusMuxOut => busmuxout,
4. Mdatain => Mdatain,
5. MDMuxOut => SYNTHESIZED\_WIRE\_1);

329

330

1. b2v\_LO : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => LOin,
5. rin => busmuxout,
6. rout => LOout);

337

338

1. b2v\_MDR : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => MDRin,
5. rin => SYNTHESIZED\_WIRE\_1,
6. rout => MDRout);

345

346

1. b2v\_PC : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => PCin,
5. rin => busmuxout,
6. rout => PCout);

353

354

1. b2v\_R0 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren0,
5. rin => busmuxout,
6. rout => rout0);

361

362

1. b2v\_R1 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren1,
5. rin => busmuxout,
6. rout => rout1);

369

370

1. b2v\_R2 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren2,
5. rin => busmuxout,
6. rout => rout2);

377

378

1. b2v\_R3 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren3,
5. rin => busmuxout,
6. rout => rout3);

385

386

1. b2v\_R4 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren4,
5. rin => busmuxout,
6. rout => rout4);

393

394

1. b2v\_R5 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren5,
5. rin => busmuxout,
6. rout => rout5);

401

402

1. b2v\_R6 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren6,
5. rin => busmuxout,
6. rout => rout6);

409

410

1. b2v\_R7 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren7,
5. rin => busmuxout,
6. rout => rout7);

417

418

1. b2v\_R8 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren8,
5. rin => busmuxout,
6. rout => rout8);

425

426

1. b2v\_R9 : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => ren9,
5. rin => busmuxout,
6. rout => rout9);

433

434

1. b2v\_Ra : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => renA,
5. rin => busmuxout,
6. rout => routa);

441

442

1. b2v\_Rb : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => renB,
5. rin => busmuxout,
6. rout => routb);

449

450

1. b2v\_Rc : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => renC,
5. rin => busmuxout,
6. rout => routc);

457

458

1. b2v\_Rd : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => renD,
5. rin => busmuxout,
6. rout => routd);

465

466

1. b2v\_Re : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => renE,
5. rin => busmuxout,
6. rout => route);

473

474

1. b2v\_regz : reg64
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => renz,
5. rin => SYNTHESIZED\_WIRE\_2,
6. rh => Zhigh,
7. rlow => Zlow);

482

483

1. b2v\_Rf : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => renF,
5. rin => busmuxout,
6. rout => routf);

490

491

1. b2v\_RY : reg
2. PORT MAP(clr => clr,
3. clk => clk,
4. ren => reny,
5. rin => busmuxout,
6. 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

1. entity badd32 is
2. 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

1. prod : out std\_logic\_vector(1 downto 0)); -- 2 bits of product
2. end entity badd32;
3. library IEEE;

# 12 use IEEE.std\_logic\_1164.all;

1. entity add32 is -- simple 32 bit ripple carry adder
2. port(a : in std\_logic\_vector(31 downto 0); 15 b : in std\_logic\_vector(31 downto 0); 16 cin : in std\_logic;

# 17 sum : out std\_logic\_vector(31 downto 0);

1. cout : out std\_logic);
2. end entity add32;
3. 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 |

1. end entity bmul32;
2. library IEEE;
3. use IEEE.std\_logic\_1164.all;
4. entity fadd is -- full adder stage, interface

# 30 port(a : in std\_logic; 31 b : in std\_logic; 32 cin : in std\_logic; 33 s : out std\_logic;

1. cout : out std\_logic);
2. end entity fadd;

36

37 architecture circuits of badd32 is

# 38 subtype word is std\_logic\_vector(31 downto 0);

39 signal bb : word; 40 signal psum : word; 41 signal b\_bar : word;

42 signal two\_b : word; 43 signal two\_b\_bar : word;

# 44 signal cout : std\_logic; 45 signal cin : std\_logic; 46 signal topbit : std\_logic; 47 signal topout : std\_logic; 48 signal nc1 : std\_logic;

1. begin -- circuits of badd32
2. b\_bar <= not b;
3. two\_b <= b(30 downto 0) & '0';
4. two\_b\_bar <= not two\_b;
5. bb <= b when a="001" or a="010" -- 5-input mux
6. else two\_b when a="011"
7. else two\_b\_bar when a="100" -- cin=1 56 else b\_bar when a="101" or a="110" -- cin=1
8. else x"00000000";
9. cin <= '1' when a="100" or a="101" or a="110"
10. else '0';
11. topbit <= b(31) when a="001" or a="010" or a="011"
12. else b\_bar(31) when a="100" or a="101" or a="110"
13. else '0';

63

1. a1: entity WORK.add32 port map(sum\_in, bb, cin, psum, cout);
2. a2: entity WORK.fadd port map(sum\_in(31), topbit, cout, topout, nc1); 66
3. sum\_out(29 downto 0) <= psum(31 downto 2);
4. sum\_out(31) <= topout;
5. sum\_out(30) <= topout;
6. prod <= psum(1 downto 0);
7. end architecture circuits; -- of badd32

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 downto 0); 76 subtype word is std\_logic\_vector(31 downto 0);

1. type ary is array(0 to 15) of word;
2. signal s : ary; -- temp sums
3. begin -- circuits of bmul32
4. mul0 <= a(1 downto 0) & '0';
5. a0: entity WORK.badd32 port map(
6. mul0, b, zer, s( 0), p( 1 downto 0));
7. a1: entity WORK.badd32 port map(
8. a(3 downto 1), b, s( 0), s( 1), p( 3 downto 2));
9. a2: entity WORK.badd32 port map(
10. a(5 downto 3), b, s( 1), s( 2), p( 5 downto 4));
11. a3: entity WORK.badd32 port map(
12. a(7 downto 5), b, s( 2), s( 3), p( 7 downto 6));
13. a4: entity WORK.badd32 port map(
14. a(9 downto 7), b, s( 3), s( 4), p( 9 downto 8));
15. a5: entity WORK.badd32 port map(
16. a(11 downto 9), b, s( 4), s( 5), p(11 downto 10));
17. a6: entity WORK.badd32 port map(
18. a(13 downto 11), b, s( 5), s( 6), p(13 downto 12));
19. a7: entity WORK.badd32 port map(
20. a(15 downto 13), b, s( 6), s( 7), p(15 downto 14));
21. a8: entity WORK.badd32 port map(
22. a(17 downto 15), b, s( 7), s( 8), p(17 downto 16));
23. a9: entity WORK.badd32 port map(
24. a(19 downto 17), b, s( 8), s( 9), p(19 downto 18));
25. a10: entity WORK.badd32 port map(
26. a(21 downto 19), b, s( 9), s(10), p(21 downto 20));
27. a11: entity WORK.badd32 port map(
28. 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(

1. a(29 downto 27), b, s(13), s(14), p(29 downto 28));
2. a15: entity WORK.badd32 port map(
3. a(31 downto 29), b, s(14), p(63 downto 32) , p(31 downto 30));
4. end architecture circuits; -- of bmul32

114

115

116

117

1. architecture circuits of fadd is
2. begin
3. s <= a xor b xor cin after 1 ps;
4. cout <= (a and b) or (a and cin) or (b and cin) after 1 ps;
5. end architecture circuits; -- of fadd

123

124

125

126

127 architecture circuits of add32 is

# 128 signal c : std\_logic\_vector(0 to 30);

1. begin
2. a0: entity WORK.fadd port map(a(0), b(0), cin, sum(0), c(0));
3. stage: for I in 1 to 30 generate
4. as: entity WORK.fadd port map(a(I), b(I), c(I-1) , sum(I), c(I));
5. end generate stage;
6. a31: entity WORK.fadd port map(a(31), b(31), c(30) , sum(31), cout);
7. 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 downt 0)

1. );
2. end entity reg;

12

1. architecture behavior of reg is
2. begin
3. process (clk,clr,ren,rin)
4. begin
5. if (clr= '1') then
6. rout <= x"00000000";
7. elsif ( clk='1') then
8. if ren = '1' then
9. rout <= rin;
10. end if;
11. end if;
12. end process;
13. 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;

1. R5out: in std\_logic;
2. R6out: in std\_logic;
3. R7out: in std\_logic;
4. R8out: in std\_logic;
5. R9out: in std\_logic;
6. R10out: in std\_logic;
7. R11out: in std\_logic;
8. R12out: in std\_logic;
9. R13out: in std\_logic;
10. R14out: in std\_logic;
11. R15out: in std\_logic;
12. HIout: in std\_logic;
13. LOout: in std\_logic;
14. Zhighout: in std\_logic;
15. Zlowout: in std\_logic;
16. PCout: in std\_logic;
17. MDRout: in std\_logic;
18. InPortout: in std\_logic;

# 28 Cout: in std\_logic; 29 Sin: out std\_logic\_vector(4 downto 0)

1. );
2. end entity encoder32to5;

32

1. architecture behavior of encoder32to5 is
2. begin

35

36

1. Sin <= "00001" when R0out = '1' else
2. "00010" when R1out = '1' else
3. "00011" when R2out = '1' else
4. "00100" when R3out = '1' else
5. "00101" when R4out = '1' else
6. "00110" when R5out = '1' else
7. "00111" when R6out = '1' else
8. "01000" when R7out = '1' else
9. "01001" when R8out = '1' else
10. "01010" when R9out = '1' else
11. "01011" when R10out = '1' else
12. "01100" when R11out = '1' else
13. "01101" when R12out = '1' else
14. "01110" when R13out = '1' else
15. "01111" when R14out = '1' else
16. "10000" when R15out = '1' else
17. "10001" when HIout = '1' else
18. "10010" when LOout = '1' else
19. "10011" when Zhighout = '1' else
20. "10100" when Zlowout = '1' else
21. "10101" when PCout = '1' else
22. "10110" when MDRout = '1' else
23. "10111" when InPortout = '1' else
24. "11000" when Cout = '1' else
25. "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 DOWNTO 0); 8 data1x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 9 data2x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 10 data3x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 11 data4x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 12 data5x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 13 data6x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 14 data7x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 15 data8x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 16 data9x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 17 data10x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 18 data11x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 19 data12x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 20 data13x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 21 data14x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 22 data15x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 23 data16x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 24 data17x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 25 data18x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 26 data19x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 27 data20x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 28 data21x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 29 data22x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 30 data23x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 31 data24x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 32 data25x : IN STD\_LOGIC\_VECTOR (31 DOWNTO 0); 33 sel : IN STD\_LOGIC\_VECTOR (4 DOWNTO 0); 34 result : OUT STD\_LOGIC\_VECTOR (31 DOWNTO 0)

## 35 ); 36 end good\_mux;

37 -------------------------------------------------------------------------------38

1. architecture behavioral of good\_mux is
2. 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)

1. begin
2. 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;

1. end case;
2. 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)

1. );
2. 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 std\_logic\_vector(31 downto 0); -- multiplicand 20 p : out std\_logic\_vector(63 downto 0) -- product

21 );

# 22 end component bmul32;

23

1. component CLAdder
2. 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

1. (
2. 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 DOWNTO 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

1. signal enum\_op : op\_type;
2. signal c\_in: std\_logic :='0';
3. signal c\_outH: std\_logic;
4. 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

1. CLAhigh: CLAdder port map (a(15 downto 0),
2. b (15 downto 0),
3. c\_in,
4. a\_plus\_b(15 downto 0),
5. c\_outL
6. );

72

1. CLAlow: CLAdder port map (a(31 downto 16),
2. b (31 downto 16),
3. c\_outL,
4. a\_plus\_b(31 downto 16),
5. c\_outH
6. );
7. a\_plus\_b(32)<=c\_outH;
8. booth: bmul32 port map (a,
9. b,
10. a\_mul\_b
11. );

84

1. div32: divider port map (b,
2. a,
3. quotient,
4. remainder
5. );

90

91 a\_minus\_b <= std\_logic\_vector(signed(a(a'high) & a) - signed(b(b'high) & b)); 92

1. a\_div\_b <= remainder & quotient;
2. a\_shr\_b <= to\_stdlogicvector(to\_bitvector(a) srl to\_integer( unsigned(b)));
3. a\_shl\_b <= to\_stdlogicvector(to\_bitvector(a) sll to\_integer( 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

1. when "0000" => enum\_op <= op\_and;
2. when "0001" => enum\_op <= op\_add;
3. when "0010" => enum\_op <= op\_sub;
4. when "0011" => enum\_op <= op\_mul;
5. when "0100" => enum\_op <= op\_div;

106 when "0101" => enum\_op <= op\_or;

1. when "0110" => enum\_op <= op\_shr;
2. when "0111" => enum\_op <= op\_shl;
3. when "1000" => enum\_op <= op\_ror;
4. when "1001" => enum\_op <= op\_rol;
5. when "1010" => enum\_op <= op\_neg;
6. when "1011" => enum\_op <= op\_not;
7. when others => enum\_op <= op\_nop;

# 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

1. if clk='1' then
2. case enum\_op is
3. when op\_add => reg64 <= (63 downto a\_plus\_b'length => '0')& a\_plus\_b;
4. when op\_sub => reg64 <= (63 downto a\_minus\_b'length => '0') & a\_minus\_b;
5. when op\_and => reg64 <= (63 downto 32 => '0') & (a and b);
6. when op\_mul => reg64 <= (63 downto a\_mul\_b'length => '0') & a\_mul\_b;
7. 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);

1. when op\_shr => reg64 <= (63 downto a\_shr\_b'length => '0') & a\_shr\_b;
2. when op\_shl => reg64 <= (63 downto a\_shl\_b'length => '0') & a\_shl\_b;
3. when op\_ror => reg64 <= (63 downto a\_ror\_b'length => '0') & a\_ror\_b;
4. when op\_rol => reg64 <= (63 downto a\_rol\_b'length => '0') & a\_rol\_b;
5. when op\_neg => reg64 <= (63 downto 32 => '0') & std\_logic\_vector(unsigned(not (a)) + 1);
6. when op\_not => reg64 <= (63 downto 32 => '0') & (not(To\_X01(a)));

133

1. when op\_nop =>
2. reg(32) <= '0';
3. end case;
4. end if;
5. end process;
6. process(a\_plus\_b,a\_minus\_b,a\_mul\_b,a\_div\_b,a\_shr\_b,reg64)
7. begin
8. 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

1. entity CLAdder is
2. port(Xin: in std\_logic\_vector(15 downto 0);
3. Yin: in std\_logic\_vector(15 downto 0);
4. Cin: in std\_logic;
5. Sum: out std\_logic\_vector (15 downto 0);
6. Cout: out std\_logic
7. );
8. end CLAdder;

13

14

15 architecture behaviour of CLAdder is

16

1. signal h\_sum : std\_logic\_vector(15 downto 0);
2. signal G: std\_logic\_vector(15 downto 0);
3. signal P: std\_logic\_vector(15 downto 0);
4. signal CarryIn: std\_logic\_vector(15 downto 1);

21

1. begin
2. h\_sum<= Xin xor Yin;
3. G <= Xin and Yin;
4. P <= Xin or Yin;

26

1. process (G,P, CarryIn, h\_sum, Cin)
2. begin
3. CarryIn(1) <= G(0) or (P(0) and Cin);
4. inst: for i in 1 to 14 loop
5. CarryIn(i+1) <= G(i) or (P(i) and CarryIn(i));
6. end loop;
7. Cout <= G(15) or (P(15) and CarryIn(15));
8. end process;
9. Sum(0)<= h\_sum(0) xor Cin;
10. 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

1. entity MDMux is
2. port ( ReadIn: in std\_logic;
3. BusMuxOut: in std\_logic\_vector(31 downto 0);
4. Mdatain: in std\_logic\_vector(31 downto 0);
5. MDMuxOut: out std\_logic\_vector(31 downto 0)
6. );
7. end MDMux;

13

14

1. architecture behavior of MDMux is
2. begin
3. process (ReadIn,BusMuxOut,Mdatain)
4. begin
5. if (ReadIn= '1') then
6. MDMuxOut <= Mdatain;
7. else
8. MDMuxOut <= BusMuxOut;
9. end if;
10. end process;
11. 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\_vector(63 downto 0); 9 rh,rlow: out std\_logic\_vector(31 downto 0)

1. );
2. end entity reg64;

12

1. architecture behavior of reg64 is
2. begin
3. process (clk,clr, rin, ren)
4. begin
5. if (clr = '1') then
6. rh <= x"00000000";
7. rlow <= x"00000000";
8. elsif ( clk='1') then
9. if ren = '1' then
10. rh <= rin(63 downto 32);
11. rlow <= rin(31 downto 0);
12. end if;
13. end if;

26

1. end process;
2. end behaviour;

# Code for a datapath (reg.vhd)

# 1. library ieee;

1. use ieee.std\_logic\_1164.all;
2. use ieee.std\_logic\_unsigned.all;

# 4 library work;

5

1. ENTITY testbench IS
2. END testbench;

8

1. ARCHITECTURE testbench\_arch OF testbench IS
2. --initialization and declaration of inputs
3. signal ren0\_tb: std\_logic;
4. signal ren1\_tb: std\_logic;
5. signal ren2\_tb: std\_logic;
6. signal ren3\_tb: std\_logic;
7. signal ren4\_tb: std\_logic;
8. signal ren5\_tb: std\_logic;
9. signal ren6\_tb: std\_logic;
10. signal ren7\_tb: std\_logic;
11. signal ren8\_tb: std\_logic;
12. signal ren9\_tb: std\_logic;
13. signal renA\_tb: std\_logic;
14. signal renB\_tb: std\_logic;
15. signal renC\_tb: std\_logic;
16. signal renD\_tb: std\_logic;
17. signal renE\_tb: std\_logic;
18. signal renF\_tb: std\_logic; 27 signal clr\_tb : std\_logic;
19. signal clk\_tb : std\_logic;
20. signal R0out\_tb : std\_logic;
21. signal R1out\_tb : std\_logic;
22. signal R2out\_tb : std\_logic;
23. signal R3out\_tb : std\_logic;
24. signal R4out\_tb : std\_logic;
25. signal R5out\_tb : std\_logic;
26. signal R6out\_tb : std\_logic;
27. signal R7out\_tb : std\_logic;
28. signal R8out\_tb : std\_logic;
29. signal R9out\_tb : std\_logic;
30. signal R10out\_tb :std\_logic;
31. signal R11out\_tb :std\_logic;
32. signal R12out\_tb :std\_logic;
33. signal R13out\_tb :std\_logic;
34. signal R14out\_tb :std\_logic;
35. signal R15out\_tb :std\_logic;
36. signal Zhighout\_tb: std\_logic; 46 signal Zlowout\_tb :std\_logic;
37. signal InPortout\_tb: std\_logic;
38. signal HIin\_tb :std\_logic;
39. signal PCin\_tb :std\_logic;
40. signal LOin\_tb :std\_logic;
41. signal Empty\_tb : STD\_LOGIC\_VECTOR(31 DOWNTO 0);
42. signal readin\_tb :std\_logic;
43. signal MDRin\_tb : std\_logic;
44. signal Mdatain\_tb :STD\_LOGIC\_VECTOR(31 DOWNTO 0);
45. signal HIoutEn\_tb :std\_logic;
46. signal LOoutEn\_tb :std\_logic;
47. signal PCoutEn\_tb :std\_logic;
48. signal MDRoutEn\_tb: std\_logic;
49. signal CoutEn\_tb :std\_logic; 60 signal reny\_tb :std\_logic;
50. signal operation\_tb: STD\_LOGIC\_VECTOR(3 DOWNTO 0);
51. signal renz\_tb :std\_logic;
52. signal rout2\_tb :std\_logic\_vector(31 downto 0);
53. signal rout3\_tb :std\_logic\_vector(31 downto 0);
54. signal rout4\_tb :std\_logic\_vector(31 downto 0);
55. signal rout5\_tb :std\_logic\_vector(31 downto 0);
56. signal rout6\_tb :std\_logic\_vector(31 downto 0);
57. signal rout7\_tb :std\_logic\_vector(31 downto 0);
58. signal rout8\_tb :std\_logic\_vector(31 downto 0);
59. signal rout9\_tb :std\_logic\_vector(31 downto 0);
60. signal routa\_tb :std\_logic\_vector(31 downto 0);
61. signal routb\_tb :std\_logic\_vector(31 downto 0);
62. signal routc\_tb :std\_logic\_vector(31 downto 0);
63. signal routd\_tb :std\_logic\_vector(31 downto 0);
64. signal route\_tb :std\_logic\_vector(31 downto 0);
65. signal routf\_tb :std\_logic\_vector(31 downto 0);
66. signal rout0\_tb :std\_logic\_vector(31 downto 0);
67. 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);
68. signal Zhigh\_tb :std\_logic\_vector(31 downto 0);
69. signal LOout\_tb :std\_logic\_vector(31 downto 0);
70. signal HIout\_tb :std\_logic\_vector(31 downto 0);
71. signal MDRout\_tb :std\_logic\_vector(31 downto 0);
72. signal routy\_tb : STD\_LOGIC\_VECTOR(31 DOWNTO 0);
73. signal Zlow\_tb :std\_logic\_vector(31 downto 0);

87

88

1. TYPE State IS(default, Reg\_load1, Reg\_load2, Reg\_load3, T0, T1, T2, T3, T4, T5);
2. SIGNAL Present\_state: State := default; 91

92 component ELEC374

# 93 PORT

1. (
2. ren0 : IN STD\_LOGIC;
3. ren1 : IN STD\_LOGIC;
4. ren2 : IN STD\_LOGIC;
5. ren3 : IN STD\_LOGIC;
6. ren4 : IN STD\_LOGIC;
7. ren5 : IN STD\_LOGIC;
8. ren6 : IN STD\_LOGIC;
9. ren7 : IN STD\_LOGIC;
10. ren8 : IN STD\_LOGIC;
11. ren9 : IN STD\_LOGIC;
12. renA : IN STD\_LOGIC;
13. renB : IN STD\_LOGIC;
14. renC : IN STD\_LOGIC;
15. renD : IN STD\_LOGIC;
16. renE : IN STD\_LOGIC;
17. renF : IN STD\_LOGIC;

111 clr : IN STD\_LOGIC;

112 clk : IN STD\_LOGIC;

113 R0out : IN STD\_LOGIC;

1. R1out : IN STD\_LOGIC;
2. R2out : IN STD\_LOGIC;
3. R3out : IN STD\_LOGIC;
4. R4out : IN STD\_LOGIC;
5. R5out : IN STD\_LOGIC;
6. R6out : IN STD\_LOGIC;
7. R7out : IN STD\_LOGIC;
8. R8out : IN STD\_LOGIC;
9. R9out : IN STD\_LOGIC;
10. R10out : IN STD\_LOGIC;
11. R11out : IN STD\_LOGIC;
12. R12out : IN STD\_LOGIC;
13. R13out : IN STD\_LOGIC;
14. R14out : IN STD\_LOGIC;
15. R15out : IN STD\_LOGIC; 129 Zhighout : IN STD\_LOGIC; 130 Zlowout : IN STD\_LOGIC;
16. InPortout : IN STD\_LOGIC;
17. HIin : IN STD\_LOGIC;
18. PCin : IN STD\_LOGIC;

134

1. LOin : IN STD\_LOGIC;
2. Empty : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);
3. ReadIn : IN STD\_LOGIC;
4. MDRin : IN STD\_LOGIC;
5. Mdatain : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);
6. HIoutEn : IN STD\_LOGIC;
7. LOoutEn : IN STD\_LOGIC;
8. PCoutEn : IN STD\_LOGIC; 143 MDRoutEn : IN STD\_LOGIC; 144 CoutEn : IN STD\_LOGIC;
9. reny : IN STD\_LOGIC;
10. operation: IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);
11. renz : IN STD\_LOGIC;
12. rout2 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
13. rout3 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
14. rout4 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
15. rout5 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
16. rout6 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
17. rout7 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
18. rout8 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
19. rout9 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
20. routa : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
21. routb : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
22. routc : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
23. routd : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
24. route : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
25. routf : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
26. rout0 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
27. rout1 : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
28. busmuxout : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
29. PCout : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
30. 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);

1. HIout : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
2. MDRout : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);
3. routy : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

172

1. );
2. 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, | |

1. Zhighout =>Zhighout\_tb,
2. Zlowout =>Zlowout\_tb,
3. InPortout =>InPortout\_tb,

|  |  |
| --- | --- |
| 216 HIin | =>HIin\_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, |

1. MDRoutEn =>MDRoutEn\_tb,
2. CoutEn =>CoutEn\_tb,
3. reny =>reny\_tb,
4. 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, |

1. busmuxout =>busmuxout\_tb,
2. PCout =>PCout\_tb,

|  |  |
| --- | --- |
| 249 Zhigh | =>Zhigh\_tb, |
| 250 Zlow | =>Zlow\_tb, |
| 251 LOout | =>LOout\_tb, |
| 252 HIout | =>HIout\_tb, |
| 253 MDRout | =>MDRout\_tb, |

1. routy => routy\_tb
2. );

256

1. clk\_process : process
2. begin
3. clk\_tb<='0','1' after 10 ns;

# 260 wait for 20 ns;

261 end process clk\_process;

262

263 process (clk\_tb)

264

# 265 begin

1. if (clk\_tb'event and clk\_tb ='1') then
2. case Present\_state is
3. when Default =>
4. Present\_state<=Reg\_load1;

270 when reg\_load1 =>

271 present\_state<=Reg\_load2;

272 when reg\_load2 =>

1. present\_state<=Reg\_load3;
2. when reg\_load3 =>
3. present\_state<=T0;
4. when T0 =>

277 present\_state<=T1;

278 when T1 =>

279 present\_state<=T2;

280 when T2 =>

281 present\_state<=T3;

282 when T3 =>

283 present\_state<=T4;

284 when T4 =>

285 present\_state<=T5;

# 286 when others =>

# 287 end case;

# 288 end if; 289 end process;

290

291 Process(present\_state)

# 292 begin

293 CASE present\_state IS 294 WHEN default =>

1. PCin\_tb<='0'; Zlowout\_tb<='0'; mdroutEn\_tb<='0';
2. R1Out\_tb<='0';R2Out\_tb<='0';R3Out\_tb<='0';
3. PCin\_tb <= '0'; MDRin\_tb <= '0';
4. ren1\_tb<='0'; readin\_tb<='0';
5. clr\_tb <= '1', '0' after 4 ns;
6. renz\_tb <= '0';
7. ren0\_tb <= '0';
8. ren1\_tb <= '0';
9. ren2\_tb <= '0';
10. ren3\_tb <= '0';
11. ren4\_tb <= '0';
12. ren5\_tb <= '0';
13. ren6\_tb <= '0';
14. ren7\_tb <= '0';
15. ren8\_tb <= '0';
16. ren9\_tb <= '0';
17. renA\_tb <= '0';
18. renB\_tb <= '0';
19. renC\_tb <= '0';
20. renD\_tb <= '0';
21. renE\_tb <= '0';
22. renF\_tb <= '0';
23. R0out\_tb <= '0';
24. R1out\_tb <= '0';
25. R2out\_tb <= '0';
26. R3out\_tb <= '0';
27. R4out\_tb <= '0';
28. R5out\_tb <= '0';
29. R6out\_tb <= '0';
30. R7out\_tb <= '0';
31. R8out\_tb <= '0';
32. R9out\_tb <= '0';
33. R10out\_tb <= '0';
34. R11out\_tb <= '0';
35. R12out\_tb <= '0';
36. R13out\_tb <= '0';
37. R14out\_tb <= '0';
38. R15out\_tb <= '0';
39. Zhighout\_tb <= '0';
40. InPortout\_tb <= '0';
41. HIin\_tb <= '0';

336

1. Loin\_tb <='0';
2. HIoutEn\_tb <= '0';

339 LOoutEn\_tb<='0';

1. PcoutEn\_tb <= '0';
2. CoutEn\_tb<='0'; 342 reny\_tb <= '0';
3. operation\_tb <= "0000";
4. Mdatain\_tb<=x"00000012";

345

1. --routy\_tb <= x"00000000";
2. empty\_tb <= x"00000000";
3. operation\_tb<="0000";

349

350

351

1. when reg\_load1 =>
2. mdatain\_tb<= x"00006666";
3. readin\_tb<='1','0' after 10 ns;
4. mdrin\_tb<='1','0'after 10 ns;
5. mdroutEn\_tb<='1', '0' after 10 ns ;

357

358 ren1\_tb<='1', '0' after 10 ns ;

359

1. when reg\_load2 =>
2. mdatain\_tb <= x"00000015";
3. readin\_tb<='1','0' after 10 ns;
4. mdrin\_tb<='1','0'after 10 ns;
5. mdroutEn\_tb<='1', '0' after 10 ns ;

365

366

367 ren2\_tb<='1', '0' after 10 ns ;

368

1. when reg\_load3 =>
2. mdatain\_tb <= x"00000002";
3. readin\_tb<='1','0' after 10 ns;
4. mdrin\_tb<='1','0'after 10 ns;
5. mdroutEn\_tb<='1', '0' after 10 ns ;
6. ren3\_tb<='1', '0' after 10 ns ;

375

1. when T0 =>
2. PCoutEn\_tb<='1', '0' after 10 ns ;
3. renz\_tb<='1', '0' after 10 ns ;

379

380

1. when T1 =>
2. Zlowout\_tb<='1', '0' after 10 ns ; PCin\_tb<='1', '0' after 10 ns ;
3. readin\_tb<='1', '0' after 10 ns ; MDRin\_tb<='1', '0' after 10 ns ;
4. --Mdatain\_tb<=x"294c0000";

385

386 when T2 =>

387

388 MDRoutEn\_tb<='1', '0' after 10 ns ;

389

1. when T3 =>
2. --
3. r2out\_tb<='1', '0' after 10 ns;
4. reny\_tb<='1', '0' after 10 ns ; --readin\_tb<='1', '0' after 10 ns

;MDRin\_tb<='1', '0' after 10 ns ;

1. when T4 =>
2. r3out\_tb<='1', '0' after 10 ns ; renz\_tb<='1', '0' after 10 ns ;
3. operation\_tb<="0100";
4. when T5 =>
5. zlowout\_tb<='1', '0' after 10 ns ; ren1\_tb<='1', '0' after 10 ns ;

--operation\_tb<="1111"; renz\_tb<='0';

1. when others =>

400

# 401 end case;

# 402 end process;