Contents

[Code 3](#_Toc36822258)

[32 Bit Register 3](#_Toc36822259)

[GP Registers (R0-R15) 4](#_Toc36822260)

[HI/LO/Y/IR/MAR/In.Port/Out.Port Registers 10](#_Toc36822261)

[Carry Lookahead Adder 11](#_Toc36822262)

[Hardware Incrementor (increments by 1) 12](#_Toc36822263)

[32 Bit 2-to-1 Mux 14](#_Toc36822264)

[PC Register 14](#_Toc36822265)

[MDR Register 17](#_Toc36822266)

[Z Register 19](#_Toc36822267)

[32-to-5 Encoder 20](#_Toc36822268)

[32-to-1 Mux 26](#_Toc36822269)

[Bidirectional Bus 29](#_Toc36822270)

[Negator 37](#_Toc36822271)

[Custom Shift Logic (Preforms Arithmetic and Roll-over shifts) 39](#_Toc36822272)

[Booth’s Multiplier 41](#_Toc36822273)

[Bit Restore Divider 43](#_Toc36822274)

[ALU Address Encoder 44](#_Toc36822275)

[ALU 47](#_Toc36822276)

[Datapath 54](#_Toc36822277)

[Block Schematics 71](#_Toc36822278)

[Datapath 71](#_Toc36822279)

[Testbench Files 71](#_Toc36822280)

[AND/OR/ADD/SUB 71](#_Toc36822281)

[MUL/DIV 81](#_Toc36822282)

[SHR/SHL/ROR/ROL 90](#_Toc36822283)

[NOT/NEG 100](#_Toc36822284)

[Output Waveforms 109](#_Toc36822285)

[AND 109](#_Toc36822286)

[OR 109](#_Toc36822287)

[NOT 109](#_Toc36822288)

[NEG 109](#_Toc36822289)

[ADD 110](#_Toc36822290)

[SUB 110](#_Toc36822291)

[MUL 110](#_Toc36822292)

[DIV 111](#_Toc36822293)

[SHR 111](#_Toc36822294)

[SHL 111](#_Toc36822295)

[ROR 112](#_Toc36822296)

[ROL 112](#_Toc36822297)

# Code

## 32 Bit Register

library ieee;

use ieee.std\_logic\_1164.all;

entity reg32 is

port (

clk : in std\_logic;

reset\_n : in std\_logic;

en : in std\_logic;

d : in std\_logic\_vector(31 downto 0);

q : out std\_logic\_vector(31 downto 0)

);

end entity;

architecture behavior of reg32 is

begin

process (clk, reset\_n)

begin

if (reset\_n = '0') then

q <= (others => '0');

elsif (falling\_edge(clk)) then

if (en = '1') then

q <= d;

end if;

end if;

end process;

end architecture;

## GP Registers (R0-R15)

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY gp IS

PORT

(

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

R0in : IN STD\_LOGIC;

R1in : IN STD\_LOGIC;

R2in : IN STD\_LOGIC;

R3in : IN STD\_LOGIC;

R4in : IN STD\_LOGIC;

R5in : IN STD\_LOGIC;

R6in : IN STD\_LOGIC;

R7in : IN STD\_LOGIC;

R8in : IN STD\_LOGIC;

R9in : IN STD\_LOGIC;

R10in : IN STD\_LOGIC;

R11in : IN STD\_LOGIC;

R12in : IN STD\_LOGIC;

R13in : IN STD\_LOGIC;

R14in : IN STD\_LOGIC;

R15in : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R0in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R1in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R2in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R3in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R4in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R5in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R6in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R7in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R8in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R9in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R10in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R11in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R12in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R13in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R14in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R15in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END gp;

ARCHITECTURE bdf\_type OF gp IS

COMPONENT reg32

PORT(clk : IN STD\_LOGIC;

reset\_n : IN STD\_LOGIC;

en : IN STD\_LOGIC;

d : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

q : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

BEGIN

b2v\_inst : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R0in,

d => BusMuxOut,

q => BusMuxIn\_R0in);

b2v\_inst1 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R1in,

d => BusMuxOut,

q => BusMuxIn\_R1in);

b2v\_inst10 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R10in,

d => BusMuxOut,

q => BusMuxIn\_R10in);

b2v\_inst11 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R11in,

d => BusMuxOut,

q => BusMuxIn\_R11in);

b2v\_inst12 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R12in,

d => BusMuxOut,

q => BusMuxIn\_R12in);

b2v\_inst13 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R13in,

d => BusMuxOut,

q => BusMuxIn\_R13in);

b2v\_inst14 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R14in,

d => BusMuxOut,

q => BusMuxIn\_R14in);

b2v\_inst15 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R15in,

d => BusMuxOut,

q => BusMuxIn\_R15in);

b2v\_inst2 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R2in,

d => BusMuxOut,

q => BusMuxIn\_R2in);

b2v\_inst3 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R3in,

d => BusMuxOut,

q => BusMuxIn\_R3in);

b2v\_inst4 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R4in,

d => BusMuxOut,

q => BusMuxIn\_R4in);

b2v\_inst5 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R5in,

d => BusMuxOut,

q => BusMuxIn\_R5in);

b2v\_inst6 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R6in,

d => BusMuxOut,

q => BusMuxIn\_R6in);

b2v\_inst7 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R7in,

d => BusMuxOut,

q => BusMuxIn\_R7in);

b2v\_inst8 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R8in,

d => BusMuxOut,

q => BusMuxIn\_R8in);

b2v\_inst9 : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => R9in,

d => BusMuxOut,

q => BusMuxIn\_R9in);

END bdf\_type;

## HI/LO/Y/IR/MAR/In.Port/Out.Port Registers

**Note that each component has the same VHDL code except for the names of the 32 bit register output and the load enable signal**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY hi IS

PORT

(

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_HI : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END hi;

ARCHITECTURE bdf\_type OF hi IS

COMPONENT reg32

PORT(clk : IN STD\_LOGIC;

reset\_n : IN STD\_LOGIC;

en : IN STD\_LOGIC;

d : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

q : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

BEGIN

b2v\_inst : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => HIin, -- This would be different for each component

d => BusMuxOut,

q => BusMuxIn\_HI); -- This would be different for each component

END bdf\_type;

## Carry Lookahead Adder

library ieee ;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity cl\_addr is

port (

A : in std\_logic\_vector (31 downto 0);

B : in std\_logic\_vector (31 downto 0);

Cin : in std\_logic;

S : out std\_logic\_vector (31 downto 0);

Cout : out std\_logic

);

end entity;

architecture behavior of cl\_addr is

signal sum\_internal : std\_logic\_vector (31 downto 0);

signal carry\_generate : std\_logic\_vector (31 downto 0);

signal carry\_propagate : std\_logic\_vector (31 downto 0);

signal carry\_in\_internal : std\_logic\_vector (31 downto 1);

begin

sum\_internal <= A xor B;

carry\_generate <= A and B;

carry\_propagate <= A or B;

process (carry\_generate,carry\_propagate,carry\_in\_internal)

begin

carry\_in\_internal(1) <= carry\_generate(0) or (carry\_propagate(0) and Cin);

inst: for i in 1 to 30 loop

carry\_in\_internal(i+1) <= carry\_generate(i) or (carry\_propagate(i) and carry\_in\_internal(i));

end loop;

Cout <= carry\_generate(31) or (carry\_propagate(31) and carry\_in\_internal(31));

end process;

S(0) <= sum\_internal(0) xor Cin;

S(31 downto 1) <= sum\_internal(31 downto 1) xor carry\_in\_internal(31 downto 1);

end architecture;

## Hardware Incrementor (increments by 1)

library ieee ;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity incrementer is

port(

input: in std\_logic\_vector (31 downto 0);

output: out std\_logic\_vector (31 downto 0)

);

end entity;

architecture behavior of incrementer is

component cl\_addr

port (

A : in std\_logic\_vector (31 downto 0);

B : in std\_logic\_vector (31 downto 0);

Cin : in std\_logic;

S : out std\_logic\_vector (31 downto 0);

Cout : out std\_logic

);

end component;

begin

cl\_addr\_1 : cl\_addr port map (

input,

"00000000000000000000000000000000",

'1',

output

);

end architecture;

## 32 Bit 2-to-1 Mux

library ieee ;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity mux2to1\_32 is

port(

SEL: in std\_logic; -- selection bit input

X0\_in: in std\_logic\_vector(31 downto 0); -- first input

X1\_in: in std\_logic\_vector(31 downto 0); -- second input

Y: out std\_logic\_vector(31 downto 0) -- output

);

end entity mux2to1\_32;

architecture mux2to1\_32\_arch of mux2to1\_32 is

begin

Y <= X1\_in when (SEL = '1') else X0\_in;

end architecture mux2to1\_32\_arch;

## PC Register

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY pc IS

PORT

(

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_PC : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END pc;

ARCHITECTURE bdf\_type OF pc IS

COMPONENT reg32

PORT(clk : IN STD\_LOGIC;

reset\_n : IN STD\_LOGIC;

en : IN STD\_LOGIC;

d : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

q : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT incrementer

PORT(input : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

output : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT mux2to1\_32

PORT(sel : IN STD\_LOGIC;

X0\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X1\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

Y : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

SIGNAL SYNTHESIZED\_WIRE\_0 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_1 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_2 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BEGIN

BusMuxIn\_PC <= SYNTHESIZED\_WIRE\_1;

b2v\_inst : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => PCin,

d => SYNTHESIZED\_WIRE\_0,

q => SYNTHESIZED\_WIRE\_1);

b2v\_inst1 : incrementer

PORT MAP(input => SYNTHESIZED\_WIRE\_1,

output => SYNTHESIZED\_WIRE\_2);

b2v\_inst2 : mux2to1\_32

PORT MAP(sel => IncPC,

X0\_in => BusMuxOut,

X1\_in => SYNTHESIZED\_WIRE\_2,

Y => SYNTHESIZED\_WIRE\_0);

END bdf\_type;

## MDR Register

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY mdr IS

PORT

(

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemDataIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_MDR : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemIn\_MDR : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END mdr;

ARCHITECTURE bdf\_type OF mdr IS

COMPONENT mux2to1\_32

PORT(SEL : IN STD\_LOGIC;

X0\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X1\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

Y : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT reg32

PORT(clk : IN STD\_LOGIC;

reset\_n : IN STD\_LOGIC;

en : IN STD\_LOGIC;

d : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

q : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

SIGNAL SYNTHESIZED\_WIRE\_0 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_1 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BEGIN

BusMuxIn\_MDR <= SYNTHESIZED\_WIRE\_1;

MemIn\_MDR <= SYNTHESIZED\_WIRE\_1;

b2v\_read\_select : mux2to1\_32

PORT MAP(SEL => Read,

X0\_in => BusMuxOut,

X1\_in => MemDataIn,

Y => SYNTHESIZED\_WIRE\_0);

b2v\_mdr\_reg : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => MDRin,

d => SYNTHESIZED\_WIRE\_0,

q => SYNTHESIZED\_WIRE\_1);

END bdf\_type;

## Z Register

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY z IS

PORT

(

clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Zhi\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

Zlo\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zhi : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zlow : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END z;

ARCHITECTURE bdf\_type OF z IS

COMPONENT reg32

PORT(clk : IN STD\_LOGIC;

reset\_n : IN STD\_LOGIC;

en : IN STD\_LOGIC;

d : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

q : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

BEGIN

b2v\_ZHi : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => Zin,

d => Zhi\_in,

q => BusMuxIn\_Zhi);

b2v\_ZLo : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => Zin,

d => Zlo\_in,

q => BusMuxIn\_Zlow);

END bdf\_type;

## 32-to-5 Encoder

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity encoder\_32to5 is

port(

x0 : in std\_logic := '0';

x1 : in std\_logic := '0';

x2 : in std\_logic := '0';

x3 : in std\_logic := '0';

x4 : in std\_logic := '0';

x5 : in std\_logic := '0';

x6 : in std\_logic := '0';

x7 : in std\_logic := '0';

x8 : in std\_logic := '0';

x9 : in std\_logic := '0';

x10 : in std\_logic := '0';

x11 : in std\_logic := '0';

x12 : in std\_logic := '0';

x13 : in std\_logic := '0';

x14 : in std\_logic := '0';

x15 : in std\_logic := '0';

x16 : in std\_logic := '0';

x17 : in std\_logic := '0';

x18 : in std\_logic := '0';

x19 : in std\_logic := '0';

x20 : in std\_logic := '0';

x21 : in std\_logic := '0';

x22 : in std\_logic := '0';

x23 : in std\_logic := '0';

x24 : in std\_logic := '0';

x25 : in std\_logic := '0';

x26 : in std\_logic := '0';

x27 : in std\_logic := '0';

x28 : in std\_logic := '0';

x29 : in std\_logic := '0';

x30 : in std\_logic := '0';

x31 : in std\_logic := '0';

sel0 : out std\_logic;

sel1 : out std\_logic;

sel2 : out std\_logic;

sel3 : out std\_logic;

sel4 : out std\_logic

);

end entity;

architecture behavior of encoder\_32to5 is

signal a : std\_logic\_vector(31 downto 0):= (others => '0');

signal b : std\_logic\_vector(4 downto 0) := (others => '0');

begin

a(0) <= x0;

a(1) <= x1;

a(2) <= x2;

a(3) <= x3;

a(4) <= x4;

a(5) <= x5;

a(6) <= x6;

a(7) <= x7;

a(8) <= x8;

a(9) <= x9;

a(10) <= x10;

a(11) <= x11;

a(12) <= x12;

a(13) <= x13;

a(14) <= x14;

a(15) <= x15;

a(16) <= x16;

a(17) <= x17;

a(18) <= x18;

a(19) <= x19;

a(20) <= x20;

a(21) <= x21;

a(22) <= x22;

a(23) <= x23;

a(24) <= x24;

a(25) <= x25;

a(26) <= x26;

a(27) <= x27;

a(28) <= x28;

a(29) <= x29;

a(30) <= x30;

a(31) <= x31;

process(a)

begin

if (a="00000000000000000000000000000001") then

b <= "00000";

elsif (a="00000000000000000000000000000010") then

b <= "00001";

elsif (a="00000000000000000000000000000100") then

b <= "00010";

elsif (a="00000000000000000000000000001000") then

b <= "00011";

elsif (a="00000000000000000000000000010000") then

b <= "00100";

elsif (a="00000000000000000000000000100000") then

b <= "00101";

elsif (a="00000000000000000000000001000000") then

b <= "00110";

elsif (a="00000000000000000000000010000000") then

b <= "00111";

elsif (a="00000000000000000000000100000000") then

b <= "01000";

elsif (a="00000000000000000000001000000000") then

b <= "01001";

elsif (a="00000000000000000000010000000000") then

b <= "01010";

elsif (a="00000000000000000000100000000000") then

b <= "01011";

elsif (a="00000000000000000001000000000000") then

b <= "01100";

elsif (a="00000000000000000010000000000000") then

b <= "01101";

elsif (a="00000000000000000100000000000000") then

b <= "01110";

elsif (a="00000000000000001000000000000000") then

b <= "01111";

elsif (a="00000000000000010000000000000000") then

b <= "10000";

elsif (a="00000000000000100000000000000000") then

b <= "10001";

elsif (a="00000000000001000000000000000000") then

b <= "10010";

elsif (a="00000000000010000000000000000000") then

b <= "10011";

elsif (a="00000000000100000000000000000000") then

b <= "10100";

elsif (a="00000000001000000000000000000000") then

b <= "10101";

elsif (a="00000000010000000000000000000000") then

b <= "10110";

elsif (a="00000000100000000000000000000000") then

b <= "10111";

elsif (a="00000001000000000000000000000000") then

b <= "11000";

elsif (a="00000010000000000000000000000000") then

b <= "11001";

elsif (a="00000100000000000000000000000000") then

b <= "11010";

elsif (a="00001000000000000000000000000000") then

b <= "11011";

elsif (a="00010000000000000000000000000000") then

b <= "11100";

elsif (a="00100000000000000000000000000000") then

b <= "11101";

elsif (a="01000000000000000000000000000000") then

b <= "11110";

elsif (a="10000000000000000000000000000000") then

b <= "11111";

else

b <= "ZZZZZ";

end if;

end process;

sel0 <= b(0);

sel1 <= b(1);

sel2 <= b(2);

sel3 <= b(3);

sel4 <= b(4);

end architecture;

## 32-to-1 Mux

library ieee ;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

---------------------------------------------------

entity mux32to1\_32 is

port(

SEL0 : in std\_logic; -- Selection vector bit 0

SEL1 : in std\_logic; -- 1

SEL2 : in std\_logic; -- 2

SEL3 : in std\_logic; -- 3

SEL4 : in std\_logic; -- 4

X0\_in : in std\_logic\_vector(31 downto 0) := (others => '0'); -- Input vector 0

X1\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 1

X2\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 2

X3\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 3

X4\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 4

X5\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 5

X6\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 6

X7\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 7

X8\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 8

X9\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 9

X10\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 10

X11\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 11

X12\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 12

X13\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 13

X14\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 14

X15\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 15

X16\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 16

X17\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 17

X18\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 18

X19\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 19

X20\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 20

X21\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 21

X22\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 22

X23\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 23

X24\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 24

X25\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 25

X26\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 26

X27\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 27

X28\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 28

X29\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 29

X30\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 30

X31\_in : in std\_logic\_vector(31 downto 0):= (others => '0'); -- 31

Y\_out : out std\_logic\_vector(31 downto 0)); -- Output

end entity mux32to1\_32;

----------------------------------------------------

architecture mux32to1\_32\_arch of mux32to1\_32 is

signal SEL : std\_logic\_vector(4 downto 0);

begin

SEL(0) <= SEL0;

SEL(1) <= SEL1;

SEL(2) <= SEL2;

SEL(3) <= SEL3;

SEL(4) <= SEL4;

Y\_out <= X0\_in when (SEL = "00000") else

X1\_in when (SEL = "00001") else

X2\_in when (SEL = "00010") else

X3\_in when (SEL = "00011") else

X4\_in when (SEL = "00100") else

X5\_in when (SEL = "00101") else

X6\_in when (SEL = "00110") else

X7\_in when (SEL = "00111") else

X8\_in when (SEL = "01000") else

X9\_in when (SEL = "01001") else

X10\_in when (SEL = "01010") else

X11\_in when (SEL = "01011") else

X12\_in when (SEL = "01100") else

X13\_in when (SEL = "01101") else

X14\_in when (SEL = "01110") else

X15\_in when (SEL = "01111") else

X16\_in when (SEL = "10000") else

X17\_in when (SEL = "10001") else

X18\_in when (SEL = "10010") else

X19\_in when (SEL = "10011") else

X20\_in when (SEL = "10100") else

X21\_in when (SEL = "10101") else

X22\_in when (SEL = "10110") else

X23\_in when (SEL = "10111") else

X24\_in when (SEL = "11000") else

X25\_in when (SEL = "11001") else

X26\_in when (SEL = "11010") else

X27\_in when (SEL = "11011") else

X28\_in when (SEL = "11100") else

X29\_in when (SEL = "11101") else

X30\_in when (SEL = "11110") else X31\_in;

end architecture mux32to1\_32\_arch;

## Bidirectional Bus

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY bidirectional\_bus IS

PORT

(

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;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

BusMuxIn\_R0in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R1in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R2in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R3in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R4in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R5in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R6in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R7in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R8in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R9in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R10in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R11in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R12in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R13in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R14in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R15in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_PC : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_HI : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_LO : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zhigh : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zlow : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_InPort : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_MDR : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

c\_sign\_extended : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END bidirectional\_bus;

ARCHITECTURE bdf\_type OF bidirectional\_bus IS

COMPONENT encoder\_32to5

PORT(x0 : IN STD\_LOGIC;

x1 : IN STD\_LOGIC;

x2 : IN STD\_LOGIC;

x3 : IN STD\_LOGIC;

x4 : IN STD\_LOGIC;

x5 : IN STD\_LOGIC;

x6 : IN STD\_LOGIC;

x7 : IN STD\_LOGIC;

x8 : IN STD\_LOGIC;

x9 : IN STD\_LOGIC;

x10 : IN STD\_LOGIC;

x11 : IN STD\_LOGIC;

x12 : IN STD\_LOGIC;

x13 : IN STD\_LOGIC;

x14 : IN STD\_LOGIC;

x15 : IN STD\_LOGIC;

x16 : IN STD\_LOGIC;

x17 : IN STD\_LOGIC;

x18 : IN STD\_LOGIC;

x19 : IN STD\_LOGIC;

x20 : IN STD\_LOGIC;

x21 : IN STD\_LOGIC;

x22 : IN STD\_LOGIC;

x23 : IN STD\_LOGIC;

x24 : IN STD\_LOGIC;

x25 : IN STD\_LOGIC;

x26 : IN STD\_LOGIC;

x27 : IN STD\_LOGIC;

x28 : IN STD\_LOGIC;

x29 : IN STD\_LOGIC;

x30 : IN STD\_LOGIC;

x31 : IN STD\_LOGIC;

sel0 : OUT STD\_LOGIC;

sel1 : OUT STD\_LOGIC;

sel2 : OUT STD\_LOGIC;

sel3 : OUT STD\_LOGIC;

sel4 : OUT STD\_LOGIC

);

END COMPONENT;

COMPONENT mux32to1\_32

PORT(

SEL0 : IN STD\_LOGIC;

SEL1 : IN STD\_LOGIC;

SEL2 : IN STD\_LOGIC;

SEL3 : IN STD\_LOGIC;

SEL4 : IN STD\_LOGIC;

X0\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X1\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X2\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X3\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X4\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X5\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X6\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X7\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X8\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X9\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X10\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X11\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X12\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X13\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X14\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X15\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X16\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X17\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X18\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X19\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X20\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X21\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X22\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X23\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X24\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X25\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X26\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X27\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X28\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X29\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X30\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

X31\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

Y\_out : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

SIGNAL SYNTHESIZED\_WIRE\_0 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_1 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_2 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_3 : STD\_LOGIC;

SIGNAL SYNTHESIZED\_WIRE\_4 : STD\_LOGIC;

BEGIN

b2v\_inst : encoder\_32to5

PORT MAP(x0 => R0out,

x1 => R1out,

x2 => R2out,

x3 => R3out,

x4 => R4out,

x5 => R5out,

x6 => R6out,

x7 => R7out,

x8 => R8out,

x9 => R9out,

x10 => R10out,

x11 => R11out,

x12 => R12out,

x13 => R13out,

x14 => R14out,

x15 => R15out,

x16 => HIout,

x17 => LOout,

x18 => Zhighout,

x19 => Zlowout,

x20 => PCout,

x21 => MDRout,

x22 => InPortout,

x23 => Cout,

x24 => '0',

x25 => '0',

x26 => '0',

x27 => '0',

x28 => '0',

x29 => '0',

x30 => '0',

x31 => '0',

sel0 => SYNTHESIZED\_WIRE\_0,

sel1 => SYNTHESIZED\_WIRE\_1,

sel2 => SYNTHESIZED\_WIRE\_2,

sel3 => SYNTHESIZED\_WIRE\_3,

sel4 => SYNTHESIZED\_WIRE\_4);

b2v\_inst1 : mux32to1\_32

PORT MAP(SEL0 => SYNTHESIZED\_WIRE\_0,

SEL1 => SYNTHESIZED\_WIRE\_1,

SEL2 => SYNTHESIZED\_WIRE\_2,

SEL3 => SYNTHESIZED\_WIRE\_3,

SEL4 => SYNTHESIZED\_WIRE\_4,

X0\_in => BusMuxIn\_R0in,

X1\_in => BusMuxIn\_R1in,

X2\_in => BusMuxIn\_R2in,

X3\_in => BusMuxIn\_R3in,

X4\_in => BusMuxIn\_R4in,

X5\_in => BusMuxIn\_R5in,

X6\_in => BusMuxIn\_R6in,

X7\_in => BusMuxIn\_R7in,

X8\_in => BusMuxIn\_R8in,

X9\_in => BusMuxIn\_R9in,

X10\_in => BusMuxIn\_R10in,

X11\_in => BusMuxIn\_R11in,

X12\_in => BusMuxIn\_R12in,

X13\_in => BusMuxIn\_R13in,

X14\_in => BusMuxIn\_R14in,

X15\_in => BusMuxIn\_R15in,

X16\_in => BusMuxIn\_HI,

X17\_in => BusMuxIn\_LO,

X18\_in => BusMuxIn\_Zhigh,

X19\_in => BusMuxIn\_Zlow,

X20\_in => BusMuxIn\_PC,

X21\_in => BusMuxIn\_MDR,

X22\_in => BusMuxIn\_InPort,

X23\_in => c\_sign\_extended,

X24\_in => "00000000000000000000000000000000",

X25\_in => "00000000000000000000000000000000",

X26\_in => "00000000000000000000000000000000",

X27\_in => "00000000000000000000000000000000",

X28\_in => "00000000000000000000000000000000",

X29\_in => "00000000000000000000000000000000",

X30\_in => "00000000000000000000000000000000",

X31\_in => "00000000000000000000000000000000",

Y\_out => BusMuxOut);

END bdf\_type;

## Negator

library ieee;

use ieee.std\_logic\_1164.all;

entity negator is

port (

neg\_in : in std\_logic\_vector(31 downto 0);

neg\_out : out std\_logic\_vector(31 downto 0)

);

end entity;

architecture behavior of negator is

component incrementer

port (

input : in std\_logic\_vector (31 downto 0);

output : out std\_logic\_vector (31 downto 0)

);

end component incrementer;

signal temp : std\_logic\_vector(31 downto 0);

begin

inst : for i in 0 to 31 generate

temp(i) <= not neg\_in(i);

end generate;

incrementer\_neg : incrementer port map (

temp,

neg\_out

);

end architecture;

## Custom Shift Logic (Preforms Arithmetic and Roll-over shifts)

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity shifter is

port (

shift\_in : in std\_logic\_vector(31 downto 0);

n : in std\_logic\_vector(4 downto 0);

sel : in std\_logic\_vector(1 downto 0);

shift\_out : out std\_logic\_vector(31 downto 0);

roll\_out : out std\_logic

);

end entity;

architecture behavior of shifter is

signal shifted : std\_logic\_vector(31 downto 0);

signal roll\_bit : std\_logic;

begin

process(shift\_in,n,sel)

variable n\_temp : integer;

variable left\_roll\_bit, right\_roll\_bit : std\_logic;

variable sign,zero : std\_logic\_vector(31 downto 0);

begin

n\_temp := to\_integer(unsigned(n));

sign := (others => shift\_in(31));

zero := (others => '0');

if(n\_temp /= 0) then

left\_roll\_bit := shift\_in(32-n\_temp);

right\_roll\_bit := shift\_in(n\_temp-1);

if(sel(0) = '0') then -- Right (1) vs left (0)

shifted(31 downto n\_temp) <= shift\_in((31-n\_temp) downto 0);

roll\_bit <= left\_roll\_bit;

if(sel(1) = '0') then -- Roll (1) vs shift (0)

shifted((n\_temp-1) downto 0) <= zero((n\_temp-1) downto 0);

else

shifted((n\_temp-1) downto 0) <= shift\_in(31 downto (31-n\_temp+1));

end if;

else

shifted((31-n\_temp) downto 0) <= shift\_in(31 downto n\_temp);

roll\_bit <= right\_roll\_bit;

if(sel(1) = '0') then

shifted(31 downto (31-n\_temp+1)) <= sign((n\_temp-1) downto 0);

else

shifted(31 downto (31-n\_temp+1)) <= shift\_in((n\_temp-1) downto 0);

end if;

end if;

end if;

end process;

shift\_out <= shifted;

roll\_out <= roll\_bit;

end architecture;

## Booth’s Multiplier

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use ieee.std\_logic\_unsigned.all;

entity multiplier is

port (

inputA : in std\_logic\_vector(31 downto 0);

inputB : in std\_logic\_vector(31 downto 0);

output: out std\_logic\_vector(63 downto 0)

);

end entity;

architecture behaviour of multiplier is

begin

process(inputA, inputB)

variable carry : std\_logic\_vector(32 downto 0);

variable temp : std\_logic\_vector(31 downto 0);

variable full, multiplicand : std\_logic\_vector(63 downto 0);

variable mul, mul2, nmul, nmul2 : std\_logic\_vector(31 downto 0);

begin

carry(32 downto 1) := inputB;

carry(0) := '0';

full := x"0000000000000000";

for i in 0 to 31 loop

if (inputA(i) = '1') then

nmul(i) := '0';

elsif (inputA(i) = '0') then

nmul(i) := '1';

end if;

end loop;

nmul := std\_logic\_vector(unsigned(nmul) + x"00000001");

mul2(31 downto 1) := inputA(30 downto 0);

mul2(0) := '0';

nmul2(31 downto 1) := nmul(30 downto 0);

nmul2(0) := '0';

for i in 0 to 15 loop

if (carry(2 downto 0) = "000") then

temp := x"00000000";

elsif (carry(2 downto 0) = "001") then

temp := inputA;

elsif (carry(2 downto 0) = "010") then

temp := inputA;

elsif (carry(2 downto 0) = "011") then

temp := mul2;

elsif (carry(2 downto 0) = "100") then

temp := nmul2;

elsif (carry(2 downto 0) = "101") then

temp := nmul;

elsif (carry(2 downto 0) = "110") then

temp := nmul;

elsif (carry(2 downto 0) = "111") then

temp := x"00000000";

end if;

multiplicand(63 downto 32) := temp;

multiplicand(31 downto 0) := x"00000000";

for j in 0 to (15 - i) loop

multiplicand(61 downto 0) := multiplicand(63 downto 2);

if (multiplicand(61) = '1') then

multiplicand(63 downto 62) := "11";

else

multiplicand(63 downto 62) := "00";

end if;

end loop;

full := full + multiplicand;

carry(30 downto 0) := carry(32 downto 2);

end loop;

output <= full;

end process;

end architecture;

## Bit Restore Divider

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use ieee.std\_logic\_unsigned.all;

entity divider is

port (

inputA : in std\_logic\_vector(31 downto 0);

inputB : in std\_logic\_vector(31 downto 0);

quotient: out std\_logic\_vector(31 downto 0);

remainder: out std\_logic\_vector(31 downto 0)

);

end entity;

architecture behaviour of divider is

begin

process(inputA, inputB)

variable full : std\_logic\_vector(63 downto 0);

begin

full(63 downto 32) := x"00000000"; -- restoring division

full(31 downto 0) := inputA;

for i in 0 to 31 loop

full(63 downto 1) := full(62 downto 0);

full(0) := '0';

full(63 downto 32) := full(63 downto 32) - inputB;

if (full(63) = '1') then

full(63 downto 32) := full(63 downto 32) + inputB;

else

full(0) := '1';

end if;

end loop;

quotient <= full(31 downto 0);

remainder <= full(63 downto 32);

end process;

end architecture;

## ALU Address Encoder

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity alu\_addr\_encoder is

port(

ANDin : in std\_logic := '0';

ORin : in std\_logic := '0';

NOTin : in std\_logic := '0';

NEGin : in std\_logic := '0';

ADDin : in std\_logic := '0';

SUBin : in std\_logic := '0';

MULin : in std\_logic := '0';

DIVin : in std\_logic := '0';

SRAin : in std\_logic := '0';

SLAin : in std\_logic := '0';

RORin : in std\_logic := '0';

ROLin : in std\_logic := '0';

ALU\_sel : out std\_logic\_vector(3 downto 0)

);

end entity;

architecture behavior of alu\_addr\_encoder is

signal a : std\_logic\_vector(11 downto 0) := (others => '0');

signal b : std\_logic\_vector(3 downto 0) := (others => '0');

begin

a(0) <= ANDin;

a(1) <= ORin;

a(2) <= NOTin;

a(3) <= NEGin;

a(4) <= ADDin;

a(5) <= SUBin;

a(6) <= MULin;

a(7) <= DIVin;

a(8) <= SRAin;

a(9) <= SLAin;

a(10) <= RORin;

a(11) <= ROLin;

process(a)

begin

if (a="000000000001") then

b <= "0000";

elsif (a="000000000010") then

b <= "0001";

elsif (a="000000000100") then

b <= "0010";

elsif (a="000000001000") then

b <= "0011";

elsif (a="000000010000") then

b <= "0100";

elsif (a="000000100000") then

b <= "0101";

elsif (a="000001000000") then

b <= "0110";

elsif (a="000010000000") then

b <= "0111";

elsif (a="000100000000") then

b <= "1000";

elsif (a="001000000000") then

b <= "1001";

elsif (a="010000000000") then

b <= "1010";

elsif (a="100000000000") then

b <= "1011";

else

b <= "ZZZZ";

end if;

end process;

ALU\_sel <= b;

end architecture;

## ALU

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity alu is

port (

alu\_A : in std\_logic\_vector(31 downto 0);

alu\_B : in std\_logic\_vector(31 downto 0);

SEL : in std\_logic\_vector(3 downto 0);

C\_hi : out std\_logic\_vector(31 downto 0);

C\_lo : out std\_logic\_vector(31 downto 0)

);

end entity;

architecture behavior of alu is

component cl\_addr

port (

A : in std\_logic\_vector (31 downto 0);

B : in std\_logic\_vector (31 downto 0);

Cin : in std\_logic;

S : out std\_logic\_vector (31 downto 0);

Cout : out std\_logic

);

end component cl\_addr;

component negator

port (

neg\_in : in std\_logic\_vector(31 downto 0);

neg\_out : out std\_logic\_vector(31 downto 0)

);

end component negator;

component shifter

port(

shift\_in : in std\_logic\_vector(31 downto 0);

n : in std\_logic\_vector(4 downto 0);

sel : in std\_logic\_vector(1 downto 0);

shift\_out : out std\_logic\_vector(31 downto 0);

roll\_out : out std\_logic

);

end component shifter;

component mux2to1\_32

port(

SEL: in std\_logic; -- selection bit input

X0\_in: in std\_logic\_vector(31 downto 0); -- first input

X1\_in: in std\_logic\_vector(31 downto 0); -- second input

Y: out std\_logic\_vector(31 downto 0) -- output

);

end component mux2to1\_32;

component multiplier is

port (

inputA : in std\_logic\_vector(31 downto 0);

inputB : in std\_logic\_vector(31 downto 0);

output: out std\_logic\_vector(63 downto 0)

);

end component;

component divider is

port (

inputA : in std\_logic\_vector(31 downto 0);

inputB : in std\_logic\_vector(31 downto 0);

quotient: out std\_logic\_vector(31 downto 0);

remainder: out std\_logic\_vector(31 downto 0)

);

end component;

signal NEGout : std\_logic\_vector (31 downto 0) := (others => '0');

signal ADDout : std\_logic\_vector (31 downto 0) := (others => '0');

signal SUBout : std\_logic\_vector (31 downto 0) := (others => '0');

signal MULout : std\_logic\_vector (63 downto 0) := (others => '0');

signal DIVout\_remainder : std\_logic\_vector (31 downto 0) := (others => '0');

signal DIVout\_quotient : std\_logic\_vector (31 downto 0) := (others => '0');

signal SRAout : std\_logic\_vector (31 downto 0) := (others => '0');

signal SLAout : std\_logic\_vector (31 downto 0) := (others => '0');

signal RORout : std\_logic\_vector (31 downto 0) := (others => '0');

signal ROLout : std\_logic\_vector (31 downto 0) := (others => '0');

begin

negator\_neg : negator port map (

alu\_B,

NEGout

);

cl\_addr\_add : cl\_addr port map (

alu\_A,

alu\_B,

'0',

ADDout

);

cl\_addr\_sub : cl\_addr port map (

alu\_A,

NEGout,

'0',

SUBout

);

shift\_left\_aritmetic : shifter port map (

alu\_A,

alu\_B(4 downto 0) ,

"00",

SLAout

);

shift\_right\_aritmetic : shifter port map (

alu\_A,

alu\_B(4 downto 0),

"01",

SRAout

);

rotate\_over\_left : shifter port map (

alu\_A,

alu\_B(4 downto 0),

"10",

ROLout

);

rotate\_over\_right : shifter port map (

alu\_A,

alu\_B(4 downto 0),

"11",

RORout

);

divide : divider port map (

alu\_A,

alu\_B,

DIVout\_quotient,

DIVout\_remainder

);

multiply : multiplier port map (

alu\_A,

alu\_B,

MULout

);

process(NEGout, ADDout, SUBout, SLAout, SRAout, ROLout, RORout, DIVout\_quotient, DIVout\_remainder, MULout)

begin

if (SEL = "0000") then -- AND

inst\_AND : for i in 0 to 31 loop

C\_lo(i) <= alu\_A(i) and alu\_B(i);

end loop;

C\_hi <= x"00000000";

elsif (SEL = "0001") then -- OR

inst\_OR : for i in 0 to 31 loop

C\_lo(i) <= alu\_A(i) or alu\_B(i);

end loop;

C\_hi <= x"00000000";

elsif (SEL = "0010") then -- NOT

inst\_NOT : for i in 0 to 31 loop

C\_lo(i) <= not alu\_B(i);

end loop;

C\_hi <= x"00000000";

elsif (SEL = "0011") then -- NEG

C\_lo <= NEGout;

C\_hi <= x"00000000";

elsif (SEL = "0100") then -- ADD

C\_lo <= ADDout;

C\_hi <= x"00000000";

elsif (SEL = "0101") then -- SUB

C\_lo <= SUBout;

C\_hi <= x"00000000";

elsif (SEL = "0110") then -- MUL

C\_lo <= MULout(31 downto 0);

C\_hi <= MULout(63 downto 32);

elsif (SEL = "0111") then -- DIV

C\_lo <= DIVout\_quotient;

C\_hi <= DIVout\_remainder;

elsif (SEL = "1000") then -- SRA

C\_lo <= SRAout;

C\_hi <= x"00000000";

elsif (SEL = "1001") then -- SLA

C\_lo <= SLAout;

C\_hi <= x"00000000";

elsif (SEL = "1010") then -- ROR

C\_lo <= RORout;

C\_hi <= x"00000000";

elsif (SEL = "1011") then -- ROL

C\_lo <= ROLout;

C\_hi <= x"00000000";

else

C\_lo <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";

C\_hi <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";

end if;

end process;

end architecture;

## Datapath

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY work;

ENTITY datapath IS

PORT

(

clk : IN STD\_LOGIC;

clear : 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;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

R0in : IN STD\_LOGIC;

R1in : IN STD\_LOGIC;

R2in : IN STD\_LOGIC;

R3in : IN STD\_LOGIC;

R4in : IN STD\_LOGIC;

R5in : IN STD\_LOGIC;

R6in : IN STD\_LOGIC;

R7in : IN STD\_LOGIC;

R8in : IN STD\_LOGIC;

R9in : IN STD\_LOGIC;

R10in : IN STD\_LOGIC;

R11in : IN STD\_LOGIC;

R12in : IN STD\_LOGIC;

R13in : IN STD\_LOGIC;

R14in : IN STD\_LOGIC;

R15in : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

MemDataIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

IRout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MARout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemDataOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutPortout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END datapath;

ARCHITECTURE bdf\_type OF datapath IS

COMPONENT hi

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_HI : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT reg32

PORT(clk : IN STD\_LOGIC;

reset\_n : IN STD\_LOGIC;

en : IN STD\_LOGIC;

d : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

q : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT gp

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

R0in : IN STD\_LOGIC;

R1in : IN STD\_LOGIC;

R2in : IN STD\_LOGIC;

R3in : IN STD\_LOGIC;

R4in : IN STD\_LOGIC;

R5in : IN STD\_LOGIC;

R6in : IN STD\_LOGIC;

R7in : IN STD\_LOGIC;

R8in : IN STD\_LOGIC;

R9in : IN STD\_LOGIC;

R10in : IN STD\_LOGIC;

R11in : IN STD\_LOGIC;

R12in : IN STD\_LOGIC;

R13in : IN STD\_LOGIC;

R14in : IN STD\_LOGIC;

R15in : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R0in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R10in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R11in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R12in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R13in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R14in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R15in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R1in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R2in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R3in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R4in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R5in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R6in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R7in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R8in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R9in : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT alu\_addr\_encoder

PORT(ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

ALU\_sel : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0)

);

END COMPONENT;

COMPONENT bidirectional\_bus

PORT(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;

PCout : IN STD\_LOGIC;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

BusMuxIn\_HI : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_InPort : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_LO : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_MDR : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_PC : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R0in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R10in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R11in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R12in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R13in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R14in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R15in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R1in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R2in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R3in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R4in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R5in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R6in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R7in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R8in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_R9in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zhigh : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zlow : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

c\_sign\_extended : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT alu

PORT(alu\_A : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

alu\_B : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SEL : IN STD\_LOGIC\_VECTOR(3 DOWNTO 0);

C\_hi : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

C\_lo : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT y

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

ALUIn\_Y : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT z

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

Zhi\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

Zlo\_in : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zhi : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_Zlow : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT lo

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_LO : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT mdr

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemDataIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_MDR : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemIn\_MDR : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

COMPONENT pc

PORT(clk : IN STD\_LOGIC;

clear : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

BusMuxOut : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusMuxIn\_PC : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT;

SIGNAL SYNTHESIZED\_WIRE\_1 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_3 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_4 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_5 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_6 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_7 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_8 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_9 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_10 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_11 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_12 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_13 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_14 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_15 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_16 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_17 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_18 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_19 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_20 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_21 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_22 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_23 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_24 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_25 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_27 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_29 : STD\_LOGIC\_VECTOR(3 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_30 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

SIGNAL SYNTHESIZED\_WIRE\_31 : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BEGIN

BusData <= SYNTHESIZED\_WIRE\_1;

b2v\_HI : hi

PORT MAP(clk => clk,

clear => clear,

HIin => HIin,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

BusMuxIn\_HI => SYNTHESIZED\_WIRE\_3);

b2v\_in\_port : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => InPortin,

d => SYNTHESIZED\_WIRE\_1,

q => SYNTHESIZED\_WIRE\_4);

b2v\_gp : gp

PORT MAP(clk => clk,

clear => clear,

R0in => R0in,

R1in => R1in,

R2in => R2in,

R3in => R3in,

R4in => R4in,

R5in => R5in,

R6in => R6in,

R7in => R7in,

R8in => R8in,

R9in => R9in,

R10in => R10in,

R11in => R11in,

R12in => R12in,

R13in => R13in,

R14in => R14in,

R15in => R15in,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

BusMuxIn\_R0in => SYNTHESIZED\_WIRE\_8,

BusMuxIn\_R10in => SYNTHESIZED\_WIRE\_9,

BusMuxIn\_R11in => SYNTHESIZED\_WIRE\_10,

BusMuxIn\_R12in => SYNTHESIZED\_WIRE\_11,

BusMuxIn\_R13in => SYNTHESIZED\_WIRE\_12,

BusMuxIn\_R14in => SYNTHESIZED\_WIRE\_13,

BusMuxIn\_R15in => SYNTHESIZED\_WIRE\_14,

BusMuxIn\_R1in => SYNTHESIZED\_WIRE\_15,

BusMuxIn\_R2in => SYNTHESIZED\_WIRE\_16,

BusMuxIn\_R3in => SYNTHESIZED\_WIRE\_17,

BusMuxIn\_R4in => SYNTHESIZED\_WIRE\_18,

BusMuxIn\_R5in => SYNTHESIZED\_WIRE\_19,

BusMuxIn\_R6in => SYNTHESIZED\_WIRE\_20,

BusMuxIn\_R7in => SYNTHESIZED\_WIRE\_21,

BusMuxIn\_R8in => SYNTHESIZED\_WIRE\_22,

BusMuxIn\_R9in => SYNTHESIZED\_WIRE\_23);

b2v\_alu\_addr\_encoder : alu\_addr\_encoder

PORT MAP(ANDin => ANDin,

ORin => ORin,

NOTin => NOTin,

NEGin => NEGin,

ADDin => ADDin,

SUBin => SUBin,

MULin => MULin,

DIVin => DIVin,

SRAin => SRAin,

SLAin => SLAin,

RORin => RORin,

ROLin => ROLin,

ALU\_sel => SYNTHESIZED\_WIRE\_29);

b2v\_bidirectional\_bus : bidirectional\_bus

PORT MAP(R0out => R0out,

R1out => R1out,

R2out => R2out,

R3out => R3out,

R4out => R4out,

R5out => R5out,

R6out => R6out,

R7out => R7out,

R8out => R8out,

R9out => R9out,

R10out => R10out,

R11out => R11out,

R12out => R12out,

R13out => R13out,

R14out => R14out,

R15out => R15out,

PCout => PCout,

HIout => HIout,

LOout => LOout,

Zhighout => Zhighout,

Zlowout => Zlowout,

MDRout => MDRout,

Cout => Cout,

InPortout => InPortout,

BusMuxIn\_HI => SYNTHESIZED\_WIRE\_3,

BusMuxIn\_InPort => SYNTHESIZED\_WIRE\_4,

BusMuxIn\_LO => SYNTHESIZED\_WIRE\_5,

BusMuxIn\_MDR => SYNTHESIZED\_WIRE\_6,

BusMuxIn\_PC => SYNTHESIZED\_WIRE\_7,

BusMuxIn\_R0in => SYNTHESIZED\_WIRE\_8,

BusMuxIn\_R10in => SYNTHESIZED\_WIRE\_9,

BusMuxIn\_R11in => SYNTHESIZED\_WIRE\_10,

BusMuxIn\_R12in => SYNTHESIZED\_WIRE\_11,

BusMuxIn\_R13in => SYNTHESIZED\_WIRE\_12,

BusMuxIn\_R14in => SYNTHESIZED\_WIRE\_13,

BusMuxIn\_R15in => SYNTHESIZED\_WIRE\_14,

BusMuxIn\_R1in => SYNTHESIZED\_WIRE\_15,

BusMuxIn\_R2in => SYNTHESIZED\_WIRE\_16,

BusMuxIn\_R3in => SYNTHESIZED\_WIRE\_17,

BusMuxIn\_R4in => SYNTHESIZED\_WIRE\_18,

BusMuxIn\_R5in => SYNTHESIZED\_WIRE\_19,

BusMuxIn\_R6in => SYNTHESIZED\_WIRE\_20,

BusMuxIn\_R7in => SYNTHESIZED\_WIRE\_21,

BusMuxIn\_R8in => SYNTHESIZED\_WIRE\_22,

BusMuxIn\_R9in => SYNTHESIZED\_WIRE\_23,

BusMuxIn\_Zhigh => SYNTHESIZED\_WIRE\_24,

BusMuxIn\_Zlow => SYNTHESIZED\_WIRE\_25,

c\_sign\_extended => x"00000000",

BusMuxOut => SYNTHESIZED\_WIRE\_1);

b2v\_y : y

PORT MAP(clk => clk,

clear => clear,

Yin => Yin,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

ALUIn\_Y => SYNTHESIZED\_WIRE\_27);

b2v\_alu : alu

PORT MAP(alu\_A => SYNTHESIZED\_WIRE\_27,

alu\_B => SYNTHESIZED\_WIRE\_1,

SEL => SYNTHESIZED\_WIRE\_29,

C\_hi => SYNTHESIZED\_WIRE\_30,

C\_lo => SYNTHESIZED\_WIRE\_31);

b2v\_z : z

PORT MAP(clk => clk,

clear => clear,

Zin => Zin,

Zhi\_in => SYNTHESIZED\_WIRE\_30,

Zlo\_in => SYNTHESIZED\_WIRE\_31,

BusMuxIn\_Zhi => SYNTHESIZED\_WIRE\_24,

BusMuxIn\_Zlow => SYNTHESIZED\_WIRE\_25);

b2v\_IR : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => IRin,

d => SYNTHESIZED\_WIRE\_1,

q => IRout);

b2v\_LO : lo

PORT MAP(clk => clk,

clear => clear,

LOin => LOin,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

BusMuxIn\_LO => SYNTHESIZED\_WIRE\_5);

b2v\_MAR : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => MARin,

d => SYNTHESIZED\_WIRE\_1,

q => MARout);

b2v\_MDR : mdr

PORT MAP(clk => clk,

clear => clear,

MDRin => MDRin,

Read => Read,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

MemDataIn => MemDataIn,

BusMuxIn\_MDR => SYNTHESIZED\_WIRE\_6,

MemIn\_MDR => MemDataOut);

b2v\_out\_port : reg32

PORT MAP(clk => clk,

reset\_n => clear,

en => OutPortin,

d => SYNTHESIZED\_WIRE\_1,

q => OutPortout);

b2v\_PC : pc

PORT MAP(clk => clk,

clear => clear,

PCin => PCin,

IncPC => IncPC,

BusMuxOut => SYNTHESIZED\_WIRE\_1,

BusMuxIn\_PC => SYNTHESIZED\_WIRE\_7);

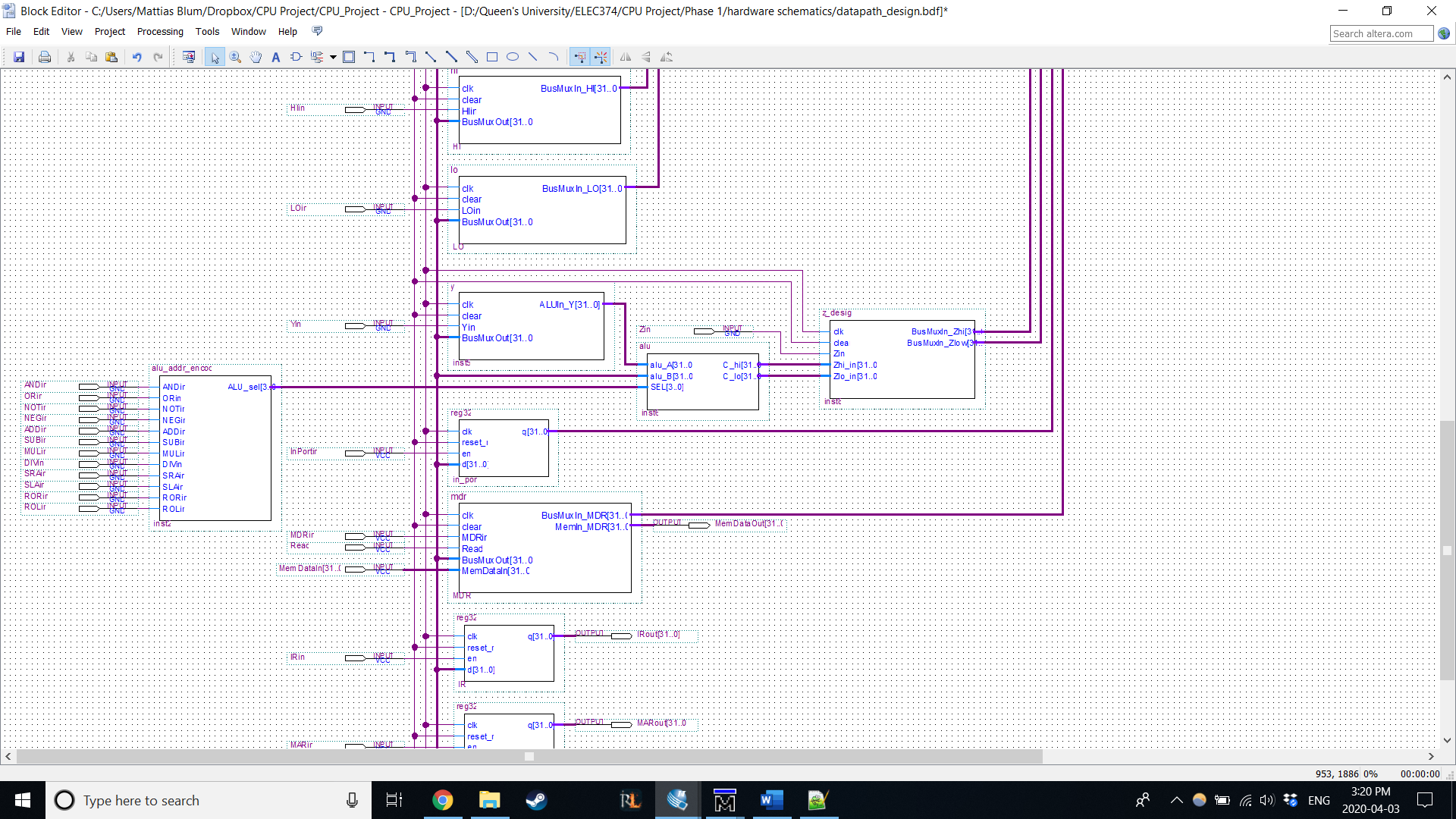
END bdf\_type;

# Block Schematics

## Datapath

**The cut-off part at the bottom is just the Out.Port register**

A screenshot of a computer

Description automatically generated

# Testbench Files

## AND/OR/ADD/SUB

**The only differences between the code for each is the instruction select signal (AND\_tb, OR\_tb, etc) and the numeric value of the operands (visible in the waveforms)**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_and IS

END ENTITY tb\_and;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_and\_arch OF tb\_and IS -- Add any other signals to see in your simulation

SIGNAL PCout\_tb, Zlowout\_tb, MDRout\_tb, R2out\_tb, R4out\_tb: std\_logic;

SIGNAL MARin\_tb, PCin\_tb, MDRin\_tb, IRin\_tb, Yin\_tb, Zin\_tb: std\_logic;

SIGNAL IncPC\_tb, Read\_tb, AND\_tb, R5in\_tb, R2in\_tb, R4in\_tb: std\_logic;

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL Mdatain\_tb,BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, Reg\_load1a, Reg\_load1b, Reg\_load2a, Reg\_load2b, Reg\_load3a, Reg\_load3b, T0, T1,

T2, T3, T4, T5);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : 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;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

R0in : IN STD\_LOGIC;

R1in : IN STD\_LOGIC;

R2in : IN STD\_LOGIC;

R3in : IN STD\_LOGIC;

R4in : IN STD\_LOGIC;

R5in : IN STD\_LOGIC;

R6in : IN STD\_LOGIC;

R7in : IN STD\_LOGIC;

R8in : IN STD\_LOGIC;

R9in : IN STD\_LOGIC;

R10in : IN STD\_LOGIC;

R11in : IN STD\_LOGIC;

R12in : IN STD\_LOGIC;

R13in : IN STD\_LOGIC;

R14in : IN STD\_LOGIC;

R15in : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

MemDataIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

IRout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MARout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemDataOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutPortout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

R0out => '0',

R1out => '0',

R2out => R2out\_tb,

R3out => '0',

R4out => R4out\_tb,

R5out => '0',

R6out => '0',

R7out => '0',

R8out => '0',

R9out => '0',

R10out => '0',

R11out => '0',

R12out => '0',

R13out => '0',

R14out => '0',

R15out => '0',

HIout => '0',

LOout => '0',

Zlowout => Zlowout\_tb,

Zhighout => '0',

MDRout => MDRout\_tb,

PCout => PCout\_tb,

Cout => '0',

InPortout => '0',

R0in => '0',

R1in => '0',

R2in => R2in\_tb,

R3in => '0',

R4in => R4in\_tb,

R5in => R5in\_tb,

R6in => '0',

R7in => '0',

R8in => '0',

R9in => '0',

R10in => '0',

R11in => '0',

R12in => '0',

R13in => '0',

R14in => '0',

R15in => '0',

HIin => '0',

LOin => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

InPortin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

IRin => IRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

ANDin => AND\_tb,

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => '0',

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

MemDataIn => Mdatain\_tb,

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

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

wait for 25 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= Reg\_load1a;

WHEN Reg\_load1a =>

Present\_state <= Reg\_load1b;

WHEN Reg\_load1b =>

Present\_state <= Reg\_load2a;

WHEN Reg\_load2a =>

Present\_state <= Reg\_load2b;

WHEN Reg\_load2b =>

Present\_state <= Reg\_load3a;

WHEN Reg\_load3a =>

Present\_state <= Reg\_load3b;

WHEN Reg\_load3b =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

R2out\_tb <= '0'; R4out\_tb <= '0'; MARin\_tb <= '0';

PCin\_tb <='0'; MDRin\_tb <= '0'; IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; AND\_tb <= '0';

R2in\_tb <= '0'; R4in\_tb <= '0'; R5in\_tb <= '0'; Mdatain\_tb <= x"00000000";

Clear\_tb <= '0';

WHEN Reg\_load1a =>

Mdatain\_tb <= x"00000022";

Read\_tb <= '1'; MDRin\_tb <= '1'; Clear\_tb <= '1';

WHEN Reg\_load1b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R2in\_tb <= '1'; -- initialize R2 with the value $22

WHEN Reg\_load2a =>

MDRout\_tb <= '0'; R2in\_tb <= '0';

Mdatain\_tb <= x"00000024";

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN Reg\_load2b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R4in\_tb <= '1'; -- initialize R4 with the value $24

WHEN Reg\_load3a =>

MDRout\_tb <= '0'; R4in\_tb <= '0';

Mdatain\_tb <= x"00000026";

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN Reg\_load3b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R5in\_tb <= '1'; -- initialize R5 with the value $26

WHEN T0 => -- see if you need to de-assert these signals

MDRout\_tb <= '0'; R5in\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1';

WHEN T1 =>

MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; Read\_tb <= '1'; MDRin\_tb <= '1';

Mdatain\_tb <= x"4A920000"; -- opcode for AND

WHEN T2 =>

PCout\_tb <= '0'; IncPC\_tb <= '0'; PCin\_tb <= '0'; Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

R2out\_tb <= '1'; Yin\_tb <= '1';

WHEN T4 =>

R2out\_tb <= '0'; Yin\_tb <= '0';

R4out\_tb <= '1'; AND\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

R4out\_tb <= '0'; AND\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; R5in\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_and\_arch;

## MUL/DIV

**The only differences are the instruction control signal used (MUL\_tb/DIV\_tb) and the numeric value of the operands (visible in the waveforms)**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_mul IS

END ENTITY tb\_mul;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_mul\_arch OF tb\_mul IS -- Add any other signals to see in your simulation

SIGNAL PCout\_tb, Zlowout\_tb, Zhighout\_tb, MDRout\_tb, HIout\_tb, LOout\_tb, R2out\_tb, R4out\_tb: std\_logic;

SIGNAL MARin\_tb, PCin\_tb, MDRin\_tb, IRin\_tb, Yin\_tb, Zin\_tb: std\_logic;

SIGNAL IncPC\_tb, Read\_tb, MUL\_tb, R2in\_tb, R4in\_tb, HIin\_tb, LOin\_tb: std\_logic;

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL Mdatain\_tb,BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, Reg\_load1a, Reg\_load1b, Reg\_load2a, Reg\_load2b, T0, T1,

T2, T3, T4, T5, T6);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : 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;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

R0in : IN STD\_LOGIC;

R1in : IN STD\_LOGIC;

R2in : IN STD\_LOGIC;

R3in : IN STD\_LOGIC;

R4in : IN STD\_LOGIC;

R5in : IN STD\_LOGIC;

R6in : IN STD\_LOGIC;

R7in : IN STD\_LOGIC;

R8in : IN STD\_LOGIC;

R9in : IN STD\_LOGIC;

R10in : IN STD\_LOGIC;

R11in : IN STD\_LOGIC;

R12in : IN STD\_LOGIC;

R13in : IN STD\_LOGIC;

R14in : IN STD\_LOGIC;

R15in : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

MemDataIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

IRout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MARout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemDataOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutPortout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

R0out => '0',

R1out => '0',

R2out => R2out\_tb,

R3out => '0',

R4out => R4out\_tb,

R5out => '0',

R6out => '0',

R7out => '0',

R8out => '0',

R9out => '0',

R10out => '0',

R11out => '0',

R12out => '0',

R13out => '0',

R14out => '0',

R15out => '0',

HIout => HIout\_tb,

LOout => LOout\_tb,

Zlowout => Zlowout\_tb,

Zhighout => Zhighout\_tb,

MDRout => MDRout\_tb,

PCout => PCout\_tb,

Cout => '0',

InPortout => '0',

R0in => '0',

R1in => '0',

R2in => R2in\_tb,

R3in => '0',

R4in => R4in\_tb,

R5in => '0',

R6in => '0',

R7in => '0',

R8in => '0',

R9in => '0',

R10in => '0',

R11in => '0',

R12in => '0',

R13in => '0',

R14in => '0',

R15in => '0',

HIin => HIin\_tb,

LOin => LOin\_tb,

Read => Read\_tb,

MDRin => MDRin\_tb,

InPortin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

IRin => IRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => '0',

SUBin => '0',

MULin => MUL\_tb,

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

MemDataIn => Mdatain\_tb,

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

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

wait for 25 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= Reg\_load1a;

WHEN Reg\_load1a =>

Present\_state <= Reg\_load1b;

WHEN Reg\_load1b =>

Present\_state <= Reg\_load2a;

WHEN Reg\_load2a =>

Present\_state <= Reg\_load2b;

WHEN Reg\_load2b =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN T5 =>

Present\_state <= T6;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; Zhighout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

R2out\_tb <= '0'; R4out\_tb <= '0'; HIout\_tb <= '0'; LOout\_tb <= '0'; MARin\_tb <= '0';

PCin\_tb <='0'; MDRin\_tb <= '0'; IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; MUL\_tb <= '0';

R2in\_tb <= '0'; R4in\_tb <= '0'; HIin\_tb <= '0'; LOin\_tb <= '0'; Mdatain\_tb <= x"00000000";

Clear\_tb <= '0';

WHEN Reg\_load1a =>

Mdatain\_tb <= x"00000022";

Read\_tb <= '1'; MDRin\_tb <= '1'; Clear\_tb <= '1';

WHEN Reg\_load1b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R2in\_tb <= '1'; -- initialize R2 with the value $22

WHEN Reg\_load2a =>

MDRout\_tb <= '0'; R2in\_tb <= '0';

Mdatain\_tb <= x"00000024";

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN Reg\_load2b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R4in\_tb <= '1'; -- initialize R4 with the value $24

WHEN T0 => -- see if you need to de-assert these signals

MDRout\_tb <= '0'; R4in\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1';

WHEN T1 =>

MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; Read\_tb <= '1'; MDRin\_tb <= '1';

Mdatain\_tb <= x"4A920000"; -- opcode for AND

WHEN T2 =>

PCout\_tb <= '0'; IncPC\_tb <= '0'; PCin\_tb <= '0'; Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

R2out\_tb <= '1'; Yin\_tb <= '1';

WHEN T4 =>

R2out\_tb <= '0'; Yin\_tb <= '0';

R4out\_tb <= '1'; MUL\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

R4out\_tb <= '0'; MUL\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; LOin\_tb <= '1';

WHEN T6 =>

Zlowout\_tb <= '0'; LOin\_tb <= '0';

Zhighout\_tb <= '1'; HIin\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_mul\_arch;

## SHR/SHL/ROR/ROL

**The only differences are the instruction control signal used (SHR\_tb, SHL\_tb) and the numeric value of the operands (visible in the waveforms)**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_shr IS

END ENTITY tb\_shr;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_shr\_arch OF tb\_shr IS -- Add any other signals to see in your simulation

SIGNAL PCout\_tb, Zlowout\_tb, MDRout\_tb, R2out\_tb, R4out\_tb: std\_logic;

SIGNAL MARin\_tb, PCin\_tb, MDRin\_tb, IRin\_tb, Yin\_tb, Zin\_tb: std\_logic;

SIGNAL IncPC\_tb, Read\_tb, SHR\_tb, R5in\_tb, R2in\_tb, R4in\_tb: std\_logic;

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL Mdatain\_tb,BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, Reg\_load1a, Reg\_load1b, Reg\_load2a, Reg\_load2b, Reg\_load3a, Reg\_load3b, T0, T1,

T2, T3, T4, T5);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : 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;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

R0in : IN STD\_LOGIC;

R1in : IN STD\_LOGIC;

R2in : IN STD\_LOGIC;

R3in : IN STD\_LOGIC;

R4in : IN STD\_LOGIC;

R5in : IN STD\_LOGIC;

R6in : IN STD\_LOGIC;

R7in : IN STD\_LOGIC;

R8in : IN STD\_LOGIC;

R9in : IN STD\_LOGIC;

R10in : IN STD\_LOGIC;

R11in : IN STD\_LOGIC;

R12in : IN STD\_LOGIC;

R13in : IN STD\_LOGIC;

R14in : IN STD\_LOGIC;

R15in : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

MemDataIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

IRout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MARout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemDataOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutPortout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

R0out => '0',

R1out => '0',

R2out => R2out\_tb,

R3out => '0',

R4out => R4out\_tb,

R5out => '0',

R6out => '0',

R7out => '0',

R8out => '0',

R9out => '0',

R10out => '0',

R11out => '0',

R12out => '0',

R13out => '0',

R14out => '0',

R15out => '0',

HIout => '0',

LOout => '0',

Zlowout => Zlowout\_tb,

Zhighout => '0',

MDRout => MDRout\_tb,

PCout => PCout\_tb,

Cout => '0',

InPortout => '0',

R0in => '0',

R1in => '0',

R2in => R2in\_tb,

R3in => '0',

R4in => R4in\_tb,

R5in => R5in\_tb,

R6in => '0',

R7in => '0',

R8in => '0',

R9in => '0',

R10in => '0',

R11in => '0',

R12in => '0',

R13in => '0',

R14in => '0',

R15in => '0',

HIin => '0',

LOin => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

InPortin => '0',

Yin => Yin\_tb,

PCin => PCin\_tb,

IncPC => IncPC\_tb,

IRin => IRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

ANDin => '0',

ORin => '0',

NOTin => '0',

NEGin => '0',

ADDin => '0',

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => SHR\_tb,

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

MemDataIn => Mdatain\_tb,

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

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

wait for 25 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= Reg\_load1a;

WHEN Reg\_load1a =>

Present\_state <= Reg\_load1b;

WHEN Reg\_load1b =>

Present\_state <= Reg\_load2a;

WHEN Reg\_load2a =>

Present\_state <= Reg\_load2b;

WHEN Reg\_load2b =>

Present\_state <= Reg\_load3a;

WHEN Reg\_load3a =>

Present\_state <= Reg\_load3b;

WHEN Reg\_load3b =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN T4 =>

Present\_state <= T5;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

R2out\_tb <= '0'; R4out\_tb <= '0'; MARin\_tb <= '0';

PCin\_tb <='0'; MDRin\_tb <= '0'; IRin\_tb <= '0'; Yin\_tb <= '0'; Zin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; SHR\_tb <= '0';

R2in\_tb <= '0'; R4in\_tb <= '0'; R5in\_tb <= '0'; Mdatain\_tb <= x"00000000";

Clear\_tb <= '0';

WHEN Reg\_load1a =>

Mdatain\_tb <= x"7000000F";

Read\_tb <= '1'; MDRin\_tb <= '1'; Clear\_tb <= '1';

WHEN Reg\_load1b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R2in\_tb <= '1'; -- initialize R2 with the value $7000000F

WHEN Reg\_load2a =>

MDRout\_tb <= '0'; R2in\_tb <= '0';

Mdatain\_tb <= x"00000004";

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN Reg\_load2b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R4in\_tb <= '1'; -- initialize R4 with the value $4

WHEN Reg\_load3a =>

MDRout\_tb <= '0'; R4in\_tb <= '0';

Mdatain\_tb <= x"00000026";

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN Reg\_load3b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R5in\_tb <= '1'; -- initialize R5 with the value $26

WHEN T0 => -- see if you need to de-assert these signals

MDRout\_tb <= '0'; R5in\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1';

WHEN T1 =>

MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; Read\_tb <= '1'; MDRin\_tb <= '1';

Mdatain\_tb <= x"4A920000"; -- opcode for AND

WHEN T2 =>

PCout\_tb <= '0'; IncPC\_tb <= '0'; PCin\_tb <= '0'; Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

R2out\_tb <= '1'; Yin\_tb <= '1';

WHEN T4 =>

R2out\_tb <= '0'; Yin\_tb <= '0';

R4out\_tb <= '1'; SHR\_tb <= '1'; Zin\_tb <= '1';

WHEN T5 =>

R4out\_tb <= '0'; SHR\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; R5in\_tb <= '1';

WHEN OTHERS =>

END CASE;

END PROCESS;

END ARCHITECTURE tb\_shr\_arch;

## NOT/NEG

**The only differences are the instruction control signal used (NOT\_tb/NEG\_tb) and the numeric value of the operands (visible in the waveforms)**

-- and datapath\_tb.vhd file: <This is the filename>

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

-- entity declaration only; no definition here

ENTITY tb\_not IS

END ENTITY tb\_not;

-- Architecture of the testbench with the signal names

ARCHITECTURE tb\_not\_arch OF tb\_not IS -- Add any other signals to see in your simulation

SIGNAL PCout\_tb, Zlowout\_tb, MDRout\_tb, R2out\_tb: std\_logic;

SIGNAL MARin\_tb, PCin\_tb, MDRin\_tb, IRin\_tb, Zin\_tb: std\_logic;

SIGNAL IncPC\_tb, Read\_tb, NOT\_tb, R2in\_tb, R5in\_tb: std\_logic;

SIGNAL Clock\_tb, Clear\_tb: std\_logic;

SIGNAL Mdatain\_tb,BusData\_tb : std\_logic\_vector (31 downto 0);

TYPE State IS (default, Reg\_load1a, Reg\_load1b, Reg\_load2a, Reg\_load2b, T0, T1,

T2, T3, T4);

SIGNAL Present\_state: State := default;

-- component instantiation of the datapath

COMPONENT datapath

PORT (

clk : IN STD\_LOGIC;

clear : 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;

HIout : IN STD\_LOGIC;

LOout : IN STD\_LOGIC;

Zlowout : IN STD\_LOGIC;

Zhighout : IN STD\_LOGIC;

MDRout : IN STD\_LOGIC;

PCout : IN STD\_LOGIC;

Cout : IN STD\_LOGIC;

InPortout : IN STD\_LOGIC;

R0in : IN STD\_LOGIC;

R1in : IN STD\_LOGIC;

R2in : IN STD\_LOGIC;

R3in : IN STD\_LOGIC;

R4in : IN STD\_LOGIC;

R5in : IN STD\_LOGIC;

R6in : IN STD\_LOGIC;

R7in : IN STD\_LOGIC;

R8in : IN STD\_LOGIC;

R9in : IN STD\_LOGIC;

R10in : IN STD\_LOGIC;

R11in : IN STD\_LOGIC;

R12in : IN STD\_LOGIC;

R13in : IN STD\_LOGIC;

R14in : IN STD\_LOGIC;

R15in : IN STD\_LOGIC;

HIin : IN STD\_LOGIC;

LOin : IN STD\_LOGIC;

Read : IN STD\_LOGIC;

MDRin : IN STD\_LOGIC;

InPortin : IN STD\_LOGIC;

Yin : IN STD\_LOGIC;

PCin : IN STD\_LOGIC;

IncPC : IN STD\_LOGIC;

IRin : IN STD\_LOGIC;

MARin : IN STD\_LOGIC;

OutPortin : IN STD\_LOGIC;

ANDin : IN STD\_LOGIC;

ORin : IN STD\_LOGIC;

NOTin : IN STD\_LOGIC;

NEGin : IN STD\_LOGIC;

ADDin : IN STD\_LOGIC;

SUBin : IN STD\_LOGIC;

MULin : IN STD\_LOGIC;

DIVin : IN STD\_LOGIC;

SRAin : IN STD\_LOGIC;

SLAin : IN STD\_LOGIC;

RORin : IN STD\_LOGIC;

ROLin : IN STD\_LOGIC;

Zin : IN STD\_LOGIC;

MemDataIn : IN STD\_LOGIC\_VECTOR(31 DOWNTO 0);

BusData : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

IRout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MARout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

MemDataOut : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0);

OutPortout : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0)

);

END COMPONENT datapath;

BEGIN

DUT : datapath

--port mapping: between the dut and the testbench signals

PORT MAP (

clk => Clock\_tb,

clear => Clear\_tb,

R0out => '0',

R1out => '0',

R2out => R2out\_tb,

R3out => '0',

R4out => '0',

R5out => '0',

R6out => '0',

R7out => '0',

R8out => '0',

R9out => '0',

R10out => '0',

R11out => '0',

R12out => '0',

R13out => '0',

R14out => '0',

R15out => '0',

HIout => '0',

LOout => '0',

Zlowout => Zlowout\_tb,

Zhighout => '0',

MDRout => MDRout\_tb,

PCout => PCout\_tb,

Cout => '0',

InPortout => '0',

R0in => '0',

R1in => '0',

R2in => R2in\_tb,

R3in => '0',

R4in => '0',

R5in => R5in\_tb,

R6in => '0',

R7in => '0',

R8in => '0',

R9in => '0',

R10in => '0',

R11in => '0',

R12in => '0',

R13in => '0',

R14in => '0',

R15in => '0',

HIin => '0',

LOin => '0',

Read => Read\_tb,

MDRin => MDRin\_tb,

InPortin => '0',

Yin => '0',

PCin => PCin\_tb,

IncPC => IncPC\_tb,

IRin => IRin\_tb,

MARin => MARin\_tb,

OutPortin => '0',

ANDin => '0',

ORin => '0',

NOTin => NOT\_tb,

NEGin => '0',

ADDin => '0',

SUBin => '0',

MULin => '0',

DIVin => '0',

SRAin => '0',

SLAin => '0',

RORin => '0',

ROLin => '0',

Zin => Zin\_tb,

MemDataIn => Mdatain\_tb,

BusData => BusData\_tb

);

--add test logic here

Clock\_process: PROCESS IS

BEGIN

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

wait for 25 ns;

END PROCESS Clock\_process;

PROCESS (Clock\_tb) IS -- finite state machine

BEGIN

IF (rising\_edge (Clock\_tb)) THEN -- if clock rising-edge

CASE Present\_state IS

WHEN Default =>

Present\_state <= Reg\_load1a;

WHEN Reg\_load1a =>

Present\_state <= Reg\_load1b;

WHEN Reg\_load1b =>

Present\_state <= Reg\_load2a;

WHEN Reg\_load2a =>

Present\_state <= Reg\_load2b;

WHEN Reg\_load2b =>

Present\_state <= T0;

WHEN T0 =>

Present\_state <= T1;

WHEN T1 =>

Present\_state <= T2;

WHEN T2 =>

Present\_state <= T3;

WHEN T3 =>

Present\_state <= T4;

WHEN OTHERS =>

END CASE;

END IF;

END PROCESS;

PROCESS (Present\_state) IS -- do the required job in each state

BEGIN

CASE Present\_state IS -- assert the required signals in each clock cycle

WHEN Default =>

PCout\_tb <= '0'; Zlowout\_tb <= '0'; MDRout\_tb <= '0'; -- initialize the signals

R2out\_tb <= '0'; MARin\_tb <= '0'; MDRin\_tb <= '0';

PCin\_tb <='0'; IRin\_tb <= '0'; Zin\_tb <= '0';

IncPC\_tb <= '0'; Read\_tb <= '0'; NOT\_tb <= '0';

R2in\_tb <= '0'; R5in\_tb <= '0'; Mdatain\_tb <= x"00000000";

Clear\_tb <= '0';

WHEN Reg\_load1a =>

Mdatain\_tb <= x"0000000F";

Read\_tb <= '1'; MDRin\_tb <= '1'; Clear\_tb <= '1';

WHEN Reg\_load1b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R2in\_tb <= '1'; -- initialize R2 with the value $F

WHEN Reg\_load2a =>

MDRout\_tb <= '0'; R2in\_tb <= '0';

Mdatain\_tb <= x"00000026";

Read\_tb <= '1'; MDRin\_tb <= '1';

WHEN Reg\_load2b =>

Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; R5in\_tb <= '1'; -- initialize R5 with the value $26

WHEN T0 => -- see if you need to de-assert these signals

MDRout\_tb <= '0'; R5in\_tb <= '0';

PCout\_tb <= '1'; MARin\_tb <= '1';

WHEN T1 =>

MARin\_tb <= '0';

IncPC\_tb <= '1'; PCin\_tb <= '1'; Read\_tb <= '1'; MDRin\_tb <= '1';

Mdatain\_tb <= x"4A920000"; -- opcode for AND

WHEN T2 =>

PCout\_tb <= '0'; IncPC\_tb <= '0'; PCin\_tb <= '0'; Read\_tb <= '0'; MDRin\_tb <= '0';

MDRout\_tb <= '1'; IRin\_tb <= '1';

WHEN T3 =>

MDRout\_tb <= '0'; IRin\_tb <= '0';

R2out\_tb <= '1'; NOT\_tb <= '1'; Zin\_tb <= '1';

WHEN T4 =>

R2out\_tb <= '0'; NOT\_tb <= '0'; Zin\_tb <= '0';

Zlowout\_tb <= '1'; R5in\_tb <= '1';

WHEN OTHERS =>

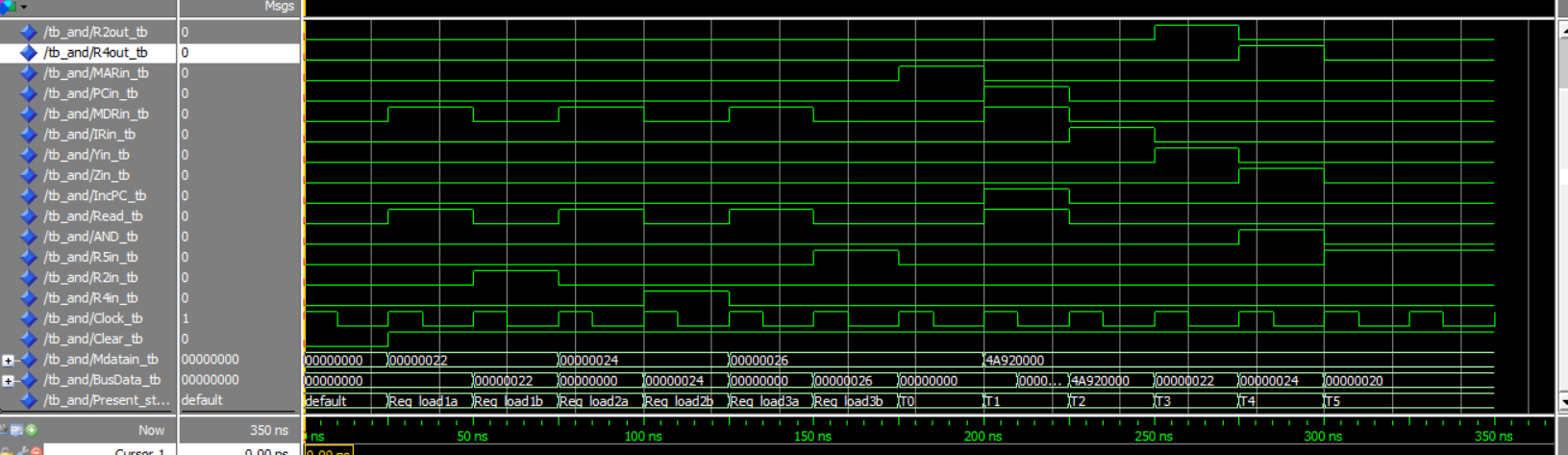
END CASE;

END PROCESS;

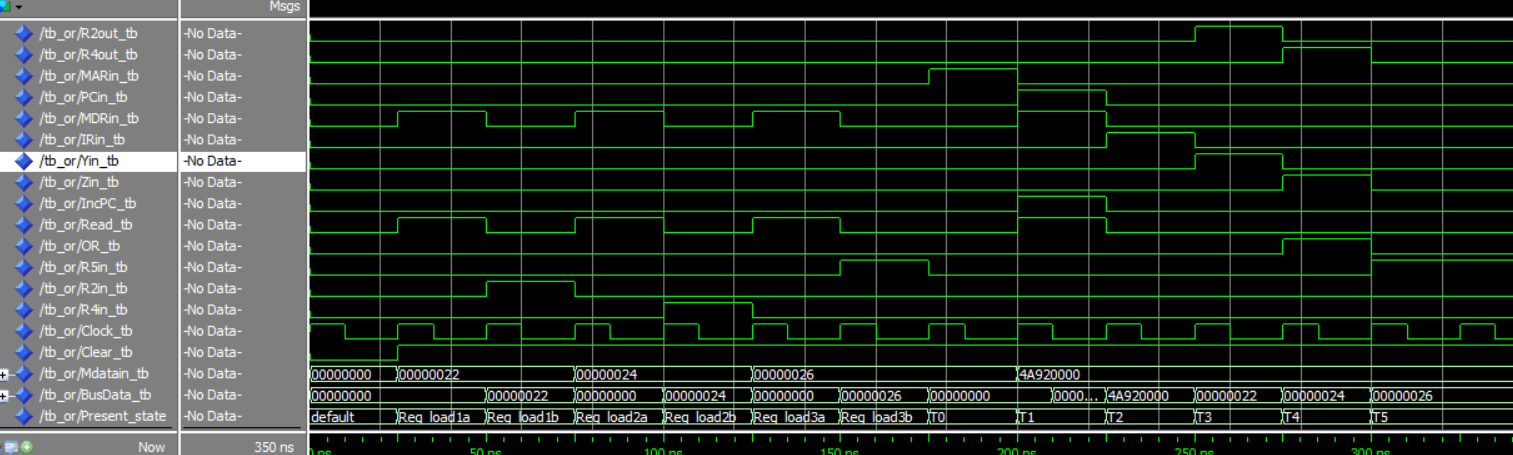
END ARCHITECTURE tb\_not\_arch;

# Output Waveforms

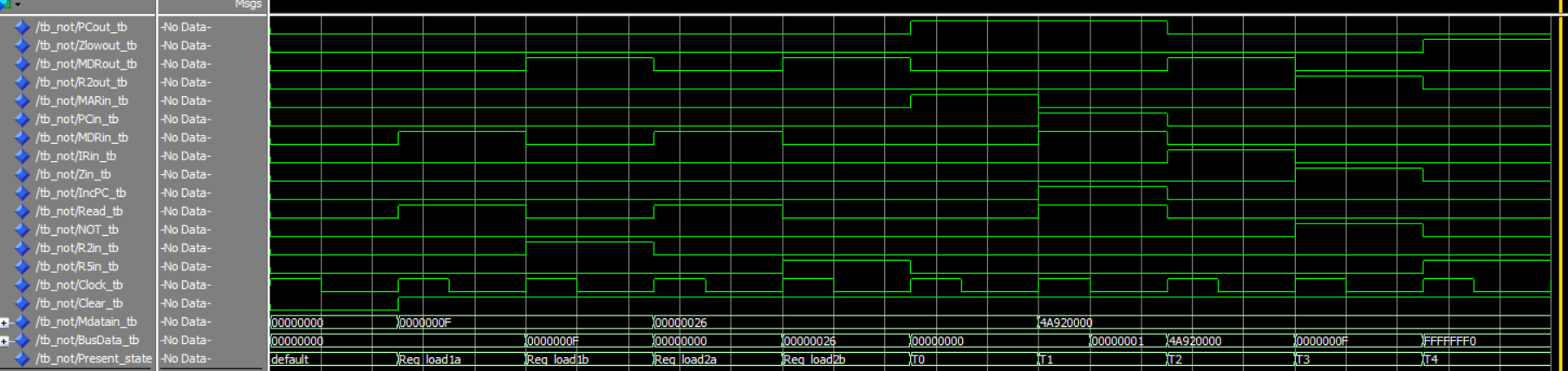
## AND



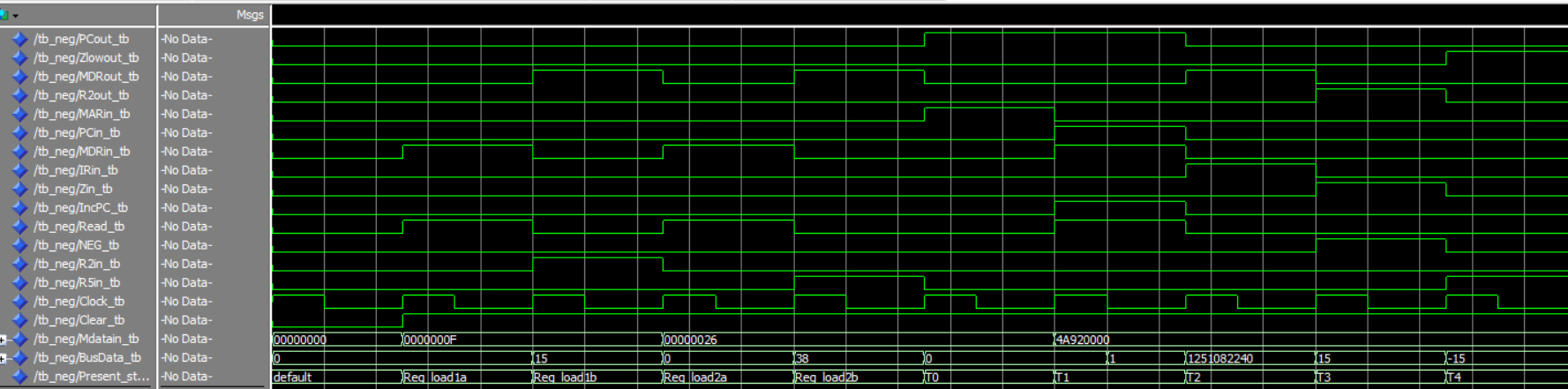
## OR



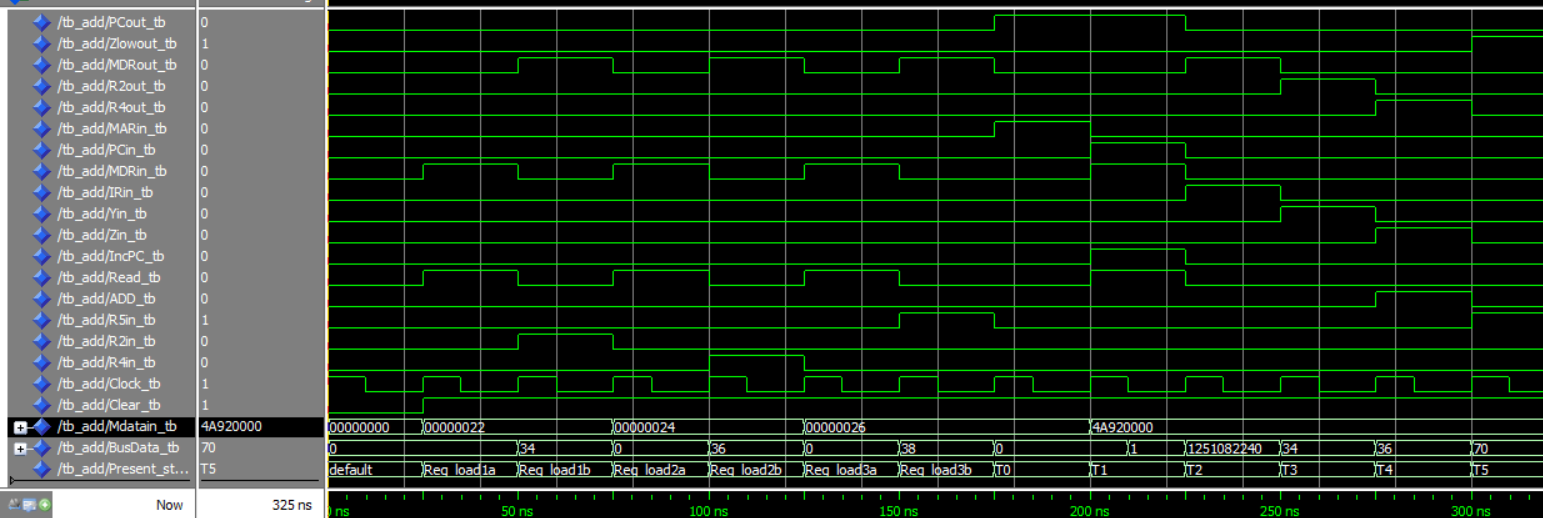
## NOT



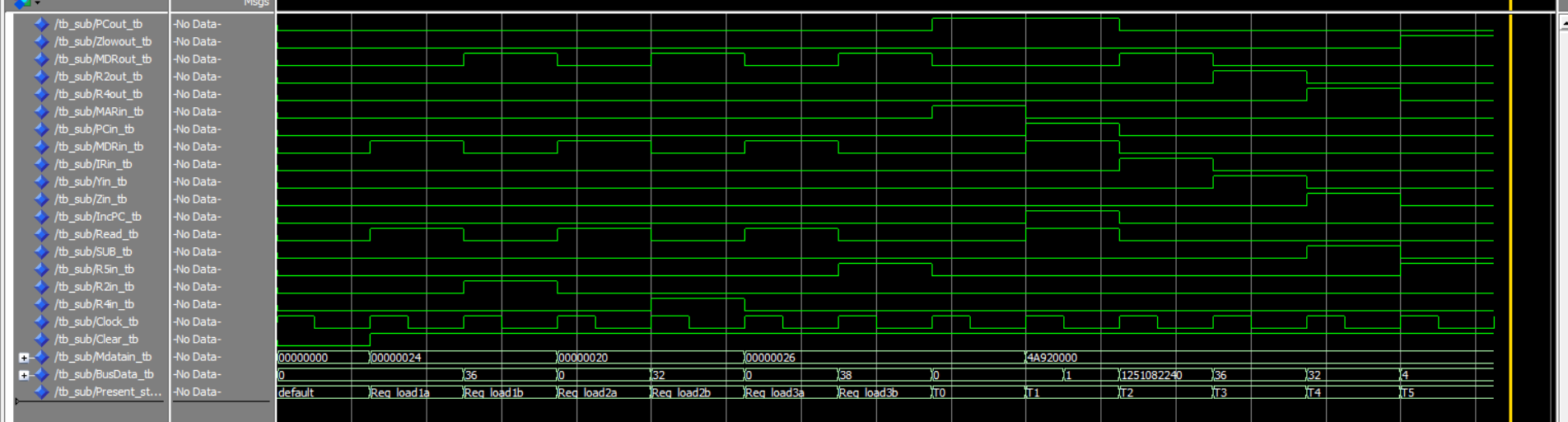
## NEG



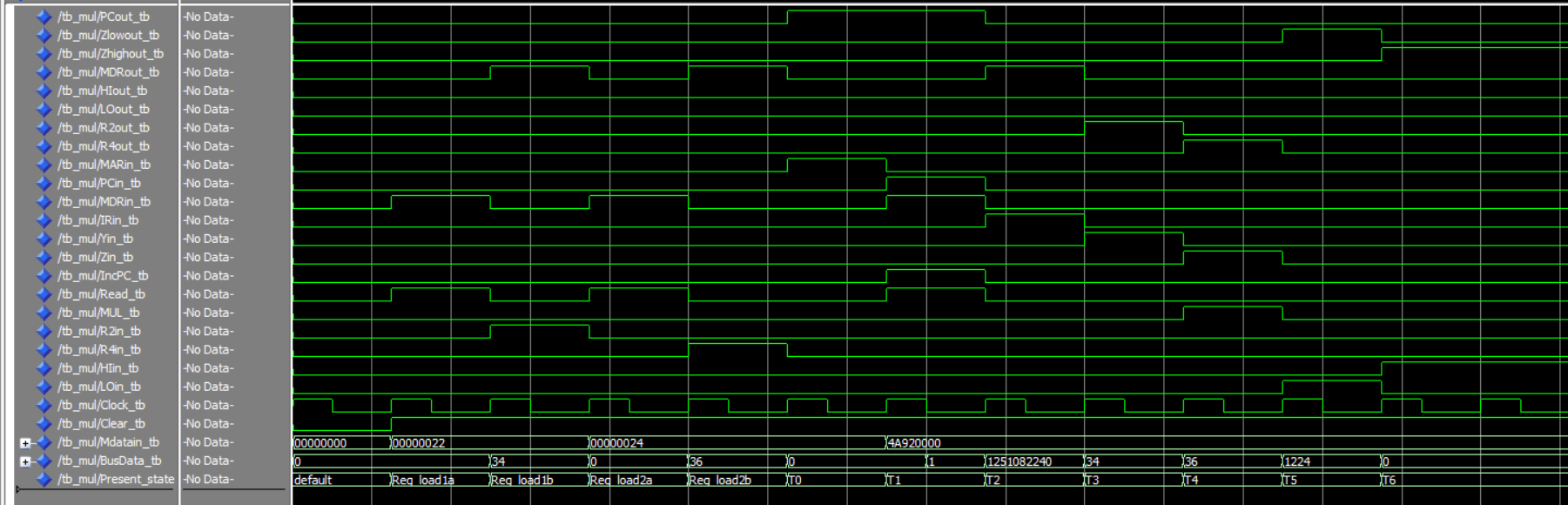
## ADD



## SUB

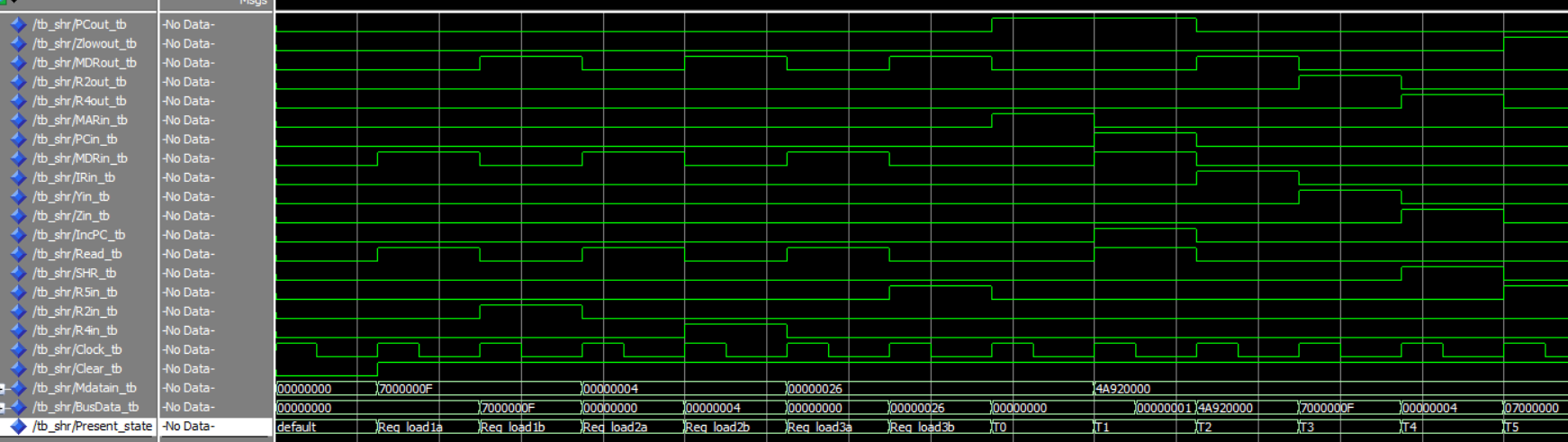


## MUL

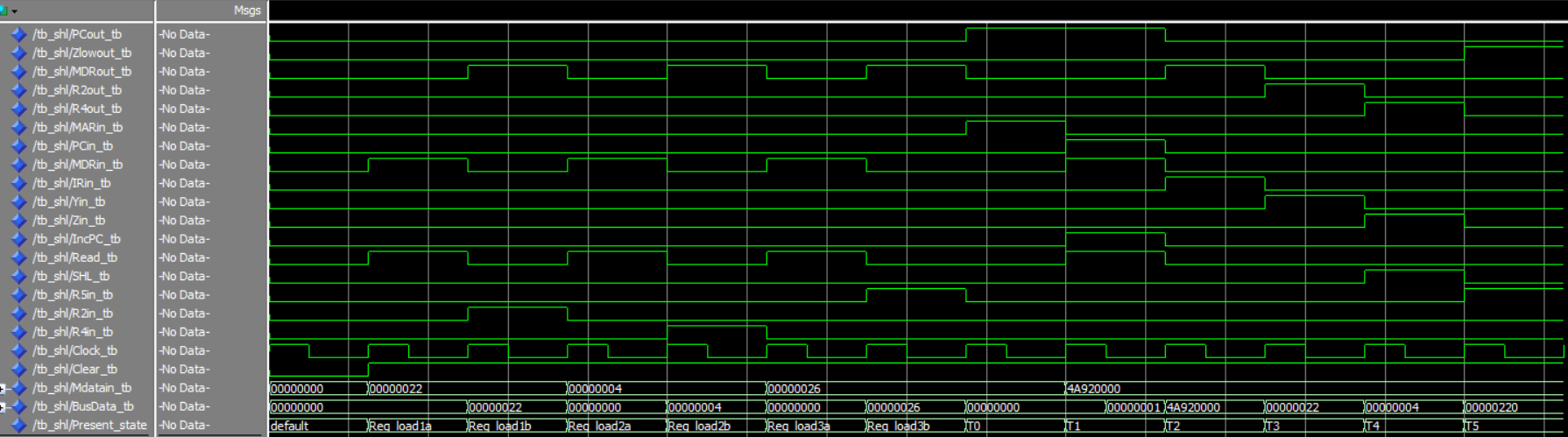


## DIV

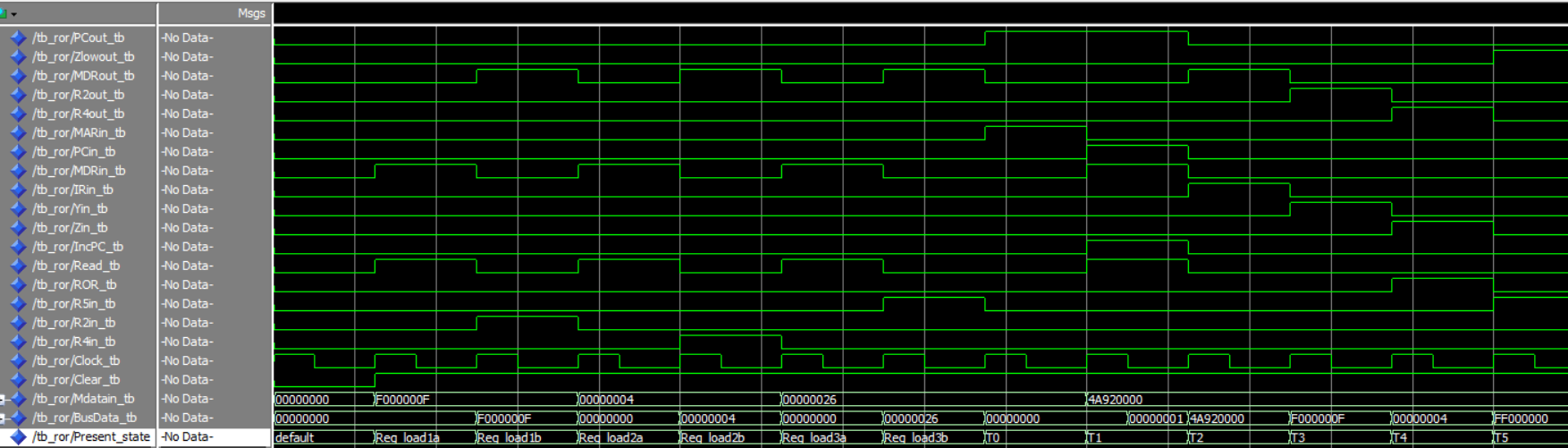
## SHR



## SHL



## ROR



## ROL

