LIBRARY ieee ;

USE ieee.std\_logic\_1164.all ;

USE ieee.std\_logic\_signed.all ;

--USE work.subccts.all ;

ENTITY proc IS

PORT ( Data : IN STD\_LOGIC\_VECTOR(7 DOWNTO 0) ;

Reset, w : IN STD\_LOGIC ;

Clock : IN STD\_LOGIC ;

Rs, Rt : IN STD\_LOGIC\_VECTOR(1 DOWNTO 0) ;--F用來控制opcode

OP :in STD\_LOGIC\_VECTOR(3 DOWNTO 0) ;

Done : BUFFER STD\_LOGIC ;

led1,led2,led3,led4,led5,led6:out std\_Logic\_vector(6 downto 0);

BusW : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0) ) ;

END proc ;

ARCHITECTURE Behavior OF proc IS

COMPONENT seven\_segment\_display

PORT ( W,X,Y,Z : IN STD\_LOGIC;

A,B,C,D,E,F,G : OUT STD\_LOGIC);

END COMPONENT ;

-- SIGNAL X, Y, Rin, Rout : STD\_LOGIC\_VECTOR(0 TO 3) ;

-- SIGNAL Clear, High, AddSub : STD\_LOGIC ;

-- SIGNAL Extern, Ain, Gin, Gout, FRin : STD\_LOGIC ;

-- SIGNAL Count, Zero, T, I : STD\_LOGIC\_VECTOR(1 DOWNTO 0) ;

SIGNAL R0, R1, R2, R3 : STD\_LOGIC\_VECTOR(7 DOWNTO 0) ;

-- SIGNAL A, Sum, G : STD\_LOGIC\_VECTOR(7 DOWNTO 0) ;

-- SIGNAL Func, FuncReg, Sel : STD\_LOGIC\_VECTOR(1 TO 6) ;

signal Func : std\_logic\_vector(3 downto 0) ;

-- SIGNAL OP : STD\_LOGIC\_VECTOR(3 DOWNTO 0) ;

SIGNAL bcd: STD\_LOGIC\_VECTOR(3 DOWNTO 0) ;

SIGNAL s,t : std\_Logic\_vector(7 downto 0);

-- signal led1,led2,led3,led4,led5,led6: std\_Logic\_vector(6 downto 0);

BEGIN

-- Zero <= "00" ; High <= '1' ;

-- Clear <= Reset OR Done OR (NOT w AND NOT T(1) AND NOT T(0)) ;

-- counter: upcount PORT MAP ( Clear, Clock, Count ) ;

-- T <= Count ;

process(clock,OP,Func)

begin

if clock'event and clock = '1' then

Func <= Rs & Rt ;

CASE OP IS

WHEN "0000" =>--LOAD

CASE Rs IS

WHEN "00" =>

R0 <= Data ;

s <= R0 ;

t <= (others=>'0');

BusW <= s ;

WHEN "01" =>

R1 <= Data ;

s <= R1 ;

t <= (others=>'0');

BusW <= s ;

WHEN "10" =>

R2 <= Data ;

s <= R2 ;

t <= (others=>'0');

BusW <= s ;

WHEN "11" =>

R3 <= Data ;

s <= R3 ;

t <= (others=>'0');

BusW <= s ;

when others =>

BusW <= data ;

END CASE ;

WHEN "0001" =>--MOVE

CASE Rs&Rt IS

WHEN "0001" =>

R0 <= R1 ;

s <= R0 ;t<= R1 ;

BusW <= s ;

WHEN "0010" =>

R0 <= R2 ;

s <= R0 ;t<= R2 ;

BusW <= s ;

WHEN "0011" =>

R0 <= R3 ;

s <= R0 ;t<= R3 ;

BusW <= s ;

WHEN "0100" =>

R1 <= R0 ;

s <= R1 ;t<= R0 ;

BusW <= s ;

WHEN "0110" =>

R1 <= R2 ;

s <= R1 ;t<= R2 ;

BusW <= s ;

WHEN "0111" =>

R1 <= R3 ;

s <= R1 ;t<= R3 ;

BusW <= s ;

WHEN "1000" =>

R2 <= R0 ;

s <= R2 ;t<= R0 ;

BusW <= s ;

WHEN "1001" =>

R2 <= R1 ;

s <= R2 ;t<= R1 ;

BusW <= s ;

WHEN "1011" =>

R2 <= R3 ;

s <= R2 ;t<= R3 ;

BusW <= s ;

WHEN "1100" =>

R3 <= R0 ;

s <= R3 ;t<= R0 ;

Busw <= s ;

WHEN "1101" =>

R3 <= R1 ;

s <= R3 ;t<= R1 ;

BusW <= s ;

WHEN "1110" =>

R3 <= R2 ;

s <= R3 ;t<= R2 ;

BusW <= s ;

when others =>

BusW <= Data ;

END CASE ;

WHEN "0010"=>--ADD

CASE Func IS

WHEN "0000" =>

R0 <= R0 + R0 ;

s <= R0 ;t<= R0 ;

BusW <= s ;

WHEN "0001" =>

R0 <= R0 + R1 ;

s <= R0 ;t<= R1 ;

BusW <= s ;

WHEN "0010" =>

R0 <= R0 + R2 ;

s <= R0 ;t<= R2 ;

BusW <= s ;

WHEN "0011" =>

R0 <= R0 + R3 ;

s <= R0 ;t<= R3 ;

BusW <= s ;

WHEN "0100" =>

R1 <= R1 + R0 ;

s <= R1 ;t<= R0 ;

BusW <= s ;

WHEN "0101" =>

R1 <= R1 + R1 ;

s <= R1 ;t<= R1 ;

BusW <= s ;

WHEN "0110" =>

R1 <= R1 + R2 ;

s <= R1 ;t<= R2 ;

BusW <= s ;

WHEN "0111" =>

R1 <= R1 + R3 ;

s <= R1 ;t<= R3;

BusW <= s ;

WHEN "1000" =>

R2 <= R2 + R0 ;

s <= R2 ;t<= R0 ;

BusW <= s ;

WHEN "1001" =>

R2 <= R2 + R1 ;

s <= R2 ;t<= R1 ;

BusW <= s ;

WHEN "1010" =>

R2 <= R2 + R2 ;

s <= R2 ;t<= R2 ;

BusW <= s ;

WHEN "1011" =>

R2 <= R2 + R3 ;

s <= R2 ;t<= R3 ;

BusW <= s ;

WHEN "1100" =>

R3 <= R3 + R0 ;

s <= R3 ;t<= R0 ;

BusW <= s ;

WHEN "1101" =>

R3 <= R3 + R1 ;

s <= R3 ;t<= R1 ;

BusW <= s ;

WHEN "1110" =>

R3 <= R3 + R2 ;

s <= R3 ;t<= R2 ;

BusW <= s ;

WHEN "1111" =>

R3 <= R3 + R3 ;

s <= R3 ;t<= R3 ;

BusW <= s ;

when others =>

BusW <= Data ;

END CASE ;

when others =>

BusW <= Data ;

END CASE ;

end if ;

end process ;

output1:seven\_segment\_display port map(s(7),s(6),s(5),s(4),led1(6),led1(5),led1(4),led1(3),led1(2),led1(1),led1(0));

output2:seven\_segment\_display port map(s(3),s(2),s(1),s(0),led2(6),led2(5),led2(4),led2(3),led2(2),led2(1),led2(0));

output3:seven\_segment\_display port map(t(7),t(6),t(5),t(4),led3(6),led3(5),led3(4),led3(3),led3(2),led3(1),led3(0));

output4:seven\_segment\_display port map(t(3),t(2),t(1),t(0),led4(6),led4(5),led4(4),led4(3),led4(2),led4(1),led4(0));

output5:seven\_segment\_display port map(BusW(7),BusW(6),BusW(5),BusW(4),led5(6),led5(5),led5(4),led5(3),led5(2),led5(1),led5(0));

output6:seven\_segment\_display port map(BusW(3),BusW(2),BusW(1),BusW(0),led6(6),led6(5),led6(4),led6(3),led6(2),led6(1),led6(0));

-- Process(R0,R1,R2,R3)

-- begin

-- case S(7 downto 0) is

-- WHEN "0000" =>

-- LED <= "0000001";

-- WHEN "0001" =>

-- LED <= "1001111";

-- WHEN "0010" =>

-- LED <= "0010010";

-- WHEN "0011" =>

-- LED <= "0000110";

-- WHEN "0100" =>

-- LED <= "1001100";

-- WHEN "0101" =>

-- LED <= "0100100";

-- WHEN "0110" =>

-- LED <= "0100000";

-- WHEN "0111" =>

-- LED <= "0001111";

-- WHEN "1000" =>

-- LED <= "0000000";

-- WHEN "1001" =>

-- LED <= "0001100";

-- WHEN "1010" =>

-- LED <= "0001000";

-- WHEN "1011" =>

-- LED <= "1100000";

-- WHEN "1100" =>

-- LED <= "1110010";

-- WHEN "1101" =>

-- LED <= "1000010";

-- WHEN "1110" =>

-- LED <= "0110000";

-- WHEN "1111" =>

-- LED <= "0111000";

-- end case;

-- end process;

END Behavior ;

-- FRin <= w AND NOT T(1) AND NOT T(0) ;

-- functionreg: regn GENERIC MAP ( N => 6 )

-- PORT MAP ( Func, FRin, Clock, FuncReg ) ;

-- I <= FuncReg(1 TO 2) ;

-- decX: dec2to4 PORT MAP ( FuncReg(3 TO 4), High, X ) ;

-- decY: dec2to4 PORT MAP ( FuncReg(5 TO 6), High, Y ) ;

-- controlsignals: PROCESS ( T, I, X, Y )

-- BEGIN

-- Extern <= '0' ; Done <= '0' ; Ain <= '0' ; Gin <= '0' ;

-- Gout <= '0' ; AddSub <= '0' ; Rin <= "0000" ; Rout <= "0000" ;

-- CASE T IS

-- WHEN "00" => -- no signals asserted in time step T0

-- WHEN "01" => -- define signals asserted in time step T1

-- CASE I IS

-- WHEN "00" => -- Load

-- Extern <= '1' ; Rin <= X ; Done <= '1' ;

-- WHEN "01" => -- Move

-- Rout <= Y ; Rin <= X ; Done <= '1' ;

-- WHEN OTHERS => -- Add, Sub

-- Rout <= X ; Ain <= '1' ;

-- END CASE ;

-- WHEN "10" => -- define signals asserted in time step T2

-- CASE I IS

-- WHEN "10" => -- Add

-- Rout <= Y ; Gin <= '1' ;

-- WHEN "11" => -- Sub

-- Rout <= Y ; AddSub <= '1' ; Gin <= '1' ;

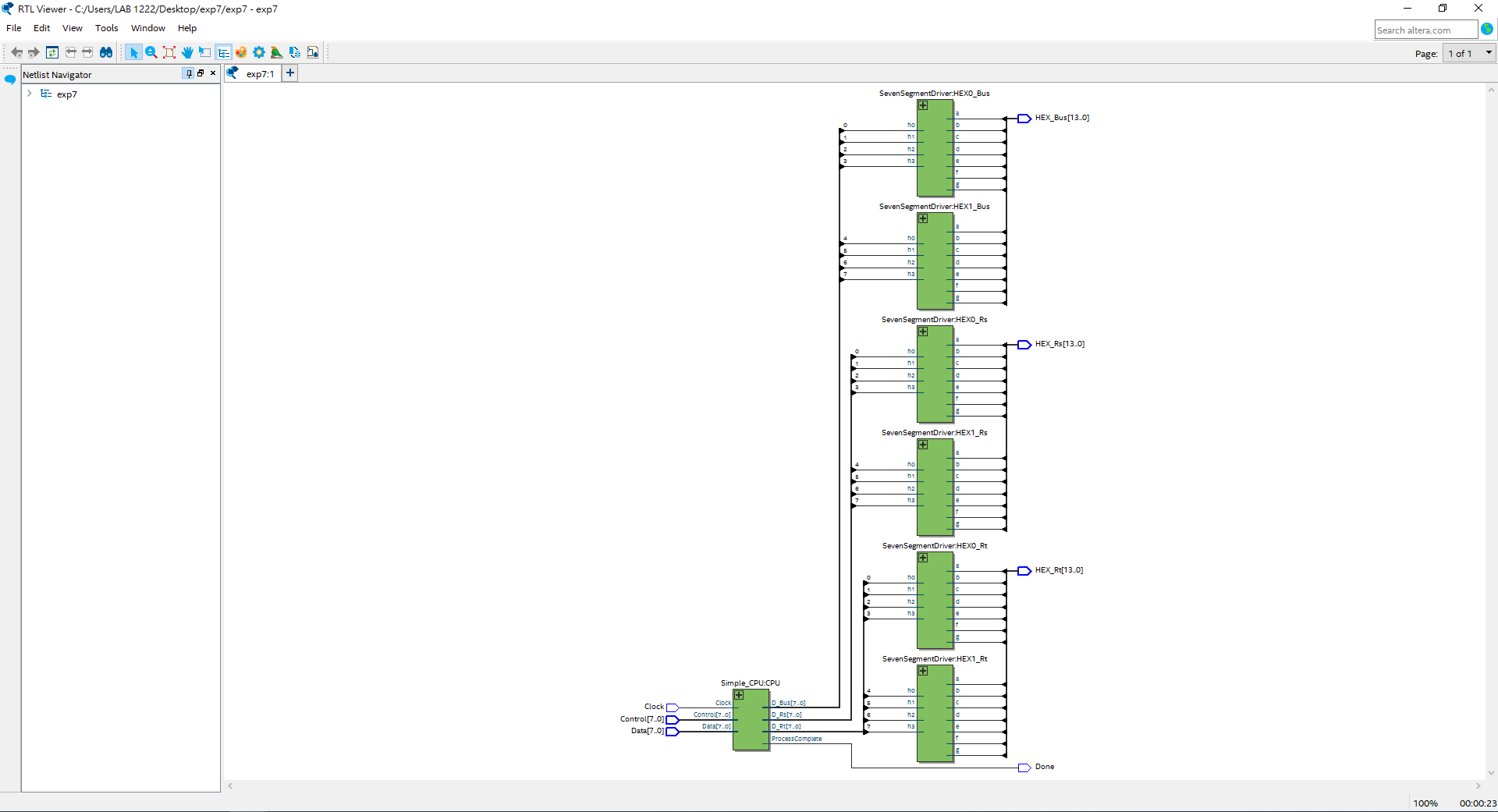
-- WHEN OTHERS => -- Load, Move

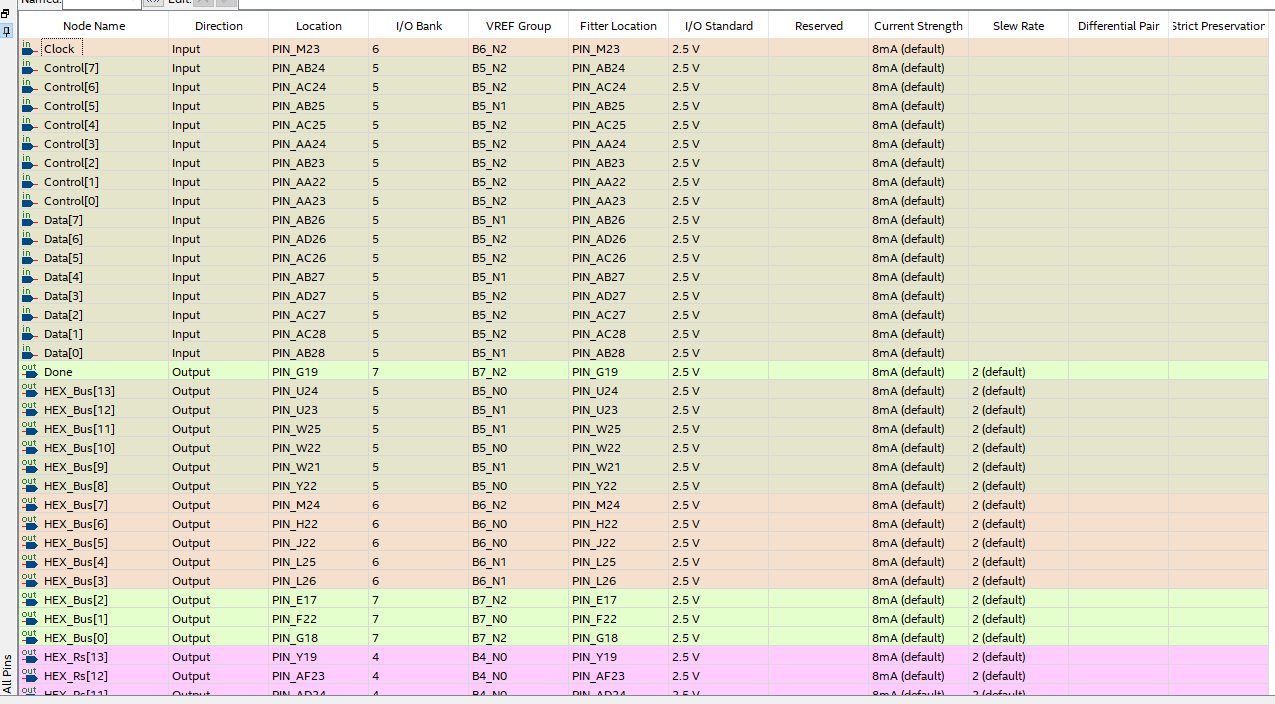
-- END CASE ;

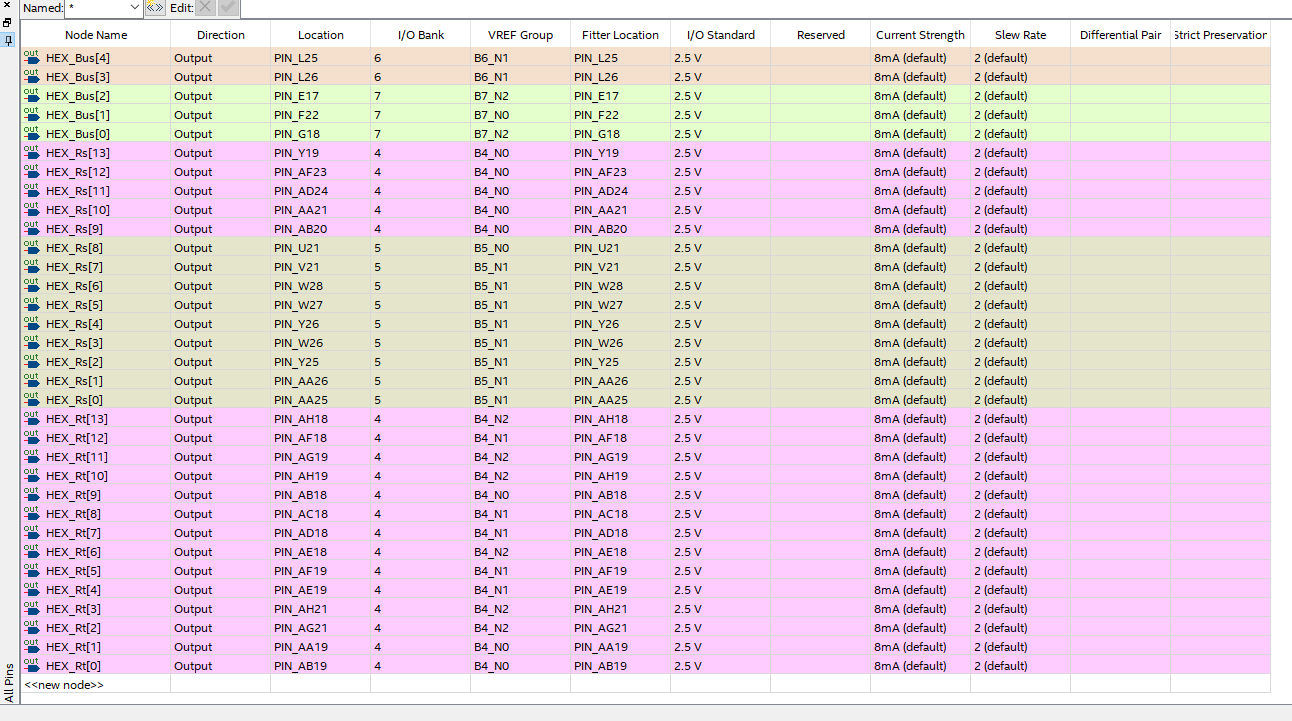
-- WHEN OTHERS => -- define signals asserted in time step T3

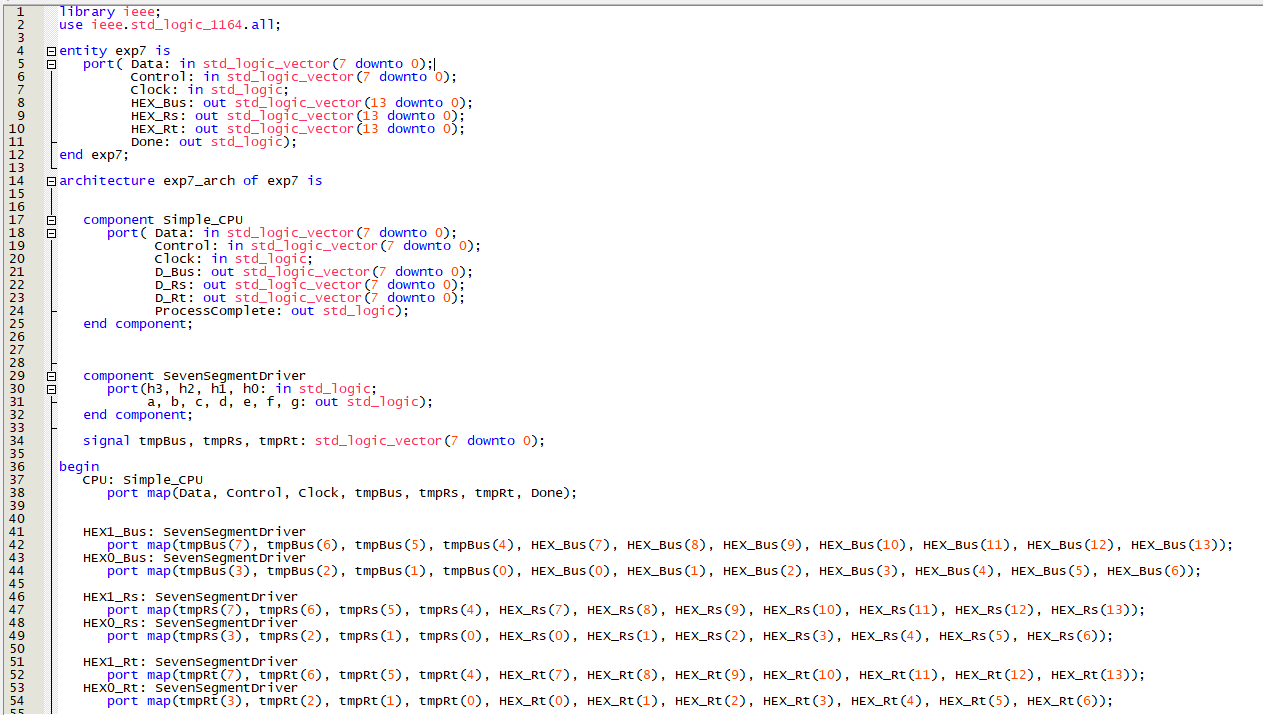
-- CASE I IS

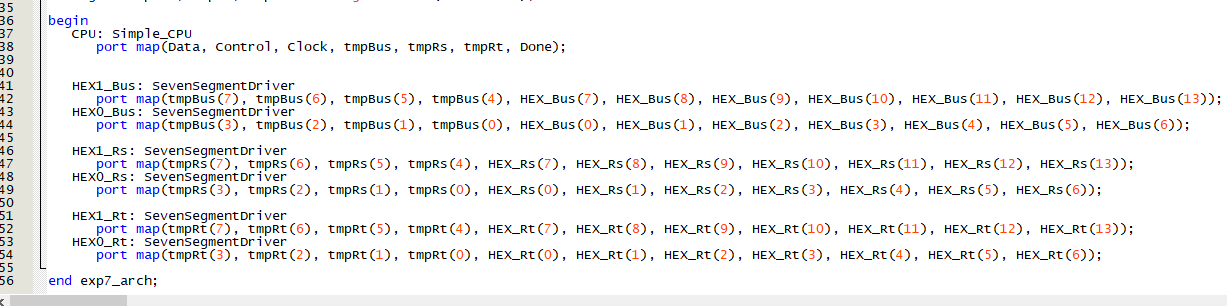
7777777777777777777777777777777777777777777777777777777777777777











library ieee;

use ieee.std\_logic\_1164.all;

entity exp7 is

port( Data: in std\_logic\_vector(7 downto 0);

Control: in std\_logic\_vector(7 downto 0);

Clock: in std\_logic;

HEX\_Bus: out std\_logic\_vector(13 downto 0);

HEX\_Rs: out std\_logic\_vector(13 downto 0);

HEX\_Rt: out std\_logic\_vector(13 downto 0);

Done: out std\_logic);

end exp7;

architecture exp7\_arch of exp7 is

component Simple\_CPU

port( Data: in std\_logic\_vector(7 downto 0);

Control: in std\_logic\_vector(7 downto 0);

Clock: in std\_logic;

D\_Bus: out std\_logic\_vector(7 downto 0);

D\_Rs: out std\_logic\_vector(7 downto 0);

D\_Rt: out std\_logic\_vector(7 downto 0);

ProcessComplete: out std\_logic);

end component;

component SevenSegmentDriver

port(h3, h2, h1, h0: in std\_logic;

a, b, c, d, e, f, g: out std\_logic);

end component;

signal tmpBus, tmpRs, tmpRt: std\_logic\_vector(7 downto 0);

begin

CPU: Simple\_CPU

port map(Data, Control, Clock, tmpBus, tmpRs, tmpRt, Done);

HEX1\_Bus: SevenSegmentDriver

port map(tmpBus(7), tmpBus(6), tmpBus(5), tmpBus(4), HEX\_Bus(7), HEX\_Bus(8), HEX\_Bus(9), HEX\_Bus(10), HEX\_Bus(11), HEX\_Bus(12), HEX\_Bus(13));

HEX0\_Bus: SevenSegmentDriver

port map(tmpBus(3), tmpBus(2), tmpBus(1), tmpBus(0), HEX\_Bus(0), HEX\_Bus(1), HEX\_Bus(2), HEX\_Bus(3), HEX\_Bus(4), HEX\_Bus(5), HEX\_Bus(6));

HEX1\_Rs: SevenSegmentDriver

port map(tmpRs(7), tmpRs(6), tmpRs(5), tmpRs(4), HEX\_Rs(7), HEX\_Rs(8), HEX\_Rs(9), HEX\_Rs(10), HEX\_Rs(11), HEX\_Rs(12), HEX\_Rs(13));

HEX0\_Rs: SevenSegmentDriver

port map(tmpRs(3), tmpRs(2), tmpRs(1), tmpRs(0), HEX\_Rs(0), HEX\_Rs(1), HEX\_Rs(2), HEX\_Rs(3), HEX\_Rs(4), HEX\_Rs(5), HEX\_Rs(6));

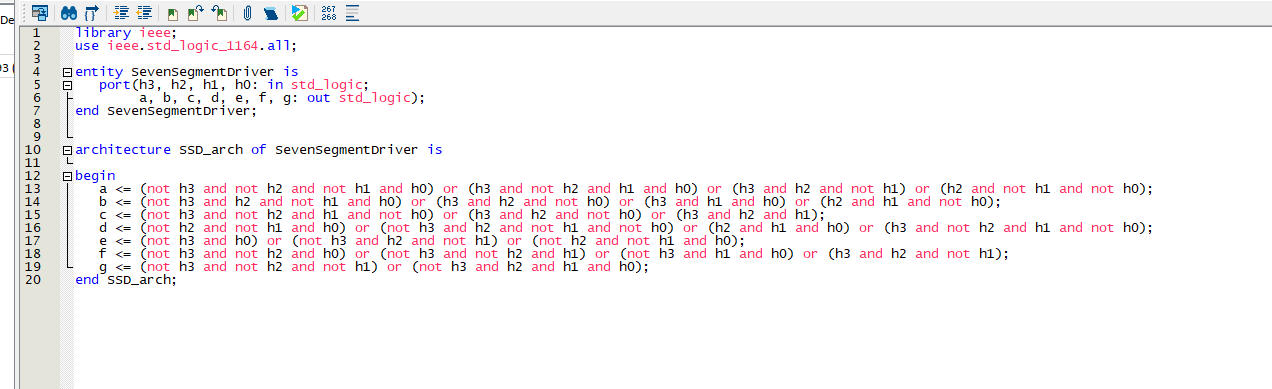
HEX1\_Rt: SevenSegmentDriver

port map(tmpRt(7), tmpRt(6), tmpRt(5), tmpRt(4), HEX\_Rt(7), HEX\_Rt(8), HEX\_Rt(9), HEX\_Rt(10), HEX\_Rt(11), HEX\_Rt(12), HEX\_Rt(13));

HEX0\_Rt: SevenSegmentDriver

port map(tmpRt(3), tmpRt(2), tmpRt(1), tmpRt(0), HEX\_Rt(0), HEX\_Rt(1), HEX\_Rt(2), HEX\_Rt(3), HEX\_Rt(4), HEX\_Rt(5), HEX\_Rt(6));

end exp7\_arch;



library ieee;

use ieee.std\_logic\_1164.all;

entity SevenSegmentDriver is

port(h3, h2, h1, h0: in std\_logic;

a, b, c, d, e, f, g: out std\_logic);

end SevenSegmentDriver;

architecture SSD\_arch of SevenSegmentDriver is

begin

a <= (not h3 and not h2 and not h1 and h0) or (h3 and not h2 and h1 and h0) or (h3 and h2 and not h1) or (h2 and not h1 and not h0);

b <= (not h3 and h2 and not h1 and h0) or (h3 and h2 and not h0) or (h3 and h1 and h0) or (h2 and h1 and not h0);

c <= (not h3 and not h2 and h1 and not h0) or (h3 and h2 and not h0) or (h3 and h2 and h1);

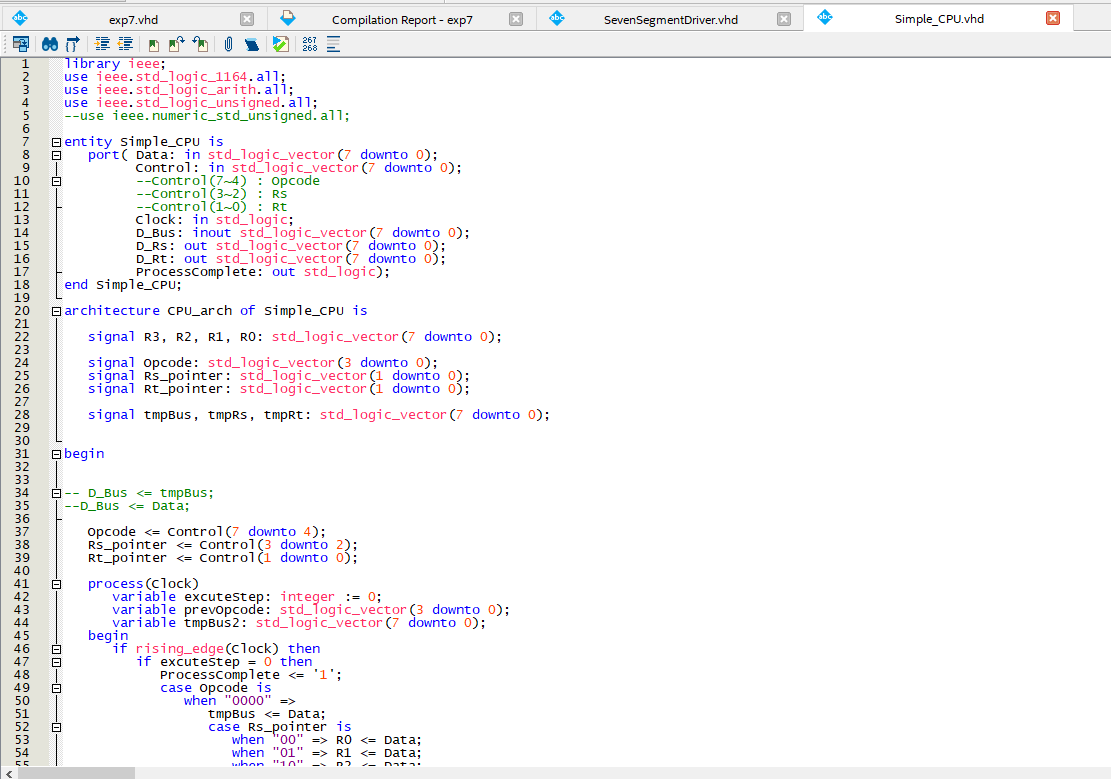
d <= (not h2 and not h1 and h0) or (not h3 and h2 and not h1 and not h0) or (h2 and h1 and h0) or (h3 and not h2 and h1 and not h0);

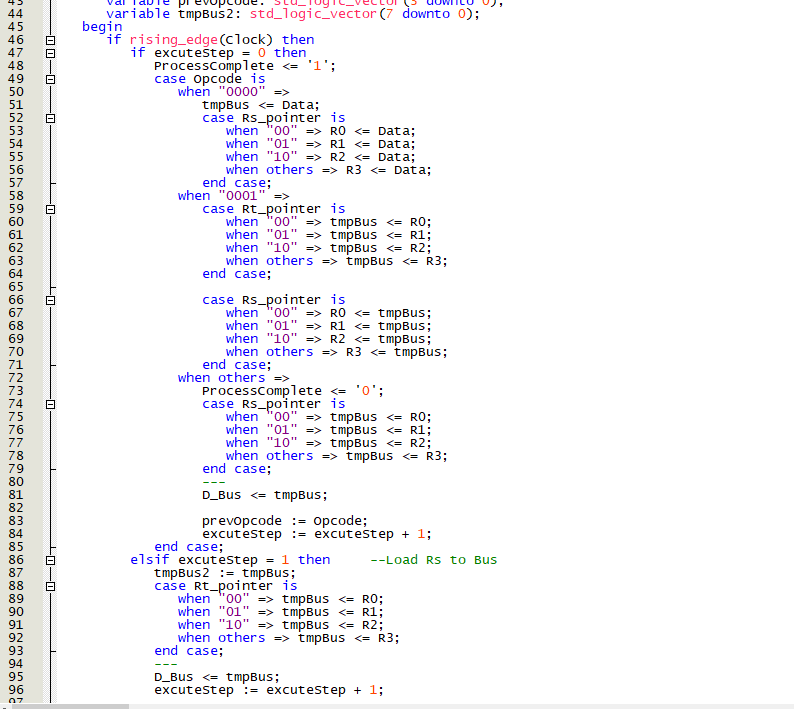
e <= (not h3 and h0) or (not h3 and h2 and not h1) or (not h2 and not h1 and h0);

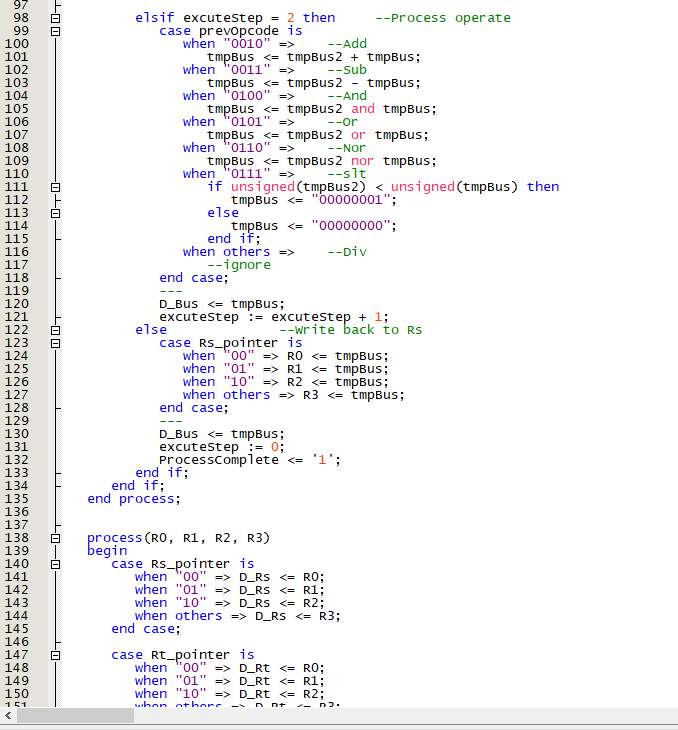
f <= (not h3 and not h2 and h0) or (not h3 and not h2 and h1) or (not h3 and h1 and h0) or (h3 and h2 and not h1);

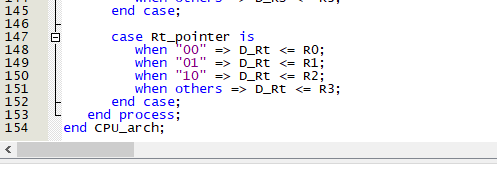
g <= (not h3 and not h2 and not h1) or (not h3 and h2 and h1 and h0);

end SSD\_arch;









library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

--use ieee.numeric\_std\_unsigned.all;

entity Simple\_CPU is

port( Data: in std\_logic\_vector(7 downto 0);

Control: in std\_logic\_vector(7 downto 0);

--Control(7~4) : Opcode

--Control(3~2) : Rs

--Control(1~0) : Rt

Clock: in std\_logic;

D\_Bus: inout std\_logic\_vector(7 downto 0);

D\_Rs: out std\_logic\_vector(7 downto 0);

D\_Rt: out std\_logic\_vector(7 downto 0);

ProcessComplete: out std\_logic);

end Simple\_CPU;

architecture CPU\_arch of Simple\_CPU is

signal R3, R2, R1, R0: std\_logic\_vector(7 downto 0);

signal Opcode: std\_logic\_vector(3 downto 0);

signal Rs\_pointer: std\_logic\_vector(1 downto 0);

signal Rt\_pointer: std\_logic\_vector(1 downto 0);

signal tmpBus, tmpRs, tmpRt: std\_logic\_vector(7 downto 0);

begin

-- D\_Bus <= tmpBus;

--D\_Bus <= Data;

Opcode <= Control(7 downto 4);

Rs\_pointer <= Control(3 downto 2);

Rt\_pointer <= Control(1 downto 0);

process(Clock)

variable excuteStep: integer := 0;

variable prevOpcode: std\_logic\_vector(3 downto 0);

variable tmpBus2: std\_logic\_vector(7 downto 0);

begin

if rising\_edge(Clock) then

if excuteStep = 0 then

ProcessComplete <= '1';

case Opcode is

when "0000" =>

tmpBus <= Data;

case Rs\_pointer is

when "00" => R0 <= Data;

when "01" => R1 <= Data;

when "10" => R2 <= Data;

when others => R3 <= Data;

end case;

when "0001" =>

case Rt\_pointer is

when "00" => tmpBus <= R0;

when "01" => tmpBus <= R1;

when "10" => tmpBus <= R2;

when others => tmpBus <= R3;

end case;

case Rs\_pointer is

when "00" => R0 <= tmpBus;

when "01" => R1 <= tmpBus;

when "10" => R2 <= tmpBus;

when others => R3 <= tmpBus;

end case;

when others =>

ProcessComplete <= '0';

case Rs\_pointer is

when "00" => tmpBus <= R0;

when "01" => tmpBus <= R1;

when "10" => tmpBus <= R2;

when others => tmpBus <= R3;

end case;

---

D\_Bus <= tmpBus;

prevOpcode := Opcode;

excuteStep := excuteStep + 1;

end case;

elsif excuteStep = 1 then --Load Rs to Bus

tmpBus2 := tmpBus;

case Rt\_pointer is

when "00" => tmpBus <= R0;

when "01" => tmpBus <= R1;

when "10" => tmpBus <= R2;

when others => tmpBus <= R3;

end case;

---

D\_Bus <= tmpBus;

excuteStep := excuteStep + 1;

elsif excuteStep = 2 then --Process operate

case prevOpcode is

when "0010" => --Add

tmpBus <= tmpBus2 + tmpBus;

when "0011" => --Sub

tmpBus <= tmpBus2 - tmpBus;

when "0100" => --And

tmpBus <= tmpBus2 and tmpBus;

when "0101" => --Or

tmpBus <= tmpBus2 or tmpBus;

when "0110" => --Nor

tmpBus <= tmpBus2 nor tmpBus;

when "0111" => --slt

if unsigned(tmpBus2) < unsigned(tmpBus) then

tmpBus <= "00000001";

else

tmpBus <= "00000000";

end if;

when others => --Div

--ignore

end case;

---

D\_Bus <= tmpBus;

excuteStep := excuteStep + 1;

else --Write back to Rs

case Rs\_pointer is

when "00" => R0 <= tmpBus;

when "01" => R1 <= tmpBus;

when "10" => R2 <= tmpBus;

when others => R3 <= tmpBus;

end case;

---

D\_Bus <= tmpBus;

excuteStep := 0;

ProcessComplete <= '1';

end if;

end if;

end process;

process(R0, R1, R2, R3)

begin

case Rs\_pointer is

when "00" => D\_Rs <= R0;

when "01" => D\_Rs <= R1;

when "10" => D\_Rs <= R2;

when others => D\_Rs <= R3;

end case;

case Rt\_pointer is

when "00" => D\_Rt <= R0;

when "01" => D\_Rt <= R1;

when "10" => D\_Rt <= R2;

when others => D\_Rt <= R3;

end case;

end process;

end CPU\_arch;