LAB7 Final-Report

20160410 윤석훈

(1)Source Code

Library IEEE;  
Use IEEE.std\_logic\_1164.all;  
Entity HA is  
 port( x,y : In std\_logic;  
   s,c : Out Std\_logic);  
END Entity;

Architecture Logic of HA is  
begin  
 s<= x XOR y;  
 c<= x AND y;  
end Logic;

Library IEEE;  
Use IEEE.std\_logic\_1164.all;

Entity FA is  
 port (Cinp, A, B: In std\_logic;  
   S, Coutp: Out std\_logic);  
END Entity;

Architecture logic of FA is  
component HA  
 port( x,y : In std\_logic;  
   s,c : Out Std\_logic);  
end component;  
signal C: std\_logic;  
signal D: std\_logic;  
signal E: std\_logic;  
signal F: std\_logic;  
begin   
U0: HA port map  
(x=>A, y=>B, s=>C, c=>D);  
U1: HA port map  
(x=>Cinp, y=>C, s=>E, c=>F);  
Coutp<=F or D;  
S<= E;  
end architecture;

Library IEEE;  
Use IEEE.std\_logic\_1164.all;  
Entity Add4 is  
 port(Cin : in std\_logic;  
  x3, x2, x1, x0: in std\_logic;  
  y3, y2, y1, y0: in std\_logic;  
  s3, s2, s1, s0: out std\_logic;  
  Cout: out std\_logic);  
End Entity;

architecture logic of Add4 is  
component FA  
 port (Cinp, A, B: In std\_logic;  
   S, Coutp: Out std\_logic);  
end component;  
signal C1,C2,C3: std\_logic;  
begin  
U0: FA PORT MAP (Cin, x0, y0, s0, C1);  
U1: FA PORT MAP (C1, x1,y1,s1,C2);  
U2: FA PORT MAP (C2, x2,y2,s2,C3);  
U3: FA PORT MAP (Cinp=>C3, Coutp=>Cout,A=>x3,B=>y3,S=>s3);  
END Architecture;

Library IEEE;  
Use IEEE.std\_logic\_1164.all;  
Entity Add5 is  
 port(Cin : in std\_logic;  
  x4, x3, x2, x1, x0: in std\_logic;  
  y4, y3, y2, y1, y0: in std\_logic;  
  s4, s3, s2, s1, s0: out std\_logic;  
  Cout: out std\_logic);  
End Entity;

architecture logic of Add5 is  
component FA  
 port (Cinp, A, B: In std\_logic;  
   S, Coutp: Out std\_logic);  
end component;  
signal C1,C2,C3,C4: std\_logic;  
begin  
U0: FA PORT MAP (Cin, x0, y0, s0, C1);  
U1: FA PORT MAP (C1, x1,y1,s1,C2);  
U2: FA PORT MAP (C2, x2,y2,s2,C3);  
U3: FA PORT MAP (C3, x3,y3,s3,C4);  
U4: FA PORT MAP (Cinp=>C4, Coutp=>Cout,A=>x4,B=>y4,S=>s4);  
END Architecture;

Library IEEE;  
Use IEEE.std\_logic\_1164.all;  
Entity FourBitBinaryAddition is  
port(X,Y: in std\_logic\_vector(3 downto 0);  
  ANS: out std\_logic\_vector(3 downto 0));  
end Entity;

architecture behavior of FourBitBinaryAddition is  
component Add4  
port(Cin : in std\_logic;  
  x3, x2, x1, x0: in std\_logic;  
  y3, y2, y1, y0: in std\_logic;  
  s3, s2, s1, s0: out std\_logic;  
  Cout: out std\_logic);  
end component;  
signal ABD: std\_logic;  
begin  
U0: Add4 PORT MAP (Cin=>'0', x3=> X(3),x2=> X(2),x1=> X(1),x0=>X(0),  
    y3=>Y(3),y2=>Y(2),y1=>Y(1),y0=>Y(0),  
    s3=>ANS(3),s2=>ANS(2),s1=>ANS(1),s0=>ANS(0), Cout=>ABD);  
END architecture;

Library IEEE;  
Use IEEE.std\_logic\_1164.all;  
Entity FiveBitBinaryAddition is  
port(X,Y: in std\_logic\_vector(4 downto 0);  
  ANS: out std\_logic\_vector(4 downto 0));  
end Entity;

architecture logic of FiveBitBinaryAddition is  
component Add5  
port(Cin : in std\_logic;  
  x4, x3, x2, x1, x0: in std\_logic;  
  y4, y3, y2, y1, y0: in std\_logic;  
  s4, s3, s2, s1, s0: out std\_logic;  
  Cout: out std\_logic);  
end component;  
signal ABD: std\_logic;  
begin  
U0: Add5 PORT MAP (Cin=>'0', x4=> X(4),x3=> X(3),x2=> X(2),x1=> X(1),x0=>X(0),  
     y4=>Y(4),y3=>Y(3),y2=>Y(2),y1=>Y(1),y0=>Y(0),  
     s4=>ANS(4),s3=>ANS(3),s2=>ANS(2),s1=>ANS(1),s0=>ANS(0), Cout=>ABD);  
END architecture;

Library ieee;  
Use ieee.std\_logic\_1164.all;  
Use ieee.std\_logic\_unsigned.all;

Entity BCDADDER is  
 PORT (X,Y: in std\_logic\_vector(7 downto 0);  
  S: OUT std\_logic\_vector(11 downto 0));  
END BCDADDER;

Architecture behavior of BCDADDER is  
 Component FiveBitBinaryAddition  
 port(X, Y: in std\_logic\_vector (4 downto 0); Ans: out std\_logic\_vector (4 downto 0));  
 end component;  
 Signal A,B,C,D,E,F,G,H,I ,J,K: std\_logic\_vector(4 downto 0);  
 Signal carry1,carry2 : std\_logic;    
Begin  
  A<='0' & X(3 downto 0);  
 B<='0' & Y(3 downto 0);  
 U0: FiveBitBinaryAddition PORT MAP(X=>A,Y=>B,ANS=>C);  
 Carry1<='1' when C>9 ELSE '0';  
 U1: FiveBitBinaryAddition PORT MAP(X=>C,Y=>"00110",ANS=>D);  
 S(3 downto 0) <= C(3 downto 0) WHEN (Carry1='0') ELSE D(3 downto 0);

 H<="0000" & D(4);  
 E<="0" & X(7 downto 4);  
 F<="0" & Y(7 downto 4);  
 U2: FiveBitBinaryAddition PORT MAP(X=>E,Y=>F,ANS=>G);  
 U3: FiveBitBinaryAddition PORT MAP(X=>G,Y=>H,ANS=>I);  
 Carry2<='1'when I>9 ELSE '0';  
 U4: FiveBitBinaryAddition PORT MAP(X=>I,Y=>"00110",ANS=>J);  
 S(7 downto 4) <= I(3 downto 0) WHEN (Carry2='0') ELSE J(3 downto 0);  
   
 K<="0000" & J(4);  
 S(11 downto 8) <= K(3 downto 0);

End Architecture;

Library IEEE;  
Use IEEE.std\_logic\_1164.all;  
Use IEEE.std\_logic\_unsigned.all;  
Entity tensComplement is  
port(A: in std\_logic\_vector(7 downto 0);  
 B: out std\_logic\_vector(7 downto 0));  
end entity;

Architecture behavior of tensComplement is  
component FourBitBinaryAddition  
port(X,Y: in std\_logic\_vector(3 downto 0);  
  ANS: out std\_logic\_vector(3 downto 0));

end component;  
component BCDADDER  
PORT (X,Y: in std\_logic\_vector(7 downto 0);  
  S: OUT std\_logic\_vector(11 downto 0));  
end component;

signal One,Ten,V,W,OneV,TenW: std\_logic\_vector(3 downto 0);  
signal C: std\_logic\_vector(7 downto 0);  
signal D: std\_logic\_vector(11 downto 0);  
begin  
 One<=A(3 downto 0);  
 Ten <=A(7 downto 4);  
 U0: FourBitBinaryAddition PORT MAP(X=>"1001", Y=>(not One), ANS=>V);  
 U1: FourBitBinaryAddition PORT MAP(X=>V,Y=>"0001", ANS=>OneV);  
  U2: FourBitBinaryAddition PORT MAP(X=>"1001", Y=>(not Ten), ANS=>W);  
 U3: FourBitBinaryAddition PORT MAP(X=>W, Y=>"0001", ANS=>TenW);  
 C(3 downto 0)<=OneV;  
 C(7 downto 4)<=TenW;  
 U4: BCDADDER PORT MAP(X=>C, Y=>"00000001",S=>D);  
 B<=D(7 downto 0);  
end architecture;

Library IEEE;  
Use IEEE.std\_logic\_1164.all;  
Use IEEE.std\_logic\_unsigned.all;

Entity LAB7 is  
port(  
 sw: in std\_logic\_vector(17 downto 0);  
 ledr: out std\_logic\_vector(11 downto 0);  
 HEX0: out std\_logic\_vector (6 downto 0);  
 HEX1: out std\_logic\_vector (6 downto 0);  
 HEX2: out std\_logic\_vector (6 downto 0);  
 HEX3: out std\_logic\_vector (6 downto 0);  
 HEX4: out std\_logic\_vector (6 downto 0);  
 HEX5: out std\_logic\_vector (6 downto 0);  
 HEX6: out std\_logic\_vector (6 downto 0);  
 HEX7: out std\_logic\_vector (6 downto 0)  
 );

END Entity;

architecture LB7 of LAB7 is  
component BCDADDER   
PORT (X,Y: in std\_logic\_vector(7 downto 0);  
  S: OUT std\_logic\_vector(11 downto 0));  
end component;

component tensComplement  
PORT(A: in std\_logic\_vector(7 downto 0);  
 B: out std\_logic\_vector(7 downto 0));  
end component;

signal ANS : std\_logic\_vector(7 downto 0);  
signal H2,H1,H0: std\_logic\_vector(3 downto 0);  
signal SUM: std\_logic\_vector(11 downto 0);  
signal SUB1,SUB2: std\_logic\_vector(11 downto 0);  
signal compA, compB : std\_logic\_vector(7 downto 0);  
function Matching (digit: in std\_logic\_vector(3 downto 0))  
  return std\_logic\_vector is  
  begin  
   if digit="0000" then return "1000000";  
   elsif digit="0001" then return "1111001";  
   elsif digit="0010" then return "0100100";  
   elsif digit="0011" then return "0110000";  
   elsif digit="0100" then return "0011001";  
   elsif digit="0101" then return "0010010";  
   elsif digit="0110" then return "0000010";  
   elsif digit="0111" then return "1111000";  
   elsif digit="1000" then return "0000000";  
   elsif digit="1001" then return "0011000";  
   else   
    return "1111111";  
   end if;  
  END Matching;

begin  
 HEX7<=Matching(sw(15 downto 12));  
 HEX6<=Matching(sw(11 downto 8));  
 HEX5<=Matching(sw(7 downto 4));  
 HEX4<=Matching(sw(3 downto 0));  
   
 U0: BCDADDER port map(X=>sw(15 downto 8), Y=>sw(7 downto 0), S=>SUM);  
 U1: tensComplement port map(A=> sw(7 downto 0), B=> compB);  
 U2: BCDADDER port map(X=>sw(15 downto 8), Y=> compB, S=>SUB1);  
 U3: tensComplement port map(A=> sw(15 downto 8), B=> compA);  
 U4: BCDADDER port map(X=>sw(7 downto 0), Y=> compA, S=>SUB2);  
   
 process(sw,SUM,SUB1,SUB2)  
 begin  
  if(sw(15 downto 8) > sw(7 downto 0)) Then  
   HEX3<="1110000";  
  elsif(sw(15 downto 8) = sw(7 downto 0)) Then  
   HEX3<="0110111";  
  else  
   HEX3<="1000110";  
  End If;  
     
  if (sw(17)='0') Then  
   ledr(11 downto 0) <= SUM(11 downto 0);  
   HEX2<=Matching(SUM(11 downto 8));  
   HEX1<=Matching(SUM(7 downto 4));  
   HEX0<=Matching(SUM(3 downto 0));  
  else  
   if(sw(16)='0') Then  
    if(sw(15 downto 8)>=sw(7 downto 0)) Then  
     ledr(11 downto 0) <= SUB1(11 downto 0);  
     HEX2<="1111111";  
     HEX1<=Matching(SUB1(7 downto 4));  
     HEX0<=Matching(SUB1(3 downto 0));  
    else  
     ledr(11 downto 0) <= SUB2(11 downto 0);  
     HEX2<="0111111";  
     HEX1<=Matching(SUB2(7 downto 4));  
     HEX0<=Matching(SUB2(3 downto 0));  
    end if;  
   else  
    if(sw(7 downto 0)>=sw(15 downto 8)) Then  
     ledr(11 downto 0) <= SUB2(11 downto 0);  
     HEX2<="1111111";  
     HEX1<=Matching(SUB2(7 downto 4));  
     HEX0<=Matching(SUB2(3 downto 0));  
    else  
     ledr(11 downto 0) <= SUB1(11 downto 0);  
     HEX2<="0111111";  
     HEX1<=Matching(SUB1(7 downto 4));  
     HEX0<=Matching(SUB1(3 downto 0));  
    End If;  
   End If;  
  end if;

 End Process;  
End architecture;

(2) Explanation: 처음 HA부터 FourBitBinaryAddition 엔티티까지는 BCDAdder 컴포넌트와 TensComplement 컴포넌트에서 쓰이는 비트덧셈 연산을 위해 선언되었다.

Full Adder와 ripple-carry adder들로 구현된 FourBitBinaryAddition과 FiveBitBinaryAddition은 각각 4,5 비트의 변수 2개를 입력 받아 그것들의 4,5 비트 합을 반환하는 것이다. 여기까지의 구체적인 내용은 LAB5에서 다루어졌기에 생략한다.

BCDAdder 컴포넌트에서는 두 개의 8비트의 BCD입력과 한 개의 11비트 BCD 출력을 가진다. BCDAdder에서는 5비트 덧셈으로 각 자릿수의 합을 계산하는데, 이때 4비트 자릿수 계산에서 4비트 덧셈이 아닌 5비트 덧셈을 사용한 이유는 Carry의 자리를 고려하기 위함이었다. 일의 자리 합이 9보다 큰 경우에는 6을 더한 뒤 Carry에 1을 할당하면, 일의 자리는 0~9중 적절한 값으로 할당된다. 마찬가지로 십의 자리에서도 십의 자릿수 덧셈을 하는데, 여기서는 일의 자릿수 계산에서 발생한 Carry도 함께 더한다. 최종적인 결과값은 십의 자릿수 계산에서 발생한 Carry를 출력의 8~11번째 비트에 할당하고, 십의 자릿수를 출력의 4~7번째 비트, 일의 자릿수를 출력의 0~3번째 비트에 할당하여 구한다.

TensComplement 컴포넌트에서는 한 개의 8비트 BCD입력과 그것의 10의 보수인 8비트를 출력한다. 어떤 두 자리 정수의 10의 보수는 각 자릿수를 각 자릿수의 값과 9와의 차이로 할당하고, 그것에 1을 더함으로써 얻어진다. “1001” (9) – 자릿수 = “1001” + (not 자릿수) +1 이므로, FourBitBinaryAddition으로 각 자릿수와 9와의 차이를 구할 수 있다. 이렇게 각 자릿수를 반전시킨 후에 생성된 8비트 BCD를 BCDADDER 컴포넌트를 이용해 “00000001” (1) 과 더해준다. 그러면 10의 보수가 완성된다.

Main Entity에서는 위의 BCDADDER와 TENScomplement를 이용해 BCD Operation을 한다. A(sw(15 downto 8))와 B(sw(7 downto 0))의 합과 차를 미리 구해놓는데, 차는 A-B와 B-A의 두 종류가 있을 수 있으므로 SUB1, SUB2 의 두 변수를 정의한다.

BCDADDER로 A와 B를 더하면 합인 SUM이 구해진다. BCDADDER로 A와 (B의 10의 보수)를 더하면 A-B인 SUB1이 구해진다. BCDADDER로 B와 (A의 10의 보수)를 더하면 B-A인 SUB2이 구해진다.

SW(17)=0일 때(합)는 HEX2~0에 SUM을 4비트씩 할당하여 Matching한다. SW(17)=1일 때(차)는 SW(16)=0(A-B)와 SW(16)=1(B-A)일 때를 나누어 생각한다. A-B일 때, A>=B이면 A-B는 양수이므로 SUB1의 값을 그대로 HEX1~0에 할당하면 된다. 그런데, A<B이면 A-B가 음수가 되는데 이때에는 B-A의 값(SUB2)를 HEX1~0에 할당한 후 HEX2에 ‘-‘ 표시를 Matching하면 결론적으로 A-B의 값이 표시된다. SW(16)=1 인 반대의 경우도 마찬가지로 하면 된다.

(3)Difficult Part

BCDAdder 컴포넌트에서 두 자리 정수의 BCD 덧셈을 구현했는데, 겨우 두 자리 정수였지만 일의 자리와 십의 자리에서 발생하는 Carry를 고려하기 위해 각 단계(일의 자리/ 십의 자리)마다 6~7줄의 코드와 5개 남짓의 변수가 쓰이게 되었다. 여기에서 어떤 변수를 어떻게 할당할지, Carry를 어떻게 고려해야 할지 생각하는 것이 어려웠다. 특히, 만약 문제에서 BCDAdder가 4자리 혹은 8자리 이상의 BCD 입력을 요구했다면 이런 각 단계별 수작업은 많은 매우 복잡한 작업을 요했을 것이다. 그런 경우, loop를 이용하기 위해 어떻게 좀 더 일반화된 작업을 할 수 있을지에 대해 생각해보게 되었다.