LAB5 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 Add8 is

port(Cin : in std\_logic;

x7, x6, x5, x4, x3, x2, x1, x0: in std\_logic;

y7, y6, y5, y4, y3, y2, y1, y0: in std\_logic;

s7, s6, s5, s4, s3, s2, s1, s0: out std\_logic;

Cout: out std\_logic);

End Entity;

architecture logic of Add8 is

component FA

port (Cinp, A, B: In std\_logic;

S, Coutp: Out std\_logic);

end component;

signal C1,C2,C3,C4,C5,C6,C7: 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 (C4, x4,y4,s4,C5);

U5: FA PORT MAP (C5, x5,y5,s5,C6);

U6: FA PORT MAP (C6, x6,y6,s6,C7);

U7: FA PORT MAP (Cinp=>C7, Coutp=>Cout,A=>x7,B=>y7,S=>s7);

END Architecture;

Library IEEE;

Use IEEE.std\_logic\_1164.all;

Entity Addition is

port(X,Y: in std\_logic\_vector(7 downto 0);

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

end Entity;

architecture logic of Addition is

component Add8

port(Cin : in std\_logic;

x7, x6, x5, x4, x3, x2, x1, x0: in std\_logic;

y7, y6, y5, y4, y3, y2, y1, y0: in std\_logic;

s7, s6, s5, s4, s3, s2, s1, s0: out std\_logic;

Cout: out std\_logic);

end component;

signal ABD: std\_logic;

begin

U0: Add8 PORT MAP (Cin=>'0', x7=>X(7), x6=> X(6),x5=> X(5),x4=> X(4),x3=> X(3),x2=> X(2),x1=> X(1),x0=>X(0),

y7=>Y(7),y6=>Y(6),y5=>Y(5),y4=>Y(4),y3=>Y(3),y2=>Y(2),y1=>Y(1),y0=>Y(0),

s7=>ANS(7),s6=>ANS(6),s5=>ANS(5),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;

Entity Comp is

port(X,Y : in std\_logic\_vector(7 downto 0);

P,N,E: out std\_logic);

end Entity;

architecture logic of Comp is

component Addition

port(X,Y : in std\_logic\_vector(7 downto 0);

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

end component;

signal notY: std\_logic\_vector(7 downto 0);

signal Var1: std\_logic\_vector(7 downto 0);

signal Var2: std\_logic\_vector(7 downto 0);

begin

notY<= not Y;

U0: Addition PORT MAP(X=>X, Y=>notY, ANS=>Var1);

U1: Addition PORT MAP(X=>Var1, Y=>"00000001", ANS=>Var2);

process(Var2,X,Y)

begin

if((((not X) and "10000000")="10000000") and ((Y and "10000000")="10000000")) Then

P<='1';

N<='0';

E<='0';

elsif(((X and "10000000")="10000000") and (((not Y) and "10000000")="10000000")) Then

P<='0';

N<='1';

E<='0';

elsif((Var2 and "10000000")="10000000") Then

P<='0';

N<='1';

E<='0';

elsif(Var2="00000000") Then

P<='0';

E<='1';

N<='0';

else

P<='1';

E<='0';

N<='0';

end if;

end process;

end architecture;

Library IEEE;

Use IEEE.std\_logic\_1164.all;

Entity LAB5 is

port(

sw: in std\_logic\_vector(17 downto 0);

ledr: out std\_logic\_vector(7 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 LB5 of LAB5 is

signal ANS : std\_logic\_vector(7 downto 0);

signal ADD : std\_logic\_vector(7 downto 0);

signal SUB1, SUB2 : std\_logic\_vector(7 downto 0);

signal Z1,Z2: std\_logic\_vector(7 downto 0);

signal P,N,E: std\_logic;

component Addition

port(X,Y : in std\_logic\_vector(7 downto 0);

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

end component;

component Comp

port(X,Y : in std\_logic\_vector(7 downto 0);

P,N,E: out std\_logic);

end component;

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";

elsif digit="1010" then return "0001000";

elsif digit="1011" then return "0000011";

elsif digit="1100" then return "1000110";

elsif digit="1101" then return "0100001";

elsif digit="1110" then return "0000110";

elsif digit="1111" then return "0001110";

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));

HEX2<= "1111111";

U0: Addition PORT MAP(X=>sw(15 downto 8), Y=>sw(7 downto 0), ANS=>ADD);

U1: Comp PORT MAP(X=>sw(15 downto 8), Y=>sw(7 downto 0),P=>P, N=>N, E=>E);

U2: Addition PORT MAP(X=>(not sw(15 downto 8)), Y=>"00000001", ANS=>Z1);

U3: Addition PORT MAP(X=>(not sw(7 downto 0)), Y=>"00000001", ANS=>Z2);

U4: Addition PORT MAP(X=>sw(15 downto 8), Y=>Z2, ANS=>SUB1);

U5: Addition PORT MAP(X=>sw(7 downto 0), Y=>Z1, ANS=>SUB2);

process(sw,P,N,E)

begin

if(P= '1') Then

HEX3<="1110000";

elsif(E= '1') Then

HEX3<="0110111";

else

HEX3<="1000110";

End If;

if (sw(17)='0') Then

ledr(7 downto 0)<= ADD;

ANS(7 downto 0)<= ADD;

else

if(sw(16)='0') Then

ledr(7 downto 0)<= SUB1;

ANS(7 downto 0)<= SUB1;

else

ledr(7 downto 0)<= SUB2;

ANS(7 downto 0)<= SUB2;

End If;

end if;

HEX1<= Matching(ANS(7 downto 4));

HEX0<= Matching(ANS(3 downto 0));

End Process;

End architecture;

(2) Explanation

처음 HA라는 엔티티에서는 1비트 덧셈에서 sum과 carry를 출력하는 Half Adder를 구현한다.

그 다음 FA라는 엔티티에서는 HA를 Component로 2번 이용해서 Input-Carry와 2개의 1비트를 받으면 Output-Carry와 Sum을 출력하는 Full Adder를 구현한다. 그 다음 Add8이라는 엔티티에서는 8비트 덧셈을 구현하는데, 입력을 16개의 1비트로, 출력을 8개의 1비트로 설정한다. 이는 8비트 입력 2개와 그 두개가 더해져서 나오는 8비트 출력 1개를 모두 1비트씩 입력하고 출력하기 위함이다. Add8에서는 FA를 Component로 8번 이용해서 8비트 덧셈을 수행하고, 더해진 8비트 합을 각 1비트씩 변수에 할당하여 출력한다.

이제 Add8을 Component로 이용하여 최종적으로 Addition 엔티티를 구현한다. 이것은 덧셈을 할 8비트벡터 2개를 입력으로 받고 그 합을 8비트벡터로 출력하는데, 각 벡터의 비트들을 Add8에 Mapping하여 더해진 결과인 8개의 1비트를 8비트벡터에 차례대로 입력한다.

Comp라는 엔티티에서는 2개의 8비트벡터의 Two's complement 표현에서의 대소를 비교한다. X가 Y보다 크다면 P에 1을 할당하고, 같다면 E에 1을, 작다면 N에 1을 할당한다. P, N,E중 하나에 1이 할당되었다면 나머지는 모두 0으로 할당한다. 두 수의 대소는 작거나 크거나 같거나 셋 중의 하나이기 때문이다. 기본적으로 X-Y의 부호에 따라 X와 Y의 대소를 비교하는데, 이때 부호는 X-Y의 MSB가 1이면 음수이고 0이면 양수, X-Y가 "00000000"이면 0인것으로 판단한다.

그런데 그보다 먼저, X와 Y의 부호가 다르다면 X-Y에서 Overflow가 발생할 수도 있다는 것을 따져야 한다. X가 양수이고, Y가 음수이면 X가 Y보다 큰 것은 자명한데, X-Y에서 Overflow가 일어나서 이 값이 음수가 되면 X가 Y보다 작다고 출력되기 때문이다. 반대의 경우도 마찬가지로, X가 음수이고 Y가 양수이면 Y가 X보다 큰 것은 자명한데, X-Y에서 Overflow가 일어나서 양수로 출력된다면 X가 Y보다 큰 것으로 출력될 수 있다. X와 Y가 부호가 같다면 X-Y에서 Overflow가 일어나지 않으므로 X-Y의 부호에 따라 X와 Y의 대소를 비교할 수 있다.

따라서, X와 Y의 부호가 다른 경우는 X-Y로 인한 Overflow가 발생하지 않도록 자명한 결과를 P, N, E에 할당한다. X, Y의 부호가 같은 경우에는 X-Y의 부호에 따라 P, N, E를 할당하는데, 이때 X-Y는 X+(~Y)+1로 계산한다. 왜냐하면 임의의 8비트 벡터 Y에 대해, Y+(~Y)=11111111(2)인데, 이때 11111111(2)=-1이므로, Y+(~Y)=-1이고, 이항하면 -Y=1+(~Y)가 되기 때문에 X-Y=X+(~Y)+1임을 알 수 있기 때문이다. Comp Entity에서 Addition을 Component로 이용하는 것도 X+(~Y)+1를 계산하기 위함이다.

이제 본 Entity인 LAB5에서, 18개의 스위치 sw(0~17)와 8개의 LED ledr(0~7)를 입력으로 하고, 8개의 숫자판 HEX0~HEX7을 출력으로 지정한다. 위에서 설명한 Entity중 Addition과 Comp를 Component로 지정한 후, 지난번 LAB에서와 같이 숫자를 숫자판에 대응시키기 위한 Matching함수도 선언한다. 두 입력값 sw(15 downto 8)와 sw(7 downto 0)를 더한 값과 뺀 값을 미리 8비트벡터 Signal ADD, SUB1(X-Y 일때), SUB2(Y-X 일때)에 Mapping을 통하여 할당하고, 그 대소까지 Signal P,N,E에 Comp Component를 통한 Mapping으로 할당한다.

HEX 4~7에는 별다른 조작 없이 스위치 입력값을 4개씩 HEXADECIMAL로 매칭시키면 된다. HEX 3에는 P,N,E의 값에 따라 ],=,[ 의 숫자판 모양 7비트를 할당한다. 덧셈의 경우 ADD값을 HEX0~1에 Hexadecimal로 매칭시키고 ledr에도 할당한다. 뺄셈의 경우 정 순서일 때 SUB1을, 반대 순서일 때 SUB2를 마찬가지로 할당하면 된다.

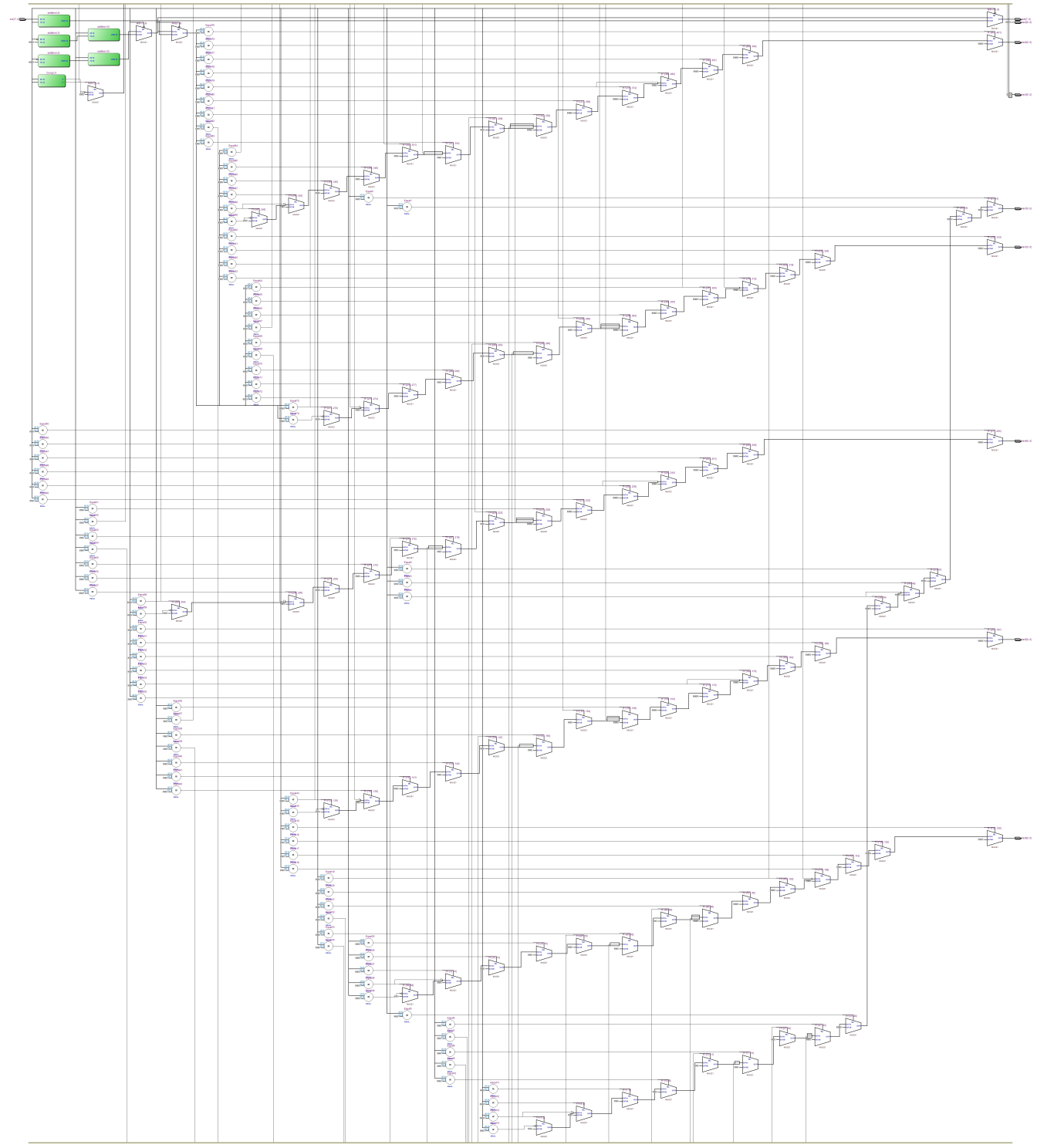
(3)Discussion how to detect overflow in 2’s Complement

2’s Complement가 표시하는 수 범위는 n비트일 때 -2^(n-1) 부터 2^(n-1)-1이다. 덧셈에서는 두 양수를 더했을 때 결과값이 2^(n-1)-1보다 커지거나, 두 음수를 더했을 때 결과값이 -2^(n-1)보다 작아질 때 Overflow가 발생한다. 뺄셈에서는 양수에서 음수를 빼었을 때 결과값이 2^(n-1)-1보다 커지거나, 음수에서 양수를 빼었을 때 결과값이 -2^(n-1)보다 작아질 때 Overflow가 발생한다.

Overflow가 발생했을 때 감지할 수 있는 중요한 특징은 부호가 변한다는 것이다. 두 양수를 더했을 때 값이 음수가 나오거나, 두 음수를 더했을 때 값이 양수가 나오거나, 양수에서 음수를 빼었을 때 값이 음수가 나오거나, 음수에서 양수를 빼었을 때 값이 양수가 나오는 경우가 그것이다.

따라서, 덧셈이나 뺄셈을 하기 전에 두 수의 부호와 결과값의 부호를 보았을 때, 위의 4가지 경우에 해당된다면 Overflow가 발생했다고 판단할 수 있는 것이다.

(4) RTL Diagram



왼쪽 상단에 Component들이 도식화되었음을 확인할 수 있다. Half Adder로 Full Adder를 구현했고, Full Adder를 8번 ripple-carry하여 Add8를 구현했고, Add8로 Addition을, Addition으로 Comp를 구현했음을 알 수 있다. 계단으로 연결된 Component표시를 통해 Component가 계층적으로 구현되었음을 알 수 있다. 이 Component들은 본 Entity인 LAB5에서 모두 쓰인다. 나머지 부분은 지난 LAB4에서의 RTL Diagram 에서와 같이 If문에서 발생하는 ‘=’ 표시와 MUX21 들이 나열된 모양을 나타내고 있다. Sw(0~17)의 신호로 시작하여 최종적으로 HEX0~7 숫자판과 ledr(0~7)에 Signal들이 할당되고 있음을 볼 수 있다.