Lab4 Final Report

20160410 윤석훈

1. Source code.

Library IEEE;

Use IEEE.std\_logic\_1164.all;

Use IEEE.std\_logic\_unsigned.all;

Entity LAB4 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 LB4 of LAB4 is

signal ANS : 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";

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";

process(sw(17 downto 0))

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(7 downto 0)<= sw(15 downto 8) + sw(7 downto 0);

ANS(7 downto 0)<= sw(15 downto 8) + sw(7 downto 0);

else

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

if(sw(15 downto 8)>=sw(7 downto 0)) Then

ledr(7 downto 0)<= sw(15 downto 8) - sw(7 downto 0);

ANS(7 downto 0)<= sw(15 downto 8) - sw(7 downto 0);

else

ledr(7 downto 0)<= sw(7 downto 0)- sw(15 downto 8) + "10000000";

ANS(7 downto 0) <= sw(7 downto 0)- sw(15 downto 8) + "10000000";

end if;

else

if(sw(7 downto 0)>=sw(15 downto 8)) Then

ledr(7 downto 0)<= sw(7 downto 0) - sw(15 downto 8);

ANS(7 downto 0)<= sw(7 downto 0) - sw(15 downto 8);

else

ledr(7 downto 0)<= sw(15 downto 8)- sw(7 downto 0) + "10000000";

ANS(7 downto 0) <= sw(15 downto 8)- sw(7 downto 0) + "10000000";

End If;

End If;

end if;

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

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

End Process;

End architecture;

(2)Explanation

Entity에서 18개의 스위치(sw[0~17])를 입력으로 선언하였고, 8개의 led(ledr[0~7])와 8개의 7비트벡터 숫자판(HEX0~HEX7)을 출력으로 선언하였다.

Matching함수는 4비트 이진수를 숫자표시로 변환하는데, 4비트 입력 0000(2)~1111(2)를 숫자판의 0,1,…,9,A,b,C,d,E,F 표시로 대응하는 7비트 벡터로 반환한다.

HEX 7,6,5,4는 입력된 스위치 값을 그대로 숫자로 표시하면 되므로 architecture의 처음부터 비트->숫자판 대응을 시키면 된다. 각 숫자판에 Matching함수로 스위치를 4개씩 대응시키면 Hexadecimal 표시로 변환할 수 있다. (HEX2는 사용되지 않으므로 7비트를 “1111111”로 지정했다.)

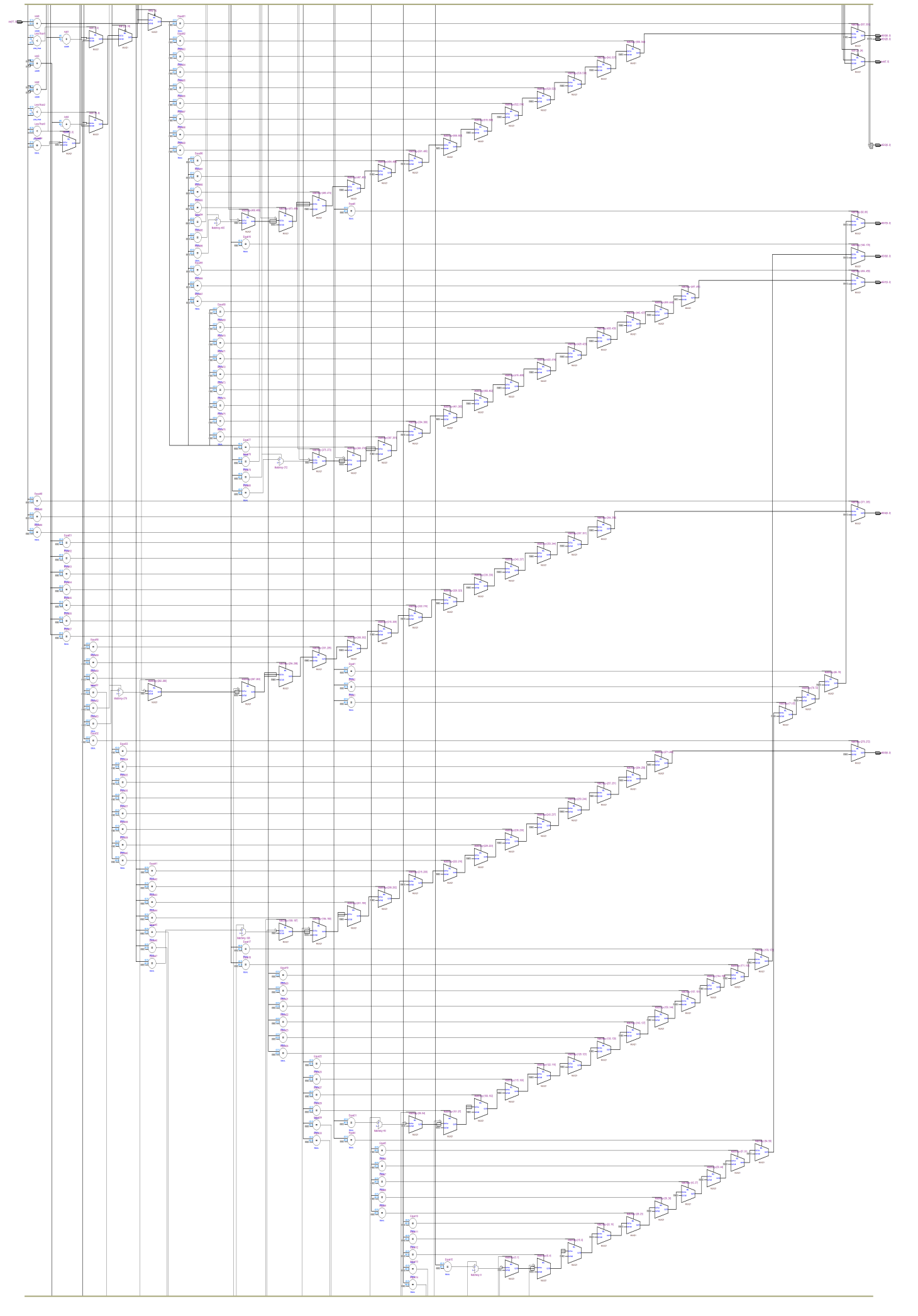
HEX 3은 두 입력 sw[8~15]와 sw[0~7]의 대소를 비교하는 표시를 출력한다. ‘>’ ‘<’ ‘=’ 연산자를 이용해서 각 경우에 전광판 표시 ‘]’, ‘[‘, ‘=’가 출력되도록 HEX3에 알맞은 7비트 숫자판표시 지정한다.

먼저 덧셈의 경우(sw[17]=’0’), 교환법칙이 성립하므로 sw[16]의 값과 상관없이 ledr[7~0]에 단순히 두 8비트 입력을 더한 값을 입력해준다. HEX1과 HEX0에서 결과값을 출력하기 위해서 미리 지정해둔 8비트 시그널 ‘ANS[7~0]’에 그 값을 똑같이 대응시킨다.

뺄셈의 경우(sw[17]=’1’), Operand의 순서를 따져야 하므로, sw[16]=’0’ (A-B)인 경우와 sw[16]=’1’ (B-A)인 경우를 나누어서 뺄셈의 방향을 바꿔준다. 첫 번째 연산자가 두 번째 연산자보다 큰 경우에는 별다른 작업 없이 ledr[7~0]과 ANS[7~0]에 두 8비트 입력을 뺀 값을 대응시킨다. 첫 번째 연산자가 두 번째 연산자보다 작은 경우에는 결과값이 음수이므로 ledr[7~0]이 negative sign 형태로 출력되어야 하는데, 필자는 two’s complement표현을 negative sign 형태로 바꾸는 함수를 따로 선언하지 않고 두 번째 연산자에 첫 번째 연산자의 값을 뺀 후 “10000000”을 더해서 바로 negative sign의 결과를 출력하도록 설계했다.

(3) RTL(Register-transfer Level) Diagram

아래의 그림은 위의 소스코드에서의 변수와 논리연산을 통해 구현되는 디지털 신호에 따라 구현된 회로이다. Matching 함수의 16개의 If-Elsif문에 의한 멀티플렉서(MUX21)들이 많은 부분을 차지한다. HEX3에 저장되는 대소비교(>,=,<) Operator들도 회로의 단계를 구성한다. 위 코드의 대부분이 If-else문으로 구성되어 있으므로, 각 단계마다 입력값에 따라 2가지의 출력 중 골라야 하므로 2-to-1 Multiplexer(MUX21)이 주를 이룬다. 대략적으로 보면, 왼쪽 상단에는 입력인 sw[17~0]으으부터 논리적 분기가 시작되어 출력인 ledr[7~0]과 HEX0~7에서 끝을 맺는다.



(4) Adding and Subtracting without Arithmetic operators.

(1)Adding (A,B)

Process(A,B)

Begin

Ans <=A;

While(B!=”00000000”) loop

Carry <= Ans and B;

Ans <=Ans xor B;

B <= Carry << 1;

End loop

End process

(2)Subtraction (A,B)

Adding(A, Adding(not B, ‘1’))

B+(~B)=111..11(2)=-1(Two’s complement)

🡺(~B)+1=-B

따라서, A-B=A+(-B)=A+((~B)+1)=Adding(A, Adding(~B,1))