Lab6 Final Report

20160410 윤석훈

(1) Source Code

library ieee;  
use ieee.std\_logic\_1164.all;  
use ieee.std\_logic\_unsigned.all;

entity BCDconverter is  
 port(BiNum : IN std\_logic\_vector(8 downto 0);  
 BCD: OUT std\_logic\_vector(11 downto 0));  
   
end entity;

Architecture design of BCDconverter is  
   
 function Adding3(Digit: IN std\_logic\_vector(3 downto 0))   
  return std\_logic\_vector is  
  begin  
  if Digit="0101" then return "1000";  
  elsif Digit="0110" then return "1001";  
  elsif Digit="0111" then return "1010";  
  elsif Digit="1000" then return "1011";  
  elsif Digit="1001" then return "1010";  
  elsif Digit="1010" then return "1101";  
  elsif Digit="1011" then return "1110";  
  elsif Digit="1100" then return "1111";  
  else return "1111";  
  end if;  
 end function Adding3;  
   
 begin  
 process(BiNum)  
 variable InputtoOne,OnetoTen,TentoHun: std\_logic;  
 variable ONE,TEN,HUN : std\_logic\_vector(3 downto 0);  
 variable Input: std\_logic\_vector(8 downto 0);  
 begin  
 ONE:="0000";  
 TEN:="0000";  
 HUN:="0000";  
 Input:=BiNum;  
 for i in 0 to 8 loop  
  if ONE>4 then  
   ONE:=Adding3(ONE);  
  end if;  
    
  if TEN>4 then  
   TEN:=Adding3(TEN);  
  end if;  
    
  if HUN>4 then  
   HUN:=Adding3(HUN);  
  end if;  
    
    
  InputtoOne:=Input(8);  
  OnetoTen:=ONE(3);  
  TentoHun:=TEN(3);  
    
  ONE(3 downto 1) := ONE(2 downto 0);  
  TEN(3 downto 1) := TEN(2 downto 0);  
  HUN(3 downto 1) := HUN(2 downto 0);  
  Input(8 downto 1) := Input(7 downto 0);  
    
  ONE(0):=InputtoOne;  
  TEN(0):=OnetoTen;  
  HUN(0):=TentoHun;  
    
 end loop;  
   
 BCD(3 downto 0)<=ONE;  
 BCD(7 downto 4)<=TEN;  
 BCD(11 downto 8)<=HUN;  
 end process;

end architecture;

library ieee;  
use ieee.std\_logic\_1164.all;  
use ieee.std\_logic\_unsigned.all;

entity lab6 is  
 port(HEX0,HEX1,HEX2,HEX3 : OUT std\_logic\_vector(6 downto 0);  
 sw: IN std\_logic\_vector(8 downto 0));  
   
end lab6;

architecture design of lab6 is  
 signal SignRep: std\_logic\_vector(6 downto 0);  
 signal input: std\_logic\_vector(8 downto 0);  
 component BCDconverter  
 port(BiNum: IN std\_logic\_vector(8 downto 0);  
 BCD: OUT std\_logic\_vector(11 downto 0));  
 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;  
 signal FinalBCD: std\_logic\_vector(11 downto 0);  
 begin  
 process(sw)  
 begin  
  if (sw(8)='1') then  
   SignRep<="0111111";  
   input<=((not sw)+1);  
  else  
   SignRep<="1111111";  
   input<=sw;  
  end if;  
 end process;  
   
 c0: BCDconverter port map(BiNum => input, BCD=>FinalBCD);  
   
 HEX0<=Matching(FinalBCD(3 downto 0));  
 HEX1<=Matching(FinalBCD(7 downto 4));  
 HEX2<=Matching(FinalBCD(11 downto 8));  
 HEX3<=SignRep;  
   
end design;

(2) Explanation

Main Entity에서 9비트 입력이 음수일 때 (sw(8)=’1’)와 양수일 때 (else) BCDconverter Component에 들어갈 Input에 다른 값을 할당한다. 음수일 때는 그것의 절댓값을 나타낸 후 부호만 –로 나타내면 되는데, 절댓값은 ~N+1의 비트연산을 통해 나타나므로 ~sw+1을 Input에 할당한다. 양수의 경우 sw를 그대로 할당한다. 100000000(2) (-256)의 경우 ~N+1을 해서도 100000000(2)이 되지만 위의BCDconverter에서 사용되는 Algorithm을 통해서 256의 BCD표현이 출력되므로 고려하지 않아도 된다.

BCDConverter에서는 BCD표현의 일의 자리, 십의 자리, 백의 자리를 “0000”으로 초기화한 후, Double-Dabble Algorithm를 구현하기 위해 9번의 loop를 설정한다. 각 BCD자리가 5이상일 때 3을 더해주어야 하는데, 12개의 bit를 한번에 조작하기 어려우므로 4비트의 BCD자릿수마다 조작을 가하는 방식으로 진행한다. 3을 더하는 것은 함수로 구현했는데, BCD자릿수가 5~12일 때, 8~15를 반환하는 함수이다.

BCDConverter를 통해 9비트의 Input이 12비트의 BCD Expression으로 바뀌면 4비트씩 숫자판에 Matching하고, 양수 및 음수 부호를 HEX3에 표시해주는 것으로 마무리한다.

(3) Algorithm Used

BCDConverter에는 Double-Dabble Method (Shift-and-add3 algorithm)이 사용되었다. Binary N bit입력에 대해 4\*Ceil(N/3) bit의 BCD bit를 모두 0으로 할당하고, 그 오른쪽에 입력된 N bit를 놓는다. N번 left-shift하는데, 각 shift마다 BCD의 자릿수 4비트의 십진수 값이 4보다 크면 3씩 더해준다. N번의 과정이 끝나면 BCD bit에는 Binary N bit 입력의 BCD표현 값이 할당된다.