CS211 LAB3 Final Report

20160410 윤석훈

(1) Source Code

library IEEE;

use IEEE.std\_logic\_1164.all;

Entity LAB3 is

port(

sw: in std\_logic\_vector (2 downto 0);

ledr: out std\_logic\_vector (2 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 LB3 of LAB3 is

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";

end if;

END Matching;

begin

ledr<=sw;

Process(sw)

begin

if(sw="000")Then

HEX7 <= Matching("0010");

HEX6 <= Matching("0000");

HEX5 <= Matching("0001");

HEX4 <= Matching("0110");

HEX3 <= Matching("0000");

HEX2 <= Matching("0100");

HEX1 <= Matching("0001");

HEX0 <= Matching("0000");

elsif(sw="001")Then

HEX0 <= Matching("0010");

HEX7 <= Matching("0000");

HEX6 <= Matching("0001");

HEX5 <= Matching("0110");

HEX4 <= Matching("0000");

HEX3 <= Matching("0100");

HEX2 <= Matching("0001");

HEX1 <= Matching("0000");

elsif(sw="010")Then

HEX1 <= Matching("0010");

HEX0 <= Matching("0000");

HEX7 <= Matching("0001");

HEX6 <= Matching("0110");

HEX5 <= Matching("0000");

HEX4 <= Matching("0100");

HEX3 <= Matching("0001");

HEX2 <= Matching("0000");

elsif(sw="011")Then

HEX2 <= Matching("0010");

HEX1 <= Matching("0000");

HEX0 <= Matching("0001");

HEX7 <= Matching("0110");

HEX6 <= Matching("0000");

HEX5 <= Matching("0100");

HEX4 <= Matching("0001");

HEX3 <= Matching("0000");

elsif(sw="100")Then

HEX3 <= Matching("0010");

HEX2 <= Matching("0000");

HEX1 <= Matching("0001");

HEX0 <= Matching("0110");

HEX7 <= Matching("0000");

HEX6 <= Matching("0100");

HEX5 <= Matching("0001");

HEX4 <= Matching("0000");

elsif(sw="101")Then

HEX4 <= Matching("0010");

HEX3 <= Matching("0000");

HEX2 <= Matching("0001");

HEX1 <= Matching("0110");

HEX0 <= Matching("0000");

HEX7 <= Matching("0100");

HEX6 <= Matching("0001");

HEX5 <= Matching("0000");

elsif(sw="110")Then

HEX5 <= Matching("0010");

HEX4 <= Matching("0000");

HEX3 <= Matching("0001");

HEX2 <= Matching("0110");

HEX1 <= Matching("0000");

HEX0 <= Matching("0100");

HEX7 <= Matching("0001");

HEX6 <= Matching("0000");

elsif(sw="111")Then

HEX6 <= Matching("0010");

HEX5 <= Matching("0000");

HEX4 <= Matching("0001");

HEX3 <= Matching("0110");

HEX2 <= Matching("0000");

HEX1 <= Matching("0100");

HEX0 <= Matching("0001");

HEX7 <= Matching("0000");

end if;

end process;

END architecture;

(2)Explanation

전체 엔티티인 'LAB3'에서는 입력으로 FPGA의 3개의 스위치인 SW[0]~SW[2]를 선언했고, 출력으로 FPGA의 ledr[0]~ledr[2]와 HEX0~HEX7(8개의 숫자판)을 선언했다. LAB3의 아키텍쳐를 begin하기 전에 Matching이라는 함수를 선언했는데, 이것은 bit로 표현된 0~15의 숫자를 HEX 전광판에 대응하는 7bit 값에 대응시키기 위한 것이다. 그러므로, 4비트 벡터를 입력으로 받고, 7bit 벡터를 리턴하는 함수이다. 아키텍쳐에서는 스위치(sw)의 값에 따라 8개의 숫자판 (HEX0~7)에 본인의 학번 '2''0''1''6''0''4''1''0'을 대응시킨 것의 출력값을 shift하도록 구성했는데, 이를 위해서 Process(sw)를 사용했다. Process안에서 Sequential하게 sw=000일 때부터 sw=111일 때까지 총 8가지 경우를 나누었고, 각 경우마다 HEX0~7에 8자리 학번을 적절히 shift해서 대응시켰다. 이때, Matching함수를 사용하여 학번의 숫자가 전광판의 숫자로 출력되도록 하였다. sw가 turn on 되었을 때, 그에 대응하는 ledr이 켜지도록 ledr<=sw를 추가하였다.

(3) Function의 장점

Function을 사용하면 자료나 값을 대응시키는 작업을 반복할 때, 하나의 함수를 선언하고 여러 번 사용할 수 있어 더 깔끔하고 효율적인 코드 구성이 가능하다. 만약에 위 코드에서 Matching이라는 함수가 없었다면 2,0,1,6,0,4,1,0이라는 값을 넣을 때마다 각 숫자가 전광판에 표시될 때 구성해야 할 7bit가 어떤 것일지 계속 계산했어야 할 것이다. 또한 이럴 경우 코드를 보고서 어떤 작업을 하는 것인지 보는 이가 파악하기 무척 어려울 것이다. 적절한 함수 이름을 통해 어떤 작업을 하려는 것인지 이해하기 쉽게 되는 것도 함수의 중요한 장점 중 하나일 것이다. 이것들은 Component의 장점과도 통하는 부분이다.

(4) Component와 비교했을 때 function의 단점

Component의 경우, 대응되는 출력 변수의 개수에 따라 한번에 여러 개의 출력을 받아올 수 있다. 하지만 Function은 한 개의 리턴 값만을 받으므로 여러 개의 출력이 필요한 기능을 구현하려면 여러 개의 함수를 구현하고 실행해야 할 것이다. 또한, Component는 본 함수의 Architecture 밖에서 따로 선언되므로 Component만의 회로도를 RTL diagram에서 확인할 수 있다. 하지만 function은 Architecture 내부에서 선언된 것으로, 새로운 hardware를 형성하지 않고, 무슨 기능을 구현하는지 시각적으로 볼 수 없다.

\*참고자료

https://stackoverflow.com/questions/47866635/difference-between-function-and-component-in-vhdl