LDLAB 7

수리과학부 2019-16022 박채연

1. Counter Module

'counter.v' 파일에 Behavioral Description 방법을 이용하여 0부터 99까지 1씩 증가하는 모듈을 아래와 같이 구현하였다.

```
module counter(
    input clr,
    input clk,
    output [3:0] MSB,
    output [3:0] LSB
    );

        reg [3:0]msb_rnm0 =0;
        reg [3:0]lsb_rnm0 =0;

        assign MSB = msb_rnm0;
        assign LSB = lsb_rnm0;

        always @(posedge clk or posedge clr)
            begin
                if (clr) begin
                    lsb_rnm0 = 4'b0000;
                    msb_rnm0 = 4'b0000;
                end
                else begin
                    lsb_rnm0 = lsb_rnm0+1;
                    if (lsb_rnm0 == 4'b1010) begin
                        lsb_rnm0 = 4'b0000;
                        msb_rnm0 = msb_rnm0 + 1;
                    end
                    if (msb_rnm0 == 4'b1010) begin
                        msb_rnm0 = 4'b0000;
                    end
                end
            end

endmodule
```

counter 모듈의 output은 각각 4비트로 이루어진 MSB와 LSB이다. 이들은 각각 십의 자리수와 일의 자리 수를 이진수로 나타낸다. 베릴로그에서는 나눗셈의 몫과 나머지를 구하는 연산이 제공되지만 SNU Logic Design Board에서 해당 연산을 지원하지 않기 때문에 각 자리별로 하나의 output을 사용해야 한다. 여기에서는 0에서 99까지의 counter를 구현하면 되므로 총 두 개의 output을 사용하였다. 또한 각 output이 4비트로 구성되는 이유는 각 자리는 0에서 9까지의 총 10개의 상태를 표현할 수 있어야 하기 때문이다.

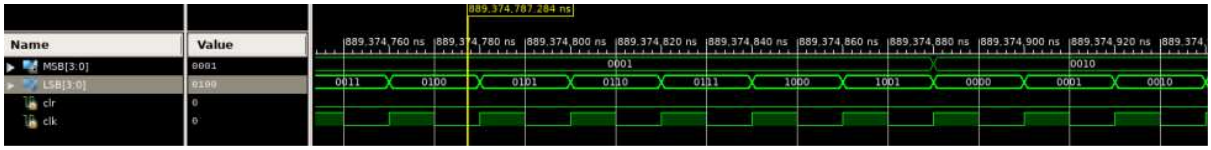
counter 모듈의 input은 clr과 clk인데, clr은 clear의 약자로서 clr가 1이 되면 MSB와 LSB를 모두 0으로 초기화해야 한다. clk는 clock의 약자로서 clk이 0에서 1로 바뀌는 순간 counter를 1씩 증가시켜야 한다.

always@(posedge clk or posedge clr) 문법을 통해 clk이 0에서 1로 바뀌거나 clr이 0에서 1로 바뀔 때 always문 내부가 실행되도록 하였다. 만약 clr = 1이면 MSB와 LSB를 각각 0으로 초기화하고 그렇지 않으면 LSB 값을 1씩 증가시킨다. 한편, 만약 LSB가 10이면 이를 0으로 바꾸어주고

MSB 값을 1만큼 증가시킨다. 이때 만약 MSB 값이 10이 되면 MSB 값 역시 0으로 바꾸어준다.

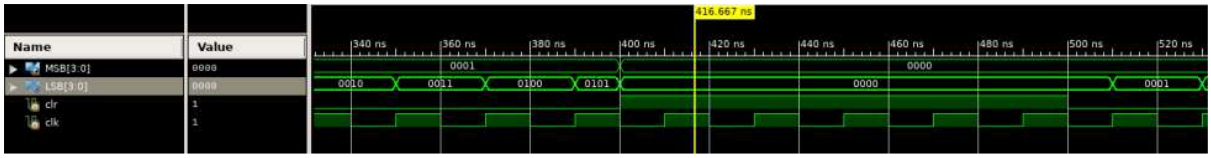
테스트 벤치 코드를 작성하여 해당 모듈을 시뮬레이션한 결과는 아래와 같다. 일반적인 경우에 1씩 올바르게 증가하며 99 다음 0으로 정상적으로 초기화되는 것을 확인 가능하다. 또한 만약 reset(=clr) = 1일 때 0으로 초기화되는 모습을 확인 가능하다.

- 일반적인 경우에 1씩 증가



- counter = 100일 때 0으로 초기화

- reset(=clr) = 1인 경우 0으로 초기화



2. Two Digit counter

앞에서 구현한 counter 모듈과 강의자료에 제시된 freq divider 모듈, BCD-7 seg decoder 모듈을 이용하여 Two Digit counter 모듈을 구현하였다.

우선, 'freq_divider.v'와 'bcd_to_7.v' 파일에 Behavioral Description 방법을 이용하여 freq divider 모듈과 BCD-7 seg decoder를 각각 아래와 같이 구현하였다.

```
module freq_divider(
    input clr,
    input clk,
    output reg clkout
    );
            reg [31:0] cnt;
            always @(posedge clk)
                    begin
                            if (clr) begin
                                    cnt<=32'd0;
                                    clkout <=1'b0;
                            end
                            else if (cnt ==32'd25000000) begin
                                    cnt <=32'd0;
                                    clkout <= ~clkout;
                            end
                            else begin
                                    cnt<=cnt+1;
                            end
                    end

endmodule
```

freq_divider 모듈은 50MHz 클럭을 이용하여 1Hz 클럭을 만드는 모듈이다. 해당 모듈의 input에는 clr과 clk가 있는데, clk는 50MHz인 클럭이고, clr은 초기화 신호이다. clk가 0에서 1로 바뀔 때 clr = 1이면 cnt 값과 clkout 값을 모두 0으로 초기화한다. 그렇지 않으면 cnt 값을 1만큼 증가시킨다. 한편, 50MHz 클럭이 5천만 번 oscillate하면 1초가 흐른다. 즉, 2천 5백만 번 oscillate하면 0.5초가 흐른다. 따라서 2천 5백만 번 oscillate했을 때 clkout을 0에서 1로 증가시키고, 다시 2천 5백만 번 oscillate하면 clkout을 1에서 0으로 감소시키는 것을 반복하면 clkout은 1Hz 클럭의 역할을 수행하게 된다.

```
module bcd_to_7(
    input [3:0] bcd,
    output reg[6:0] seg
    );
            always @(bcd) begin
                    case(bcd)
                            4'd0 : seg <=7'b0111111;
                            4'd1 : seg <=7'b0000110;
                            4'd2 : seg <=7'b1011011;
                            4'd3 : seg <=7'b1001111;
                            4'd4 : seg <=7'b1100110;
                            4'd5 : seg <=7'b1101101;
                            4'd6 : seg <=7'b1111101;
                            4'd7 : seg <=7'b0000111;
                            4'd8 : seg <=7'b1111111;
                            4'd9 : seg <=7'b1101111;
                    endcase
            end

endmodule
```

또한, 위와 같이 bcd_to_7 모듈을 구현하였는데, input은 bcd이고 output은 seg이다. 4비트 이진수로 표현되어 있는 bcd에 대응되는 7 segment display 정보를 seg에 저장하는 역할을 수행한다.

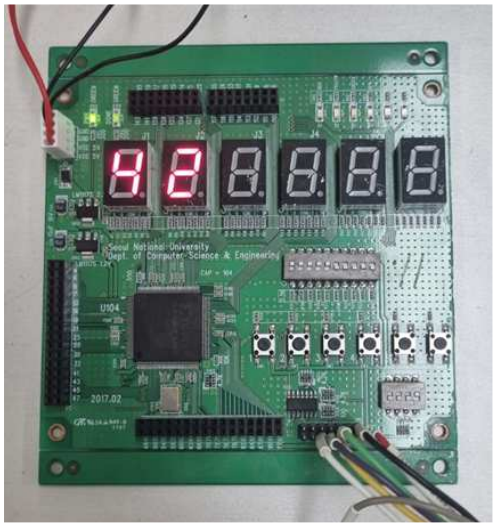
마지막으로, 'TwoDigit.v' 파일에 Two Digit Counter 메인 모듈에 해당하는 TwoDigit 모듈을 아래와 같이 구현하였다.

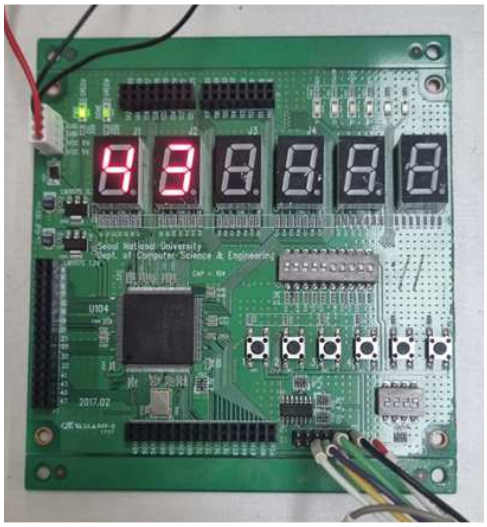
```
module TwoDigit(
    input clr,
    input clk,
    output [6:0] high,
    output [6:0] low
    );

            wire clkout;
            wire [3:0] hi;
            wire [3:0] lo;

            freq_divider Freq_module(.clr(clr), .clk(clk), .clkout(clkout));
            counter Counter_module(.clr(clr), .clk(clkout), .MSB(hi), .LSB(lo));
            bcd_to_7 MSB_module(.bcd(hi), .seg(high));
            bcd_to_7 LSB_module(.bcd(lo), .seg(low));

endmodule
```

앞에서 구현한 freq_divider 모듈을 이용하여 clkout이 1Hz 클럭의 역할을 수행하도록 만들고 counter에 이를 전달함으로써 1초에 1씩 증가시키도록 하였다. counter는 십의 자리수와 일의 자리수에 해당하는 숫자를 4비트 이진수로 wire인 hi와 lo에 각각 나타내는데, 이를 다시 두 개의 bcd_to_7 모듈에 각각 전달함으로써 7 segment display를 표현하는 7비트 이진수로 바꾸었다.

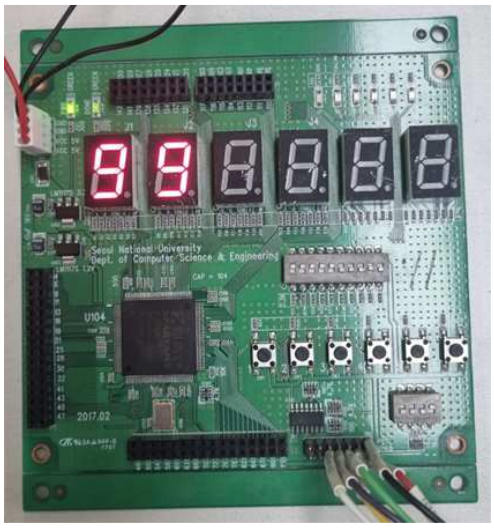
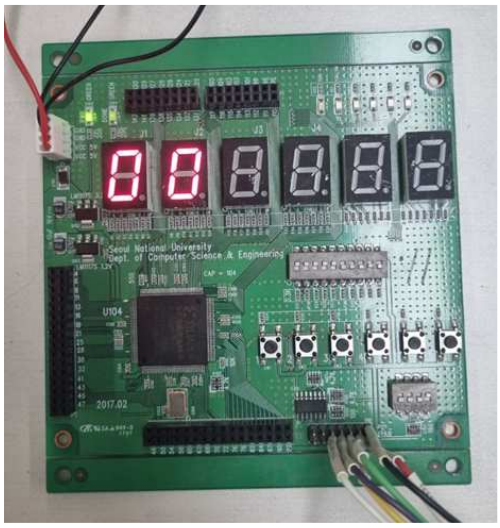
마지막으로, TwoDigit 모듈의 output에 해당하는 high와 low를 각각 SNU Design Board의 7 segment display에 연결하고, TwoDigit 모듈의 input에 해당하는 clr과 clk를 각각 pin 47에 해당하는 tactile switch와 pin 57에 해당하는 50MHz oscillator에 연결하였다. SNU Logic Design Board에서 구현 결과를 확인하면 아래와 같이 정상적으로 동작하는 것을 확인 가능하다.

- 일반적인 경우에 1씩 증가

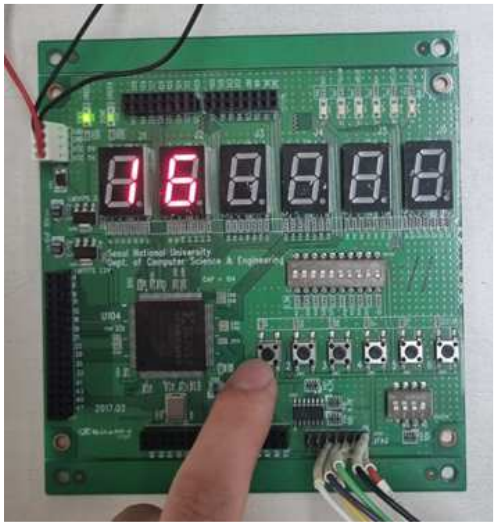
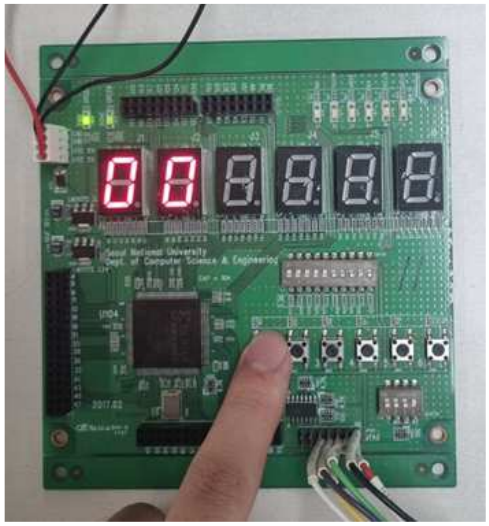




- t = 100s일 때 0으로 초기화

- reset 버튼을 눌렀을 때 0으로 초기화

3. Discussion

(1) Concepts that I learned during the lab session

ppt에 제공되어있는 'freq_divider'모듈을 통해 50MHz 클럭을 이용하여 1Hz 클럭을 만드는 등 빠르게 oscillate하는 클럭을 이용하여 더 느리게 oscillate하는 클럭을 만들 수 있다는 것을 배웠다. 50MHz 클럭은 파동이 1초에 5천만번 지속되며, 1Hz 클럭은 파동이 1초에 1번 지속된다. 따라서 50MHz 클럭이 1번 oscillate할 때마다 count를 증가시키다가 count가 2천 5백만이 되면 이를 0으로 초기화하고 clkout을 0에서 1로 바꾼다. 다시 count가 2천 5백만이 되면 이를 0으로 초기화하고 clkout을 1에서 0으로 바꾸는 과정을 반복한다. 이로써 1Hz의 역할을 수행하는 클럭인 clkout을 만들 수 있는 것이다.

또한, 구현 방법에 따라 counter에서 clear가 asynchronous 혹은 synchronous 방식으로 다르게

동작한다는 것을 배웠다. Behavioral Description 방법으로 1초에 1씩 증가시키는 counter 모듈을 구현할 때, always@(posedge clk) 내부에 clr = 1 인지 여부를 확인하는 if 문이 있다면 clk가 0에서 1로 바뀔 때만 clear가 1인지 여부를 확인하므로 clear 버튼이 synchronous하게 동작하게 된다. 즉, clear 버튼이 1이어도 clk가 0에서 1로 바뀌지 않는다면 counter가 초기화되지 않는다. 반면, 앞에서의 always문을 always@(posedge clk or posedge clr)로 바꾸어준다면 clr이 0에서 1로 바뀔 때에도 always문 내부의 코드가 실행되므로 clear가 asynchronous하게 동작하게 됨을 배웠다.

(2) Any errors that I made

Two Digit Counter의 메인 모듈에 해당하는 TwoDigit 모듈에서 7 segment display에 대한 비트 정보를 나타내는 output을 SNU Logic Design Board로 연결하는 과정에서 pin 번호를 잘못 매핑하여 숫자가 잘못 출력되는 문제가 발생하였다.

counter 모듈을 구현하고나서 시뮬레이션이 정상적으로 동작하는 것을 확인하고 이를 이용하여 TwoDigit 메인 모듈을 구현하였다. 그러나 clear 역할을 수행하는 tactile 스위치를 누를 경우 counter가 0으로 초기화되는 것이 아니라 정지되는 문제를 겪었다. counter 모듈 내부의 always@(posedge clk)를 always@(posedge clk or posedge clr)로 수정하고나서 SNU Logic Design Board에 I/O pin mapping을 수행하기 위해 'I/O Plan Planning (PlanAhead) – Post-Synthesis'을 클릭하니 'known FF or Latch ~'와 같은 에러메시지가 출력되었다. 조교님께서도 논리적으로 올바르게 작성된 코드인데 해당 에러가 발생하는 이유를 알기 어렵다고 하셨다.

(3) How to correct my errors

counter 모듈의 always@() 문 내부에서 if-else 문 뒤에 if 문 두 개를 연달아 사용하였는데, 가장 처음에 나오는 if 문의 조건이 true인 경우 논리적으로 else 문 뒤에 있는 if 문 두 개는 실행될 일이 없었다. 따라서 else 문 뒤에 나오는 두 개의 if 문을 else 문 내부로 옮겨주었다. 즉, if문 + else문 + if 문 두 개의 구조를 if문 + else {if 문 두 개}와 같은 구조로 바꾸어주었다. 그 결과 더 이상 에러메시지가 출력되지 않고 'I/O Plan Planning (PlanAhead) -Post-Synthesis'가 정상적으로 수행되는 것을 확인하였다. 또한, Two Digit Counter에서 7 segment display의 pin 번호를 제대로 다시 mapping 해줌으로써 숫자가 잘못 출력되는 문제를 해결하였다.