디지털시스템설계 프로젝트 보고서

FPGA를 활용한 Zero game의 구현

2022. 06. 14
20210054 정하우
20210716 최대현
20210907 차은성

1. 개요

한 학기동안 실습한 내용을 바탕으로 제로게임을 구현한다. 본 프로젝트를 통해 제로게임을 구현하는 FSM을 설계하고 FPGA 프로그램인 Vivado에 구현하고 실제 부품을 사용해 설계한 FSM을 실제로 회로를 제작해 Sequential logic를 포함해 한 학기 동안 배운 내용을 실제로 적용시켜보며 내용을 깊이 있게 이해하고자 한다.

2. 이론적 배경

1) Positive Logic / Negative Logic

논리식을 실제 전자 회로로 구현할 때 불 대수의 참과 거짓을 High (높은 전압)와 Low (낮은 전압)에 임의로 대응할 수 있다. 예를 들어 Positive logic (Active high)의 경우 참을 High, 거짓을 Low로 표현하고, Negative logic (Active-low)의 경우 참을 Low, 거짓을 High로 표현한다. 이 둘은 논리 회로를 전기적으로 표현하는 방법만 다를 뿐 기능적인 차이는 없다. 단, 실제 부품을 사용하여 회로를 구성할 때는 Active-high와 Active-low 부품이 섞여 있을 수 있으므로 주의해야 한다.

본 프로젝트에서 사용한 BCD-to-seven-segment display는 positive logic임을 실험적으로 확인하였다.

2) HDL (Hardware Description Language)

Hardware Description Language는 디지털 시스템의 논리 회로 구조를 표현하기 위한 언어다. 주로 사용되는 언어로는 Verilog와 VHDL이 있으며, 이번 디지털 시스템 설계 프로젝트

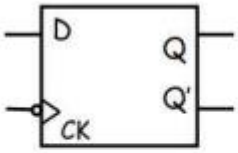
에서는 Verilog를 사용한다. System Verilog와는 다른 언어로 지원하는 기능에 차이가 있으므로 주의가 필요하다.

3) 플립플랍(flip flop)

Flip-flop은 이전상태를 계속 유지하여 1 비트의 정보를 저장, 보관, 유지할 수 있는 회로이다. Latch와 같이 Sequential logic circuits이며, Master/Slave Flip-Flop, Edge-triggered J-K Flip-Flop, T(toggle) Flip-Flop, D(delay) Flip-Flop 등이 있다.

4) D 플립플랍(D(Delay) flip flop)

Flip-Flop의 한 종류이며, 입력 D의 값을 clock의 edge에서 Q에 반영한다. edge가 발생하지 않는 시간에는 Q가 변하지 않고 유지한다. J에 D를, K에 not D를 연결한 J-K Flip-Flop와 동일하다. Truth table은 다음과 같다.



D Q	Q+
0 0	0
0 1	0
1 0	1
1 1	1

그림 1. D 플립플롭의 회로와 Excitation table

본 프로젝트에서는 게임 시작전에 clock를 low level로 설정해 player에게 받은 입력값이 BCD-to-seven-segment decoder에 반영되지 않도록 설계하였다. 그리고 모든 player에게 입력값을 받은 후 게임을 시작할 때 clock를 high level로 설정해 D FF에서 값이 출력되어서 게임이 실행되도록 설계하였다.

5) 가산기(adder)

가산기는 두 수를 더해 출력한다. 1-bit 숫자를 더하는 가산기는 반가산기(half adder), 전가산기(full adder)가 있다. 반가산기는 1-bit 이진수 두 개를 입력받아 합과 Carry (올림) out을 출력한다. 전가산기는 1-bit 이진수 두 개와 이전 가산기의 Carry in을 입력받아 합과 Carry out을 출력한다. 두 개의 반가산기를 사용해 구현할 수 있다.

그리고 1-bit 가산기를 이용해 N-bit 가산기를 구현할 수 있다. N-bit 가산기는 다양한 방법으로 구현할 수 있는데, 그중 하나로 N-bit 리플 가산기가 있다. N-bit 리플 가산기는 N개의 전가산기를 순차적으로 이어 구현한다. N개의 전가 산기는 각각 N-bit 중 한 자릿수의 연산을 맡는다. 이때 k번째 자릿수를 계산하는 전가산기의 Carry out이 k+1번째 자릿수를 담당하는 전가산기의 Carry in에 연결되어 가장 낮은 자릿수부터 가장 높은 자릿수까지 Carry가 순차적으로 전파된다. 이러한 순차적인 연결 구조로 인해 낮은 자릿수의 연산이 끝나 Carry out이 결정되어야 높은 자릿수의 연산을 시작할 수 있다. 즉, 연산하고자 하는 자릿수가 늘어남에 따라 연산에 걸리는 시간도 비례해서 늘어난다. 한편, A-B 꼴의 뺄셈은 A+(-B) 꼴의 덧셈으로 나타낼 수 있다. 이러한 성질과 2의 보수(2's complement)를 활용하면 N-bit 가산기를 사용해 감산기를 간단히 구현할 수 있다.

본 프로젝트에서는 반가산기(half adder)를 이용해 전가산기(full adder)를 만들고, 전가산기를 이용해 3 비트 가산기(3 bit adder)를 module화 시켜 player1, player2, player3에서 입력받은 값만큼 더하는 알고리즘을 구현하였다.

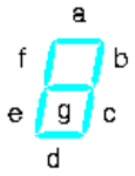
6) 디코더(decoder)

디코더는 n개의 이진 입력을 받아 최대 2^n개의 고유 출력을 가지는 회로다. n개의 이진 입력과 2^n개의 서로 다른 출력을 가지는 경우, 각 출력이 곧 minterm이 기 때문에 minterm generator라고도 한다. 실제 디코더 소자에는 n개의 입력뿐 아니라 EN, 혹은 Enable 입력이 추가로 존재한다. EN은 말 그대로 디코더를 켜거 나 끄기 위해서 사용된다. 디코더의 입력과 출력 특성을 표현할 때는 n-to-2^n과 k-of-2^n라는 표현을 사용한다. n-to-2^n은 n개의 입력을 받아 2^n개의 출력을 가진다는 입력과 출력의 관계를 표현하고, k-of-2^n은 2^n개의 입력 중

에서 k개의 입력이 동시에 참이 된다는 출 력의 특성을 나타낸다.

7) BCD-to-seven-segment decoder

디지털 또는 이진 디코더는 디지털 코드의 한 형태를 다른 형태로 변환할 수 있는 디지털 조합 논리 회로이다. BCD to 7-segment 디스플레이 디코더는 이진 코드화된 소수점을 7-segment 디스플레이를 통해 쉽게 표시할 수 있는 다른 형태로 변환할 수 있는 특수 디코더이다. 숫자를 나타내는 방법에 따라 회로가 다르다. 아래는 BCD-to-seven-segment decoder의 한 예시이다.



(a) Segment designation

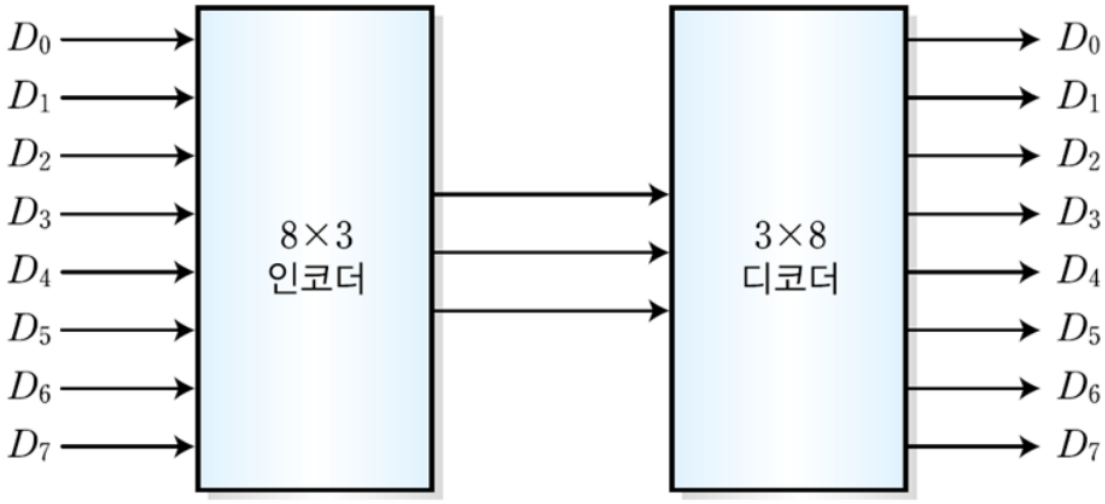
(b) Numeric designation for display

BCD Input				Seven-Segment Decoder						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
All other inputs				X	X	X	X	X	X	X

8) 인코더(encoder)

앞서 배운 디코더(decoder)와 다르게 2^n개의 입력을 받아 n개의 고유 출력을 가지는 회롱다.

본 프로젝트에서는 8개의 입력을 받아 3bit로 출력을 표현하는 8x3 encoder를 사용하였다.



본 프로젝트에서는 각 player에서 받은 입력값을 더한 값의 최대값은 6이므로, 출력값이 111(7)인 경우는 무시하였다. Truth table은 다음과 같다.

I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0

9) 멀티플렉서(multiplexer)

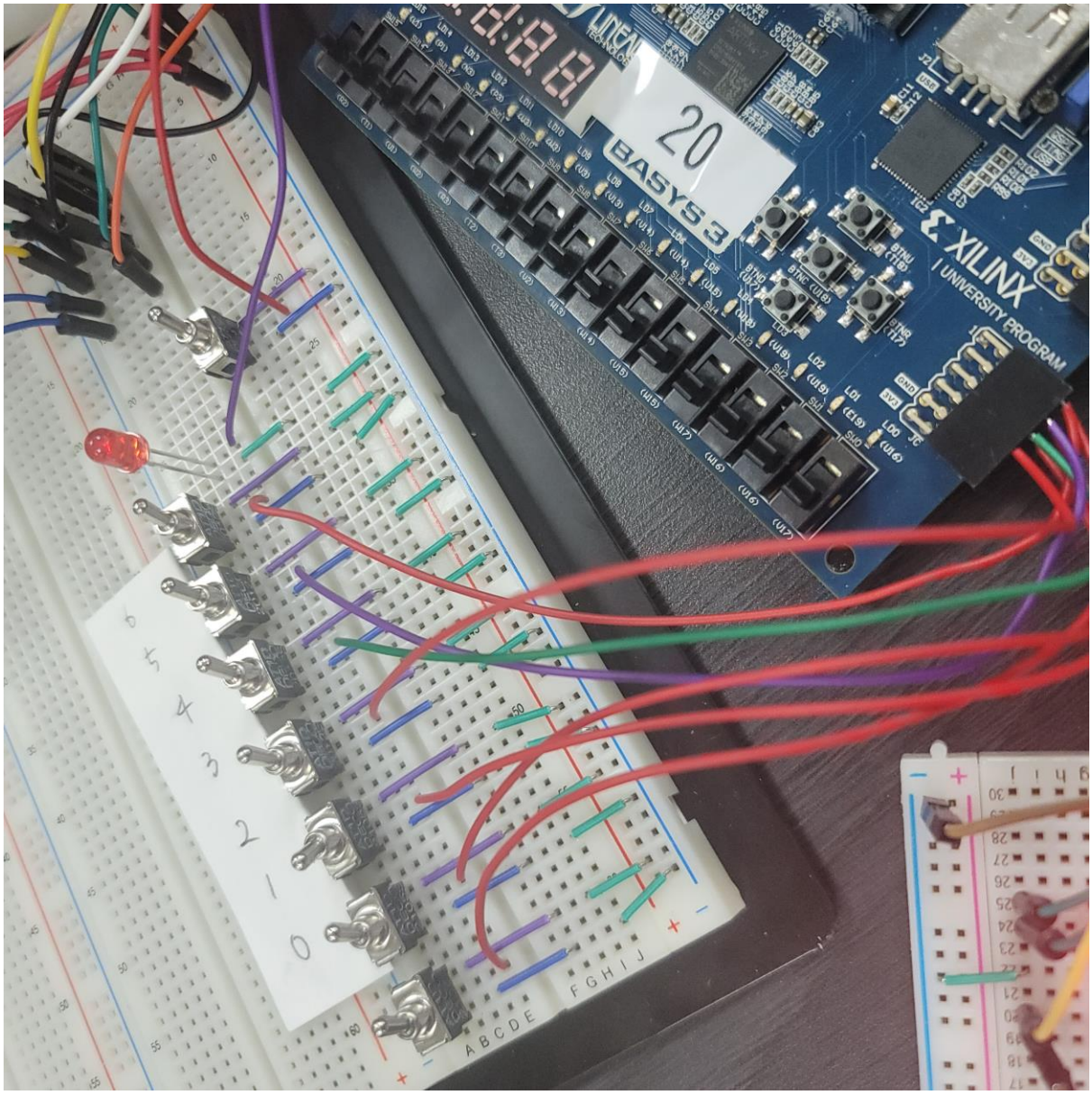
2^n개의 입력신호 중 n개의 선택선 조합에 의해 하나만 출력하는 회로이다. mux라고도 불린다. n개의 선택선은 각각 0 또는 1을 가지므로 총 2^n경우의 수를 가지므로 각 경우에 따라 하나의 입력값을 선택해 출력할 수 있다. 본 실험에서는 8x1 mux를 사용하여서 8bit짤 mux를 만들었다.

8bit mux는 8개의 8bit 입력 값 중 3개의 선택선 조합에 의해 하나의 8bit 입력값을 출력하는 회로이다. 이는 각 bit 자리수에 8x1 mux를 연결시켜 만들었다.

3. Verilog를 통한 FPGA 설계

1) encoder 설계

ㄱ. 하드웨어 설계



Toggle 스위치를 통해서 8bit 입력값을 받는다. 입력값을 받을 경우에는 한 스위치만 high상태로

둔다.

ㄴ. 코드 작성

1) select module 설계

Selector module은 zero game에서 자신의 턴인 플레이어가 손가락의 총 합을 예측하는 동작을 구현한 module이다. Encoder을 사용해서 입력을 받은 뒤, 이를 7 segment에 출력할 수 있는 형태의 8 bit output으로 내보낸다.

module encoder

```
module encoder(
    input wire [7:0]in,
    output [2:0]out
    );

    or(out[2],in[4],in[5],in[6]);
    or(out[1],in[2],in[3],in[6]);
    or(out[0],in[1],in[3],in[5]);
```

Encoder를 설계하면 다음과 같다. K-map를 그려서 결과를 얻을 수 있지만, 이 경우는 단순하기 때문에 과정을 생략하였다.

module selector

```
module selector(
    input wire [7:0]in,
    input wire clk,
    output [2:0]out
    );

    wire [2:0] enc_out;
    encoder sel(in, enc_out);
    edge_trigger_D_FF ff0(1, enc_out[0], clk, out[0], ~out[0]);
    edge_trigger_D_FF ff1(1, enc_out[1], clk, out[1], ~out[1]);
    edge_trigger_D_FF ff2(1, enc_out[2], clk, out[2], ~out[2]);

endmodule
```

Encoder를 이용해서 selector module을 설계하였다. 본 module은 ㄱ.과정에서 입력받은 8bit 값을 통해 목표값의 3bit binary 배열을 출력한다. D FF를 이용해 게임이 시작하기 전에는 BCD-to-seven-segment decoder에 반영되지 않도록 설계하였다.

```
selector sel(sel_in, clk, sel_out);
```

Main module인 zerogame module에서 사용하였다. Sel_in은 ㄱ.에서 받은 8bit 값이고, sel_out는 목표값의 3bit binary 배열이다.

2) player module 설계

Player module은 3명의 player에게 받은 값을 adder를 통해 더해서 얻은 값을 BCD-to-seven-segment decoder에 표현할 수 있도록 mux에 선택선 조합으로 입력될 3bit 값을 출력하는 module이다.

module halfAdder

```
module halfAdder(
    input in_a,
    input in_b,
    output out_s,
    output out_c
    );

    xor(out_s,in_a,in_b);
    and(out_c, in_a, in_b);

endmodule
```

module fullAdder

```
module fullAdder(
    input in_a,
    input in_b,
    input in_c,
    output out_s,
    output out_c
    );

    halfAdder PPA(in_a, in_b, S, C1);
    halfAdder PPB(in_c, S, out_s, C2);
    or(out_c, C2, C1);

endmodule
```

module bit2adder (3bit adder)

```
module bit2adder( // 3비트 애더
    input wire [2:0] n1,
    input wire [2:0] n2,
    output wire [2:0] n
);


fullAdder FA1(n1[0], n2[0], 0, n[0], cin_1);
fullAdder FA2(n1[1], n2[1], cin_1, n[1], cin_2);
fullAdder FA3(n1[2], n2[2], cin_2, n[2], cin_3);



endmodule
```

Fulladder 모듈 3개를 이어 붙인 ripple adder 형태의 3bit binary adder이다.

module player

```
module player(
    input wire [1:0] player11,
    input wire [1:0] player22,
    input wire [1:0] player33,
    input wire clk,
    output wire [2:0] out
    );

    wire [2:0] player1;
    wire [2:0] player2;
    wire [2:0] player3;
    wire [2:0] temp;
    wire [2:0] temp1;
    wire [2:0] temp2;
    wire [2:0] temp3;

    edge_trigger_D_FF(1,player11[0],clk,player1[0],~player1[0]);
    edge_trigger_D_FF(1,player11[1],clk,player1[1],~player1[1]);
    edge_trigger_D_FF(1,player22[0],clk,player2[0],~player2[0]);
    edge_trigger_D_FF(1,player22[1],clk,player2[1],~player2[1]);
    edge_trigger_D_FF(1,player33[0],clk,player3[0],~player3[0]);
    edge_trigger_D_FF(1,player33[1],clk,player3[1],~player3[1]);

    fullAdder(player1[0],player1[1], 0, temp1[0], temp1[1]);
    fullAdder(player2[0],player2[1], 0, temp2[0], temp2[1]);
    fullAdder(player3[0],player3[1], 0, temp3[0], temp3[1]);

   bit2adder BA1(temp1, temp2, temp);
   bit2adder BA2(temp, temp3, out);




endmodule
```

Player module 역시 입력값을 D FF으로 연결해 clock이 high level일 때 값이 반영되도록 설계하였다. 각 player에서 받은 2bit 값의 각 자릿수를 1bit fulladder를 통해 더하고 이를 통해 나온 세

값을 3bit adder를 통해 더하고 더한 값을 출력하였다.

3) D FF 설계

module edge_trigger_JKFF (JK flipflop 설계)

```
module edge_trigger_JKFF(input reset_n, input j, input k, input clk, output reg q, output reg q_);

    initial begin
      q = 0;
      q_ = ~q;
    end //초기화

    always @(negedge clk) begin
        q = reset_n & (j&~q | ~k&q);
        q_ = ~reset_n | ~q;
    end

endmodule
```

JK flip flop을 기반으로 D flipflop을 설계했다.

module edge_trigger_D_FF (D flipflop 설계)

```
module edge_trigger_D_FF(input reset_n, input d, input clk, output q, output q_);
    edge_trigger_JKFF temp(reset_n,d,~d,clk,q,q_);
endmodule
```

4) module compare 설계

Compare module은 각 player에서 입력받은 값의 합과 목표값을 비교하는 module이다. 값이 같으면 1을 출력하고, 다르면 0을 출력한다. 출력값은 lcd에 연결되어 값이 같은 경우 lcd에 불이 켜지도록 설계하였다. 코드는 다음과 같다.

```
module compare(
    input wire [2:0]sel,
    input wire [2:0]sum,
    output same
    );

    xnor(n2, sel[2],sum[2]);
    xnor(n1, sel[1],sum[1]);
    xnor(n0, sel[0],sum[0]);
    and(same, n2, n1, n0);

endmodule
```

== operator을 사용하지 못하기 때문에 각 자릿수를 모두 비교하여 자릿수가 모두 같을 때에만 1을 리턴했다. xnor을 통해 각 자릿수가 같을 경우에만 1이 나오도록 설계하였고, 모든 자리 숫자가 같은 경우에만 1을 출력하기 위해서 and gate를 사용하였다. 이 output이 led에 연결되어 두 수가 같을 때, 즉 현재 turn의 플레이어가 이긴 경우에만 led에 불빛이 들어온다.

5) 8 bit 8x1 mux 설계

1 bit 8x1 mux 설계

```
module mux(
    input wire [7:0] data_input,
    input wire [2:0] select_input,
    output wire out
    );

    wire i0, i1, i2, i3, i4, i5, i6, i7;// i7
    and(i0, select_input[2], select_input[1], select_input[0], data_input[0]);
    and(i1, select_input[2], select_input[1],  ~select_input[0], data_input[1]);
    and(i2, select_input[2],  ~select_input[1], select_input[0], data_input[2]);
    and(i3, select_input[2],  ~select_input[1],  ~select_input[0], data_input[3]);
    and(i4,  ~select_input[2], select_input[1], select_input[0], data_input[4]);
    and(i5,  ~select_input[2], select_input[1],  ~select_input[0], data_input[5]);
    and(i6,  ~select_input[2],  ~select_input[1], select_input[0], data_input[6]);
    and(i7,  ~select_input[2],  ~select_input[1],  ~select_input[0], data_input[7]);
    or(out, i0, i1, i2, i3, i4, i5, i6, i7);//,i7

endmodule
```

회로의 전선 연결 순서를 거꾸로 해서 실제로는 data_input[0]이 7번 주소의 데이터를 가리키는

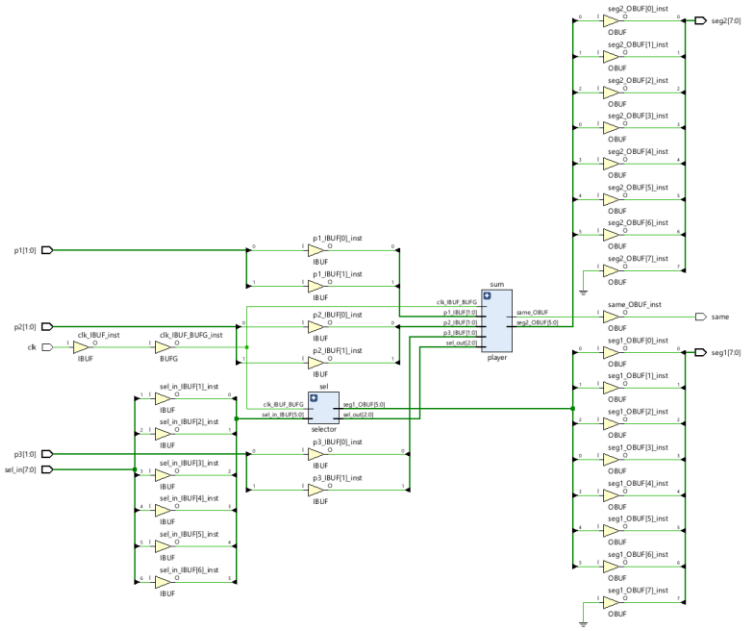
것을 시작으로, data_input[7]까지 순차적으로 감소한다. 따라서 i0의 output이 실제로는 7, i0이 실제로는 0의 output을 연결해주는 mux이다.

8 bit 8x1 mux 설계

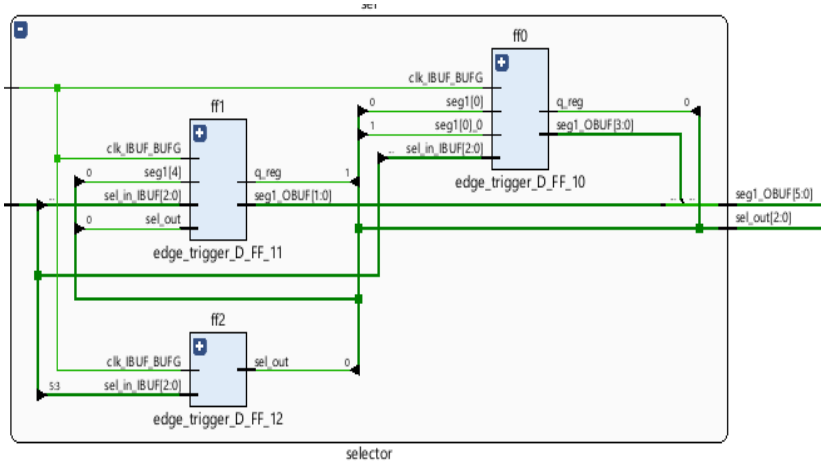
```
module mux_8_1(
    input wire[2:0] select,
    output wire[7:0] out
);

    wire [7:0] a0=8'b10110110; //A
    wire [7:0] a1=8'b11111000; //B
    wire [7:0] a2=8'b11011110; //C
    wire [7:0] a3=8'b10110110; //D
    wire [7:0] a4=8'b10100010; //E
    wire [7:0] a5=8'b10001110; //F
    wire [7:0] a6=8'b00111110; //G
    wire [7:0] a7=8'b00000000; //DOT

    mux(a0,select,out[0]);
    mux(a1,select,out[1]);
    mux(a2,select,out[2]);
    mux(a3,select,out[3]);
    mux(a4,select,out[4]);
    mux(a5,select,out[5]);
    mux(a6,select,out[6]);
    mux(a7,select,out[7]);
```

8 bit 8x1 mux를 통해 선택된 값을 BCD-to-seven-segment display에 표현하기 위해, BCD-to-seven-segment display에 입력할 8bit 값을 출력하게 하였다. 기본적인 mux에 따르면 8개의 input가 있어야 하지만, 본 프로젝트에서는 출력값이 BCD-to-seven-segment display에 입력할 값이므로 입력값을 고정하기 위해 따로 입력값을 받지 않고 mux 내부에 8bit 값 8개를 설정하고 각 자리 수에 1 bit 8x1 mux 8개를 입력시켜 1개의 8 bit 값이 출력되도록 하였다.

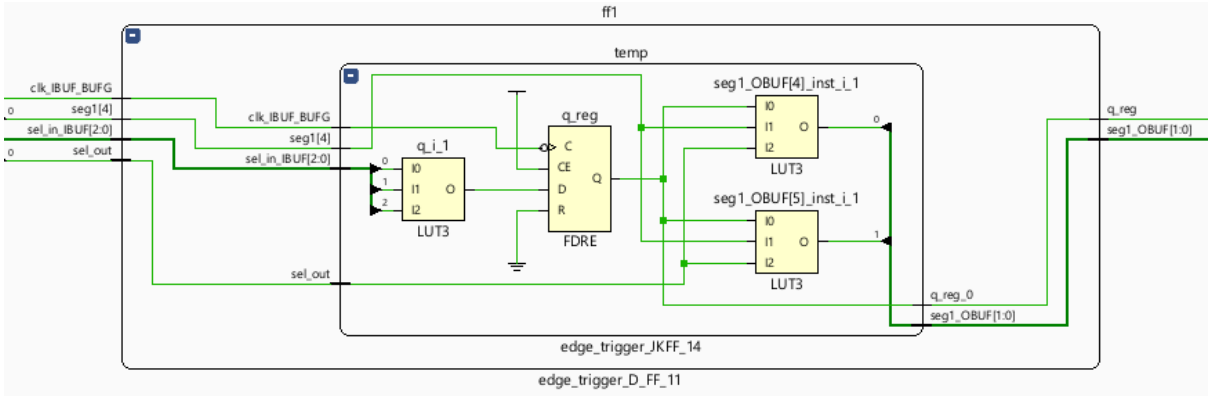
ㄴ. Verilog의 schematic 기능을 활용한 회로도



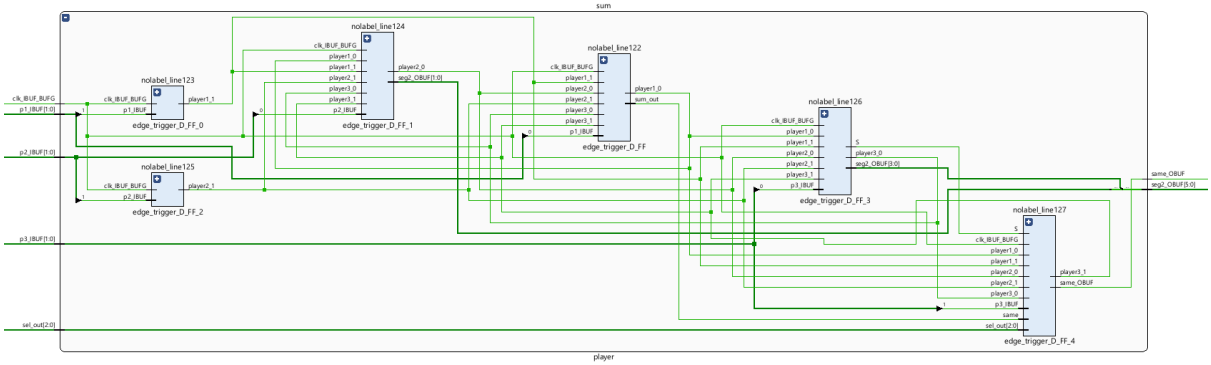
<전체 회로에 대한 schematic>



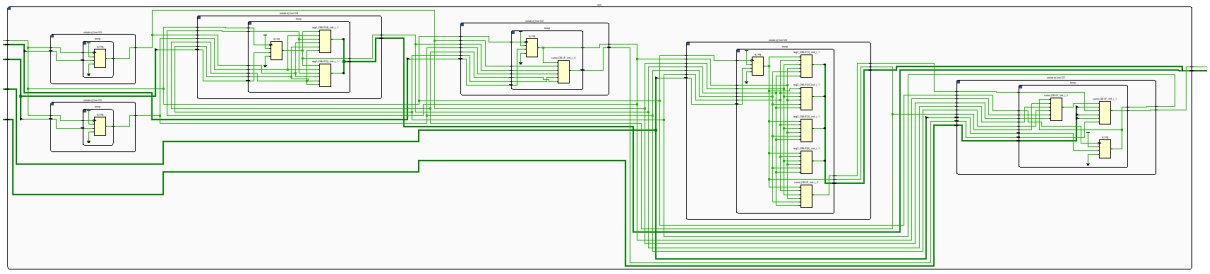
<Selector module 에 대한 schematic>



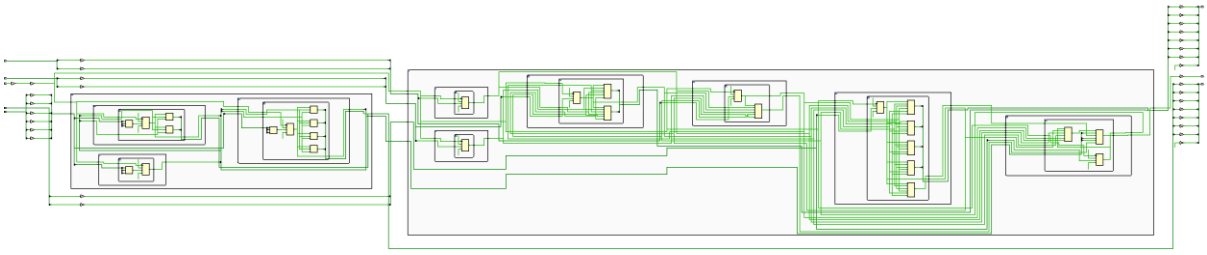
<Selector module 안의 D FF schematic>



<player module 의 schematic>



<D FF 내부를 보여주는 player module 의 schematic>

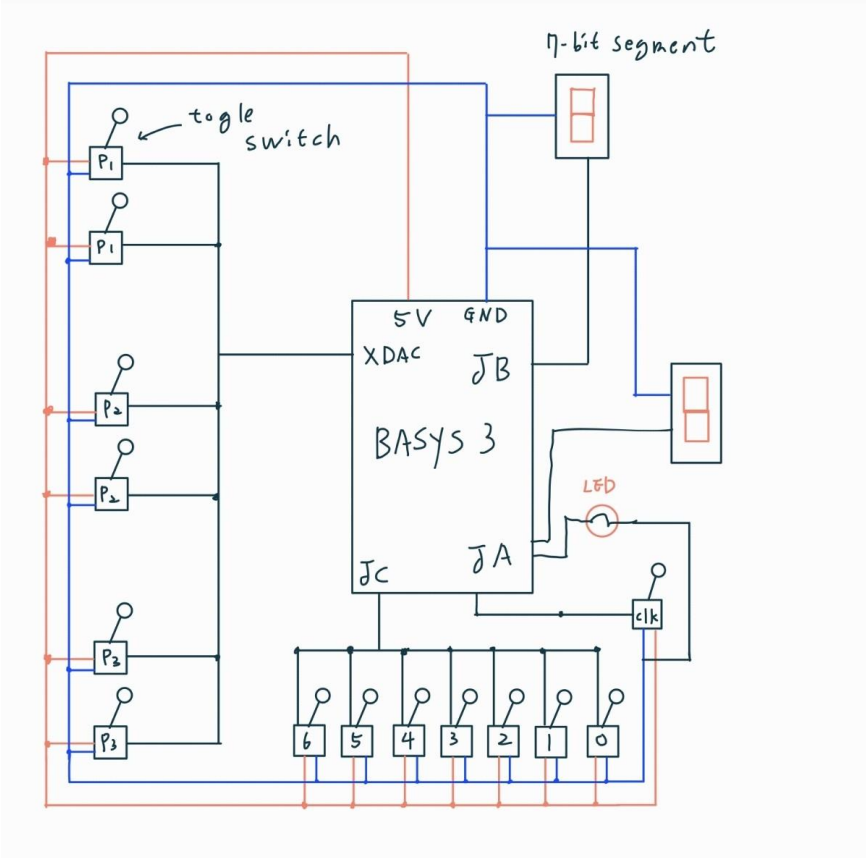


<D FF 내부를 보여주는 전체 회로의 schematic>

4. 브레드보드를 이용한 하드웨어 설계

ㄱ. 회로 구현 및 설명

Zero game 구현을 위한 대략적인 회로도 구성은 다음과 같다.



(1). 게임을 플레이하는 player 3명은 각각에게 주어진 toggle switch 1쌍(좌측)을 이용하여 매 턴 마다 0에서 2사이의 숫자를 선택한다. 예를 들어 switch 2개를 누르면 2, 둘 중 하나를 누르면 1, 한 개도 누르지 않으면 0이 된다.

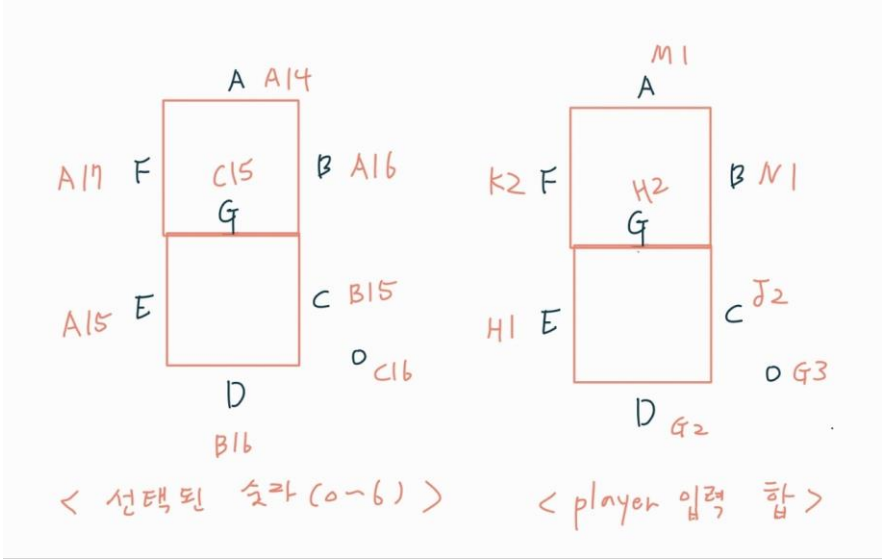
(2). 각 플레이어는 자신이 차례가 되면, 0~6사이에 숫자를 정한다. 만약 그 숫자가 (1)에서 플레이어들이 선택한 숫자들의 합과 같으면 승리한다. 숫자를 정할 차례가 된 플레이어는 7개의 toggle switch (하단) 중 자신이 원하는 숫자에 해당하는 스위치를 누른다.

(3). D-FF의 클럭 역할을 수행하는 toggle switch(우측) 을 두 번 딸깍이면 (1)과 (2)에서 플레이어들이 선택한 값들이 BASYS3에 입력 값으로 들어온다. BASYS3 내부에서는 (1)에서 각 플레이어들이 선택한 숫자 3개를 합하고(add), 그 합과 (2)에서 선택한 숫자를 비교한다 (compare).

(4). BASYS3에 연결된 두 개의 7-bit segment는 각각 player들의 입력 값을 표시하며 (하나는 (1)에서 선택한 숫자 3개의 합, 하나는 (2)에서 선택한 숫자) 만약 두 값이 같다면, LED에 전압을 가해 빛나게 한다.

(5). 본 프로젝트에서 사용된 toggle switch들은 BASYS3의 Pmod XDAC, JC에, 클럭용 switch는 JA에 input으로 연결되었으며, 7-bit segment는 JA, JB에 Led는 JA에 output으로 연결되었다.

추가적으로, 7-bit segment의 각 led 요소에 연결된 BASYS3의 port는 다음과 같다.

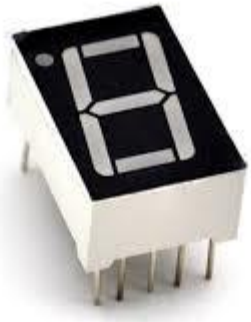


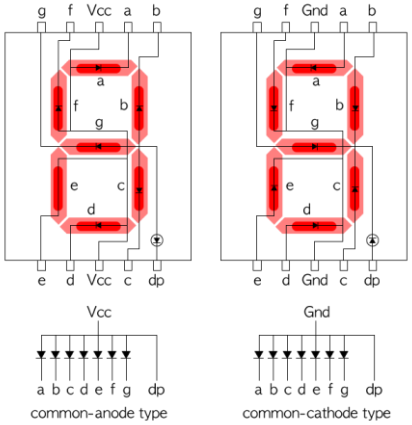
ㄴ. 부품 설명

1. Toggle switch

사용자의 입력에 따라 0 또는 1을 출력한다. 양 끝의 핀은 각각 Vcc, GND에 연결하고 중간의 핀은 스위치가 켜질 때만 전류를 흘려보낸다. BASYS3에 중앙 핀을 연결하여 스위치가 켜지면 1, 아니면 0을 입력 값으로 받을 수 있다.

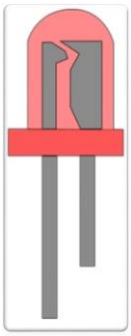
2. 7-bit segment

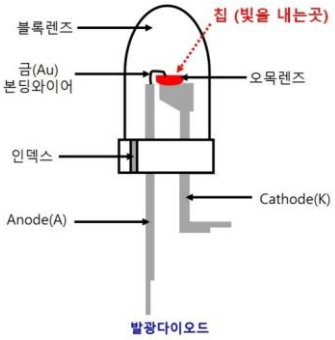




총 10핀으로 구성되어 있으며 그 중 8핀은 세그먼트를 구성하는 8개의 내부 led를 제어하는데 사용되고, 나머지 2핀은 Vcc, 또는 Gnd에 연결된다. 본 프로젝트에서는 cathode type을 사용하기 때문에 Gnd에 연결시켰다. 8개 핀에 전압을 어떻게 가하는지에 따라 다양한 숫자를 나타낼 수 있다.

3. Led

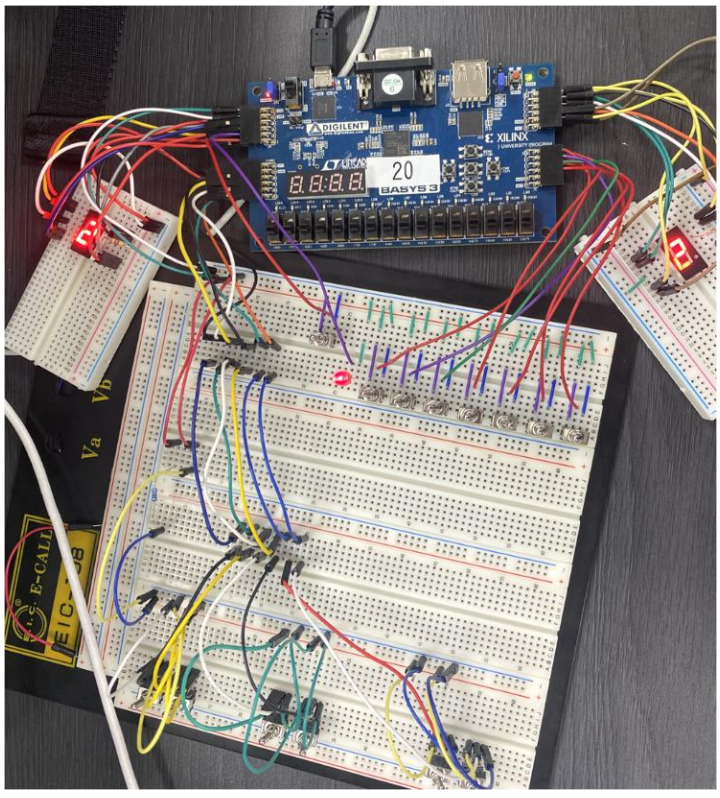




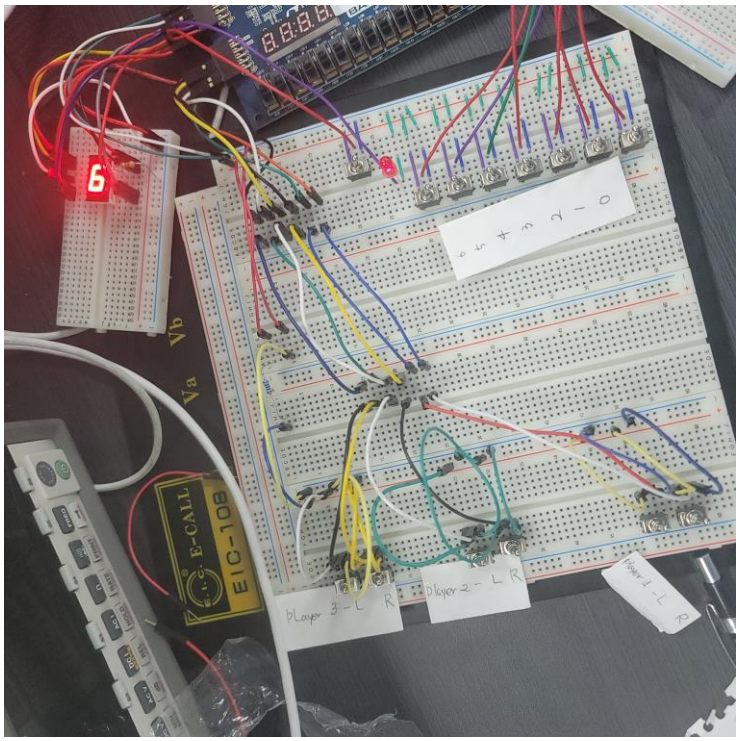
Led(발광 다이오드)는 순방향으로 전압을 걸어주었을 때 빛을 내는 반도체 소자이다. 두 개의 다리 중 긴 쪽(anode)에서 짧은 쪽(Cathode) 방향으로 전압을 걸어주면 발광한다. 높은 전압을 받았을 때, 쉽게 타버리기 때문에 보통 전압을 낮춰주는 저항과 함께 사용한다.

ㄷ. 회로 동작 설명

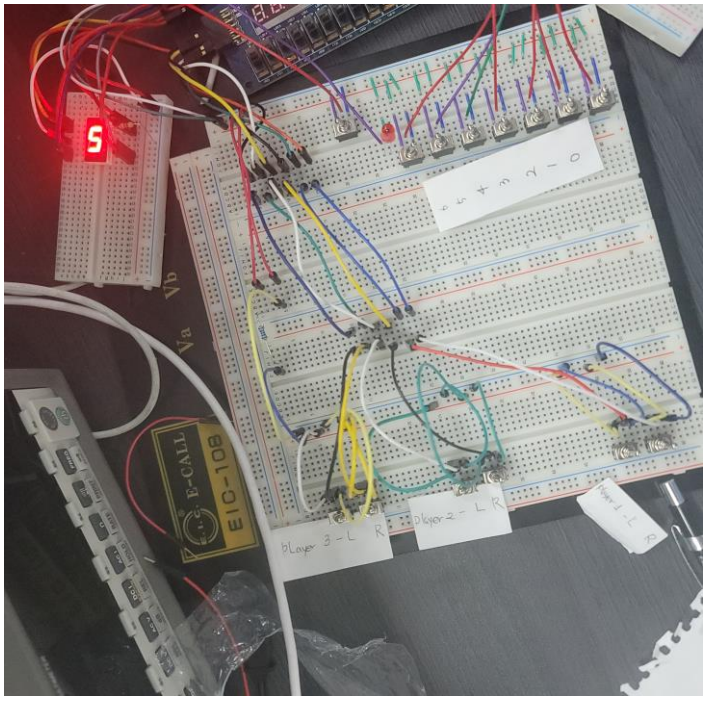
브레드보드를 사용한 전반적인 회로의 구성은 다음과 같다.



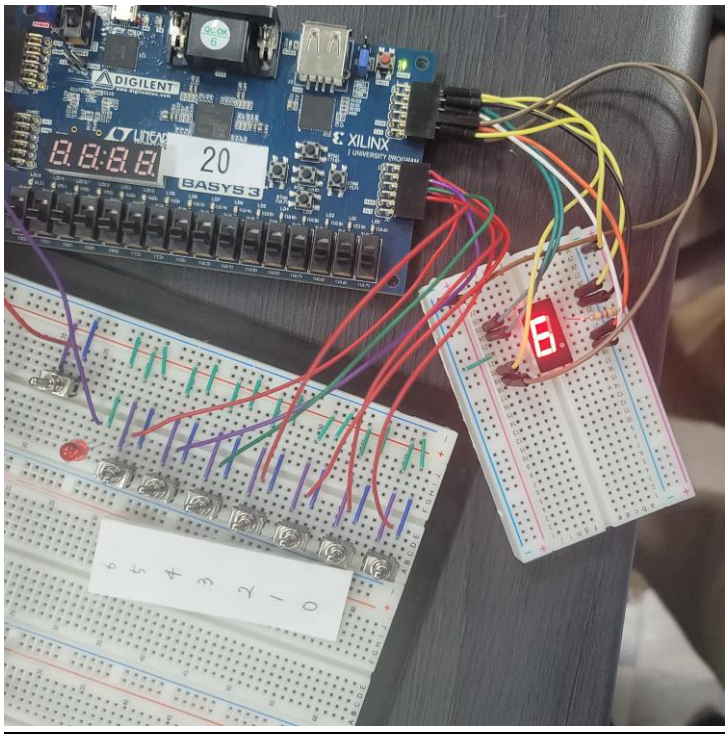
다음으로, 몇 가지 동작의 예시를 들어 보면 다음과 같다.



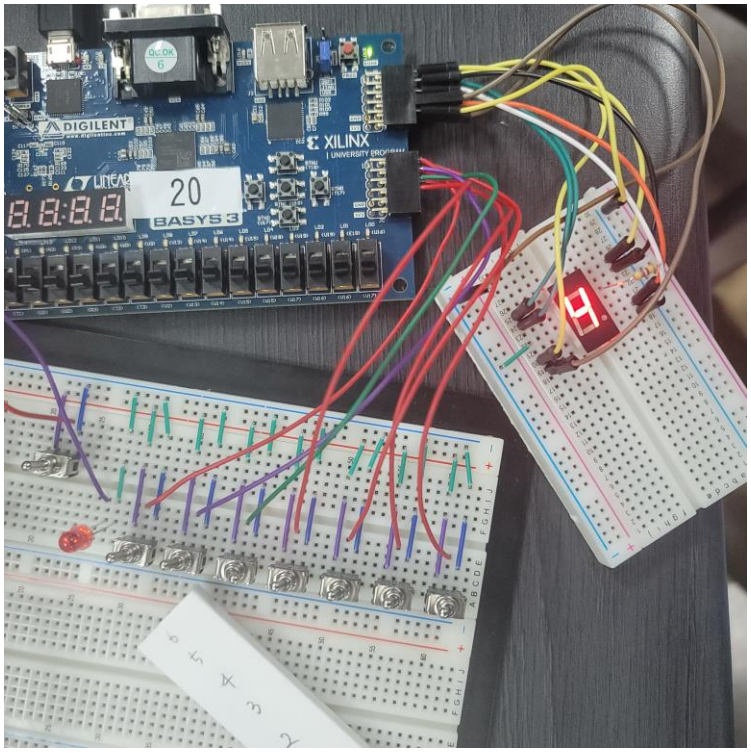
왼쪽에 7 segment는 각 손가락의 총 합이다. 현재 clock은 내려가 있고, 원래 6개의 스위치가 닫혀 있는 상태에서 스위치 한 개를 열었을 때는 clock이 내려가 있으므로 현재 상태가 유지된다.



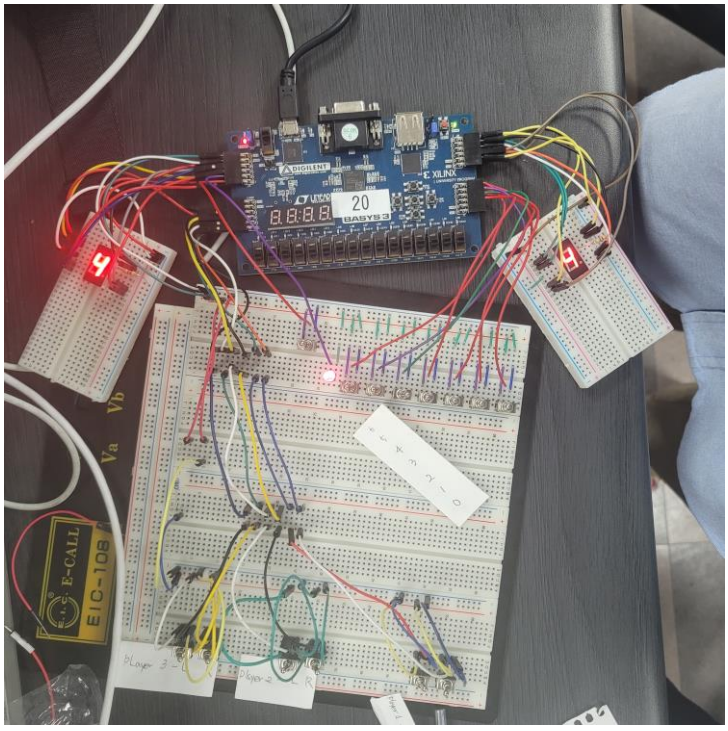
이 상태에서 clock을 올려주면 d flip flop이 작동하면서 7 segment에 손가락 수의 변화가 반영된 5가 출력된다.



다음으로, 숫자를 select 하는 selector module 에 연결된 switch 중 '6' switch 를 닫고 나머지를 열었을 때, 우측의 7 segment 에는 6 이 찍힌다.



다음으로 clock을 내린 상태에서 selector switch를 4로 바꾼 후 clock을 올리면 우측의 segment에 4가 출력된다.



이 상태에서 clock을 내리고 손가락을 총 4개 올린 뒤 clock을 다시 올리면 좌측 segment에 4가 출력되고, 두 segment에 같은 수가 입력되므로, 즉 같은 state이므로 compare module의

output이 1로 바뀌어 led에 불이 들어온다.

5. 결론

하드웨어와 소프트웨어 간의 적절한 분배를 통해 우리가 원한 형태의 zero 게임을 FPGA 실습을 통해 올바르게 구현해볼 수 있었다. 설계 당시와 약간 달라진 부분은 설계 당시에는 각 player에 개별적인 clock을 물린 뒤 그 clock값이 모두 high일 때 d flip flop이 작동되게 했는데, 제작을 하다 보니 게임의 자연스러운 구동을 위해 clock을 한 번에 물리는 것이 낫다고 생각해서 clock을 하나로 줄였다는 점이다. Adder와 encoder, compare 모듈 등 사용해야 하는 부품을 FPGA로 구현했는데, 다음에는 기회가 된다면 이런 모듈들을 하드웨어 부품으로 일부 구현해보고 싶다. 또한, 스피커가 반조립 제품이어서 납땜을 해야 하는 문제로 사용하지 못해서 이 부분도 보완해보고 싶다.