Lotto6/45 GAME

Digital System Design Final Project

PBK

20220100 박기현

20220119 배준희

20220449 김정우

1. 개요

본 프로젝트에서 제작하고자 하는 Lotto6/45 GAME 은 순서와 값을 정확히 맞춘 로또 번호 개수에 따라 1 등부터 7 등까지의 등수를 출력해주는 장치이다. 사용자의 입력에 따라 크게 두 가지의 state, 즉 로또 번호 설정 state 와 로또 번호 맞추기 state 로 넘어간다. FPGA 의 스위치를 통해 입력 받는 6 개의 숫자는 각각 00 부터 99 까지의 두 자리 숫자이며, 이때 입력 받은 숫자의 십의 자리와 일의 자리 BCD 각각을 4-Bit 2 진수로 변환하여 처리한다. 맞춘 개수에 따라 출력되는 등수는 FPGA 의 7-segment LED 를 통해 확인할 수 있다. 디지털시스템설계 과목을 수강하면서 배운 내용을 토대로 직접 FSM 을 설계하고 Verilog 와 FPGA 를 통해 Lotto6/45 GAME 을 구현한다.

2. 이론적 배경

1) Karnaugh map

K-map 이라고도 불리는 Karnaugh map 은 불 대수식을 단순화(simplification)하는 방법 중 하나이다. 입력 변수와 출력값을 도식화하여 1 이 나오는 경우, 즉 최소항(minterm)만을 찾아 그 때의 입력값을 식으로 표현하는 방법이다.

이때 주의해야 할 점은 2 의 n 제곱 개수로 묶어주어야 하며, 사각형의 형태로 묶어주어야 한다. 또한, 묶을 수 있는 경우가 다양하게 존재한다면, 그중 가장 많은 1 을 포함하는 경우를 선택하여 단순화를 진행한다. don't care 항은 경우에 따라 포함될 수도, 포함되지 않을 수도 있다.

2) 2-Bit Magnitude Comparator

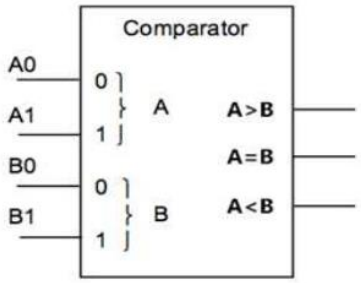


그림 1 – 2-Bit Magnitude Comparator

2-Bit Magnitude Comparator 는 서로 다른 두 개의 2-Bit 수가 입력으로 주어졌을 때, 둘의 대소 관계를 판별하여 대소 관계에 따른 알맞은 출력을 1 로, 나머지의 출력은 0 으로 설정하여 출력해주는 회로이다. 그림 1 과 같이 2-Bit 수 2 개가 주어지고, 입력 A1 과 B1 이 더 높은 자릿수라고 가정할 경우, 출력 A>B 는 0, 출력 A=B 는 1, 출력 A<B 는 0 이 된다.

본 프로젝트에서는 'A=B'의 관계만 필요하기에 출력 A>B 와 출력 A<B 는 생략한다. 또한, 입력 받은 BCD 를 4-bit 2 진수로 변환하여 사용하므로, 4-bit Magnitude Comparator 로 수정하여 구현한다. 따라서 세부적으로는 XOR 게이트를 활용하여 각 자릿수의 동일 관계만 고려한 후, 그에 맞는 결과에 따라 A=B 일 경우 1 의 출력을, A≠B 일 경우 0 의 출력을 수행한다.

3) Decoder

Decoder(디코더)는 n 개의 이진 정보 입력에서 최대 2^n 개의 고유 출력으로 변환하는 다중 입력, 다중 출력 회로이다. n 개의 이진 입력과 2^n 개의 서로 다른 출력을 가지는 경우, 각 출력이 최소항을 의미하기 때문에 최소항 생성기(minterm generator)라고도 부른다. 따라서 디코더로 임의의 함수를 구현할 수 있는데, 함수에 대한 최소항의 합이 있는 경우 디코더와 OR 게이트를 활용하여 구현할 수 있다.

디코더 회로에는 EN(Enable input)이 존재하여 디코더를 활성화하거나 비활성화할 수 있다.

본 프로젝트에서는 7-segment decoder 를 활용한다. 입력 받는 4-Bit 값에 따라 7-segment 출력에 필요한 0~9, t, n, d, r, h, E, 총 16 개의 출력으로 변환한다. 7-segment decoder 를 구현하기 위해 K-map 을 이용하여 단순화를 진행하고, 단순화된 불 대수식을 이용하여 회로를 설계한다.

4) SR Latch

SR Latch 에서 S 는 Set 을, R 을 Reset 을 의미한다. Latch 의 출력값인 Q 와 Q'는 서로 complement 관계이므로, Q = 1 이면 Q' = 0, Q = 0 이면 Q' = 1 이 된다.

NOR 게이트로 구성된 SR Latch 는 S = 1, R = 0 또는 S = 0, R = 1 일 때 출력값이 변화하며, S = 0, R = 0 일 때 이전 상태를 유지한다. S = 1, R = 1 이 입력되면 출력값이 모두 0 이 되므로 Q 와 Q′가 complement 관계라는 전제에 모순이 발생한다. 따라서 forbidden, 즉 오류이다.

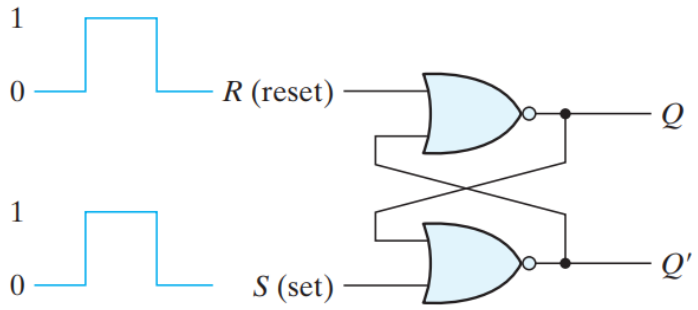


그림 2 – SR Latch with NOR gates

Function table 로 나타내면 다음과 같다.

S	R	Q	Q′	
1	0	1	0	
0	0	1	0	After S = 1, R = 0
0	1	0	1	
0	0	0	1	After S = 0, R = 1
1	1	0	0	forbidden

NAND 게이트로 구성된 SR Latch 는 S = 0, R = 1 또는 S = 1, R = 0 일 때 출력값이 변화하며, S = 1, R = 1 일 때 이전 상태를 유지한다. S = 0, R = 0 이 입력되면 출력값이 모두 1 이 되므로 Q 와 Q′가 complement 관계라는 전제에 모순이 발생한다. 따라서 forbidden, 즉 오류이다.

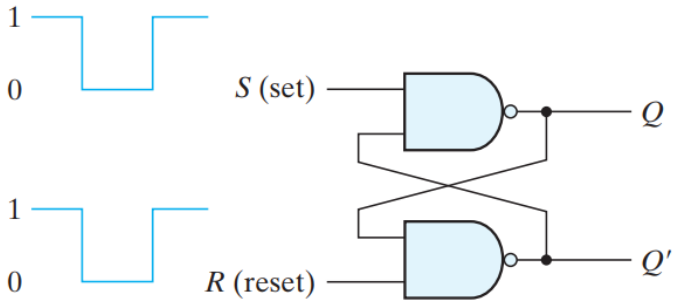


그림 3 – SR Latch with NAND gates

Function table 로 나타내면 다음과 같다.

S	R	Q	Q′	
1	0	0	1	

1	1	0	1	After S = 1, R = 0
0	1	1	0	
1	1	1	0	After S = 0, R = 1
0	0	1	1	forbidden

5) JK Latch / JK Flip-Flop

JK Latch 는 SR Latch 에 추가적인 회로를 더해 S 와 R 이 동시에 1 인 상황에서도 정상적으로 작동하도록 수정한 Latch 이다.

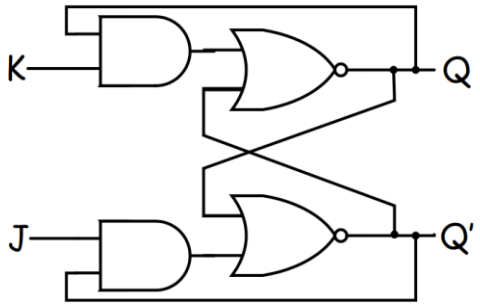


그림 4 – JK Latch

Latch 는 입력이 바뀔 때 출력도 즉시 바뀌는 비동기 회로라면, Flip-Flop 은 입력이 바뀌더라도 출력이 클럭에 맞추어 반영되는 동기 회로이다. 즉 JK Flip-Flop 은 클럭 신호를 추가로 받아 이에 맞추어 작동한다.

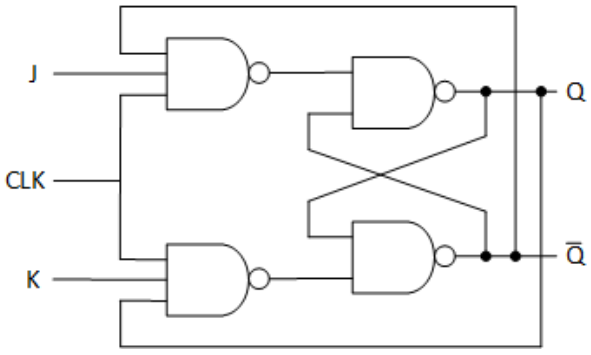


그림 5 – JK Flip-Flop

4) T Flip-Flop / D Flip-Flop

T Flip-Flop 과 D Flip-Flop 은 JK Flip-Flop 에서 입력값 J 와 K 가 함께 묶여있는 Flip-Flop 이다. T FF 은 Toggle 을 뜻하며, D FF 은 Delay 를 뜻한다. 따라서 T FF 은 입력값 T 가 1 일 때, 기존의 Q 의 값이 Toggle 되며, D FF 은 입력값 D 의 값이 Delay 되어 Q 는 D 의 값을 가지게 된다.

본 프로젝트에서는 카운터를 구현하기 위해 T FF 을, 레지스터를 구현하기 위해 D FF 을 사용한다.

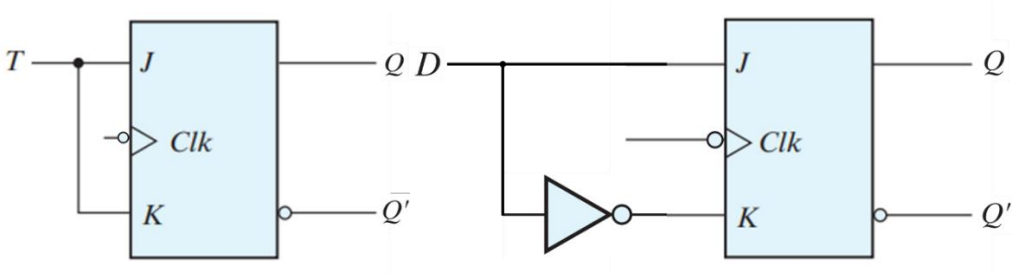


그림 6 – T Flip-Flop / 그림 7 – D Flip-Flop

5) Register / Counter

Register(레지스터)는 이진 정보를 저장하는 회로로, n 개의 FF 을 연결하여 n-Bit 의 정보를 저장한다. 일반적으로 프로세서에서 임시 저장소로 사용되며, 메인 메모리보다 빠르고 편리하다.

본 프로젝트에서는 D FF 을 사용하여 구현한다. 총 12 개의 레지스터를 연결하여 Shift Register 를 제작하고, 입력 받는 6 개의 숫자를 순서대로 저장한다.

Counter(카운터)는 정해진 수열을 순환하는 회로로, 카운터를 구성하는 FF 에 Clock pulse 가 작용하면 정해진 일련의 상태를 거치게 된다.

본 프로젝트에서는 T FF 을 사용하여 구현한다. 또한, Clock pulse 는 맞췄을 때 카운트가 될 수 있도록 Comparator 의 출력값을 이용한다.

7) FSM

FSM(Finite State Machine, 유한 상태 기계)은 state 개수가 유한한 순차 회로이다. FSM 은 크게 두 가지의 종류가 있는데, 각각 Moore Machine 과 Mealy Machine 이다. Moore Machine 은 출력값이 현재 상태에 대한 함수이며, 활성화된 Clock edge 와 동기화되게 변화(상태 전이)한다. Mealy Machine 은 출력값이 현재 상태와 입력값에 대한 함수로, 입력값이 변화하자마자 출력값이 변화한다.

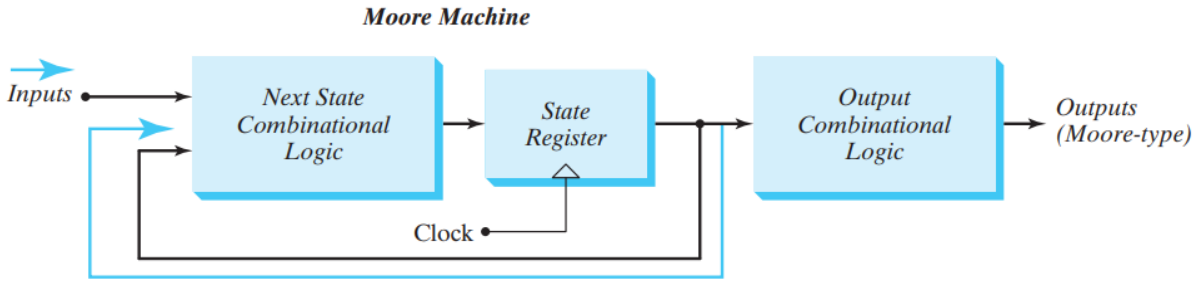


그림 7 – Moore Machine

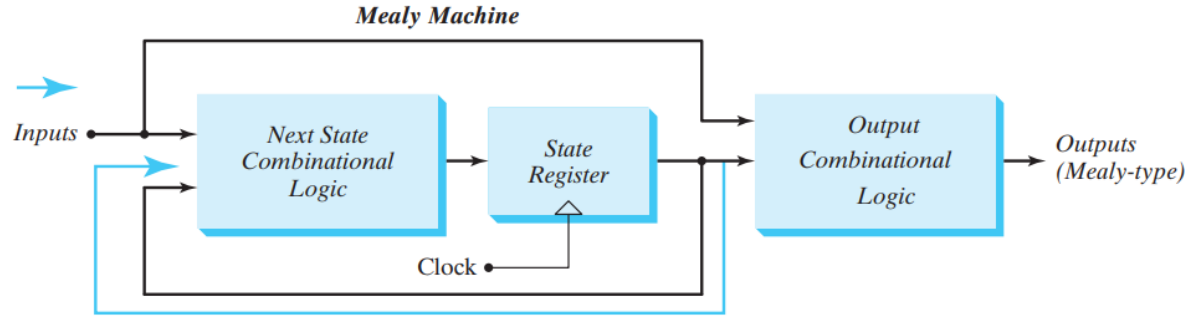


그림 8 – Mealy Machine

3. 설계

1) state transition diagram

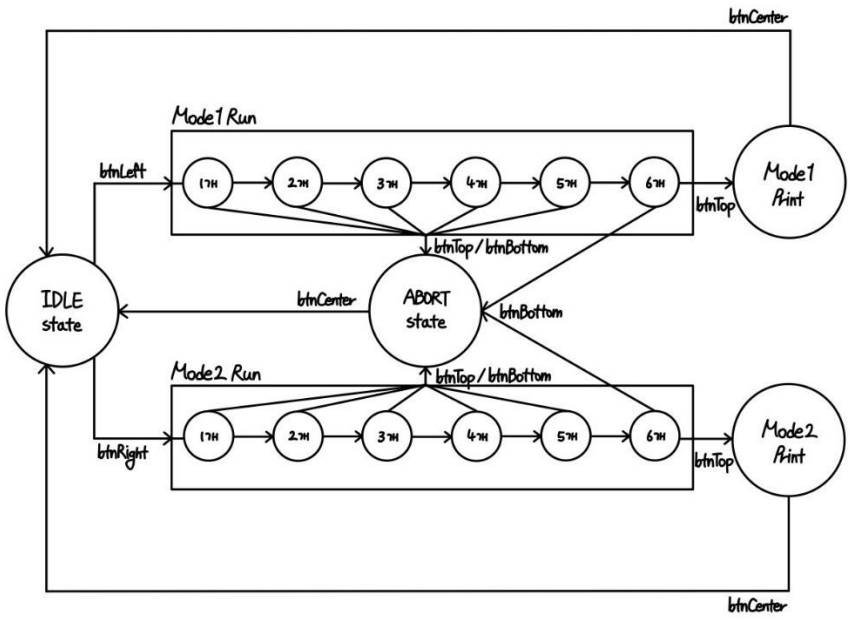


그림 9 – State Transition Diagram

2) FSM 동작

초기 state 인 IDLE state 에서 FPGA 의 LEFT 버튼을 누르면 MODE1_RUN state 로, RIGHT 버튼을 누르면 MODE2_RUN state 로 넘어간다.
먼저, MODE2_RUN state 는 로또 번호 설정 state 로, 이 state 에서는 사용자가 FPGA 의 스위치를 이용하여 입력한 숫자 6 개를 순서대로 레지스터에 저장한다. 또한, 숫자를 입력하면 7-segment 의 첫 번째 위치에는 입력 받은 숫자의 개수가 나타나고, 세 번째 위치와 네 번째 위치에는 실시간으로 입력 받은 숫자가 나타나게 된다. 만약 숫자 6 개를 입력 받는 과정에서 BOTTOM 버튼을 누르거나, 6 개가 모두 입력되지 않은 상태에서 TOP 버튼을 눌러 다음 state 로 진행하려고 하면 ABORT state, 즉 초기화 state 로 넘어가며, 7-segment 에 Err 를 출력한다. 이후 CENTER 버튼을 누르면 다시 IDLE state 로 돌아간다. IDLE state 에서 다시 MODE2_RUN state 로 들어가게 되면 이전 카운터는 초기화되어 새로 입력 받을 수 있다.
숫자 6 개를 모두 입력 받아 TOP 버튼을 누르게 되면 다음 state 인 MODE2_PRINT state 로 넘어가게 되며, 이 state 에서는 MOED2_RUN state 에서 입력 받은 숫자 6 개를 레지스터로부터 전달받아 순서대로 FPGA 의 7-segment 에 출력한다. 이후 CENTER 버튼을 누르게 되면 다시 IDLE state 로 돌아간다.
다음으로, MODE1_RUN state 는 로또 번호 맞추기 state 로, 이 state 에서는 사용자가 FPGA 의 스위치를 이용하여 입력한 숫자 6 개를 레지스터에 저장된 정답 숫자와 순서에 맞게 Comparator 를 통해 비교를 진행한다. 맞춘 경우 Comparator 의 출력값은 1 이 되고, 이를 카운터의 Clock pulse 로 넘겨 다음 카운트로 넘어간다. 맞춘 개수 카운터는 총 0 부터 6 까지 카운트를 진행한다.
MODE1_RUN state 역시 7-segment 의 첫 번째 위치에는 입력 받은 숫자의 개수가 나타나고, 세 번째 위치와 네 번째 위치에는 실시간으로 입력한 숫자가 나타나게 된다. 또한, 숫자 6 개를 입력 받는 과정에서 BOTTOM 버튼을 누르거나, 6 개가 모두 입력되지 않은 상태에서 TOP 버튼을 눌러 다음 state 로 진행하려고 하면 ABORT state 로 넘어가며, 7-segment 에 Err 을 출력한다. 이후 CENTER 버튼을 누르면 다시 IDLE state 로 돌아간다. IDLE state 에서 다시 MODE1_RUN state 로 들어가게 되면 이전 카운터는 초기화되어 새로 입력 받을 수 있다.
TOP 버튼을 눌러 MODE1_PRINT state 로 넘어가게 되면, 카운터의 출력값, 즉 정답을 맞춘 개수에 따라 FPGA 의 7-segment 에 등수(1st, 2nd, 3rd, 4th, ...)를 출력한다. 이후 CENTER 버튼을 누르면 다시 IDLE state 로 돌아간다. 다시 MODE1_RUN state 를 거쳐 MODE1_PRINT state 로 들어가는 경우, 맞춘 개수 카운터도 초기화되어 새로 맞춘 개수에 따라 등수가 출력된다.

3) FSM 회로 구조도

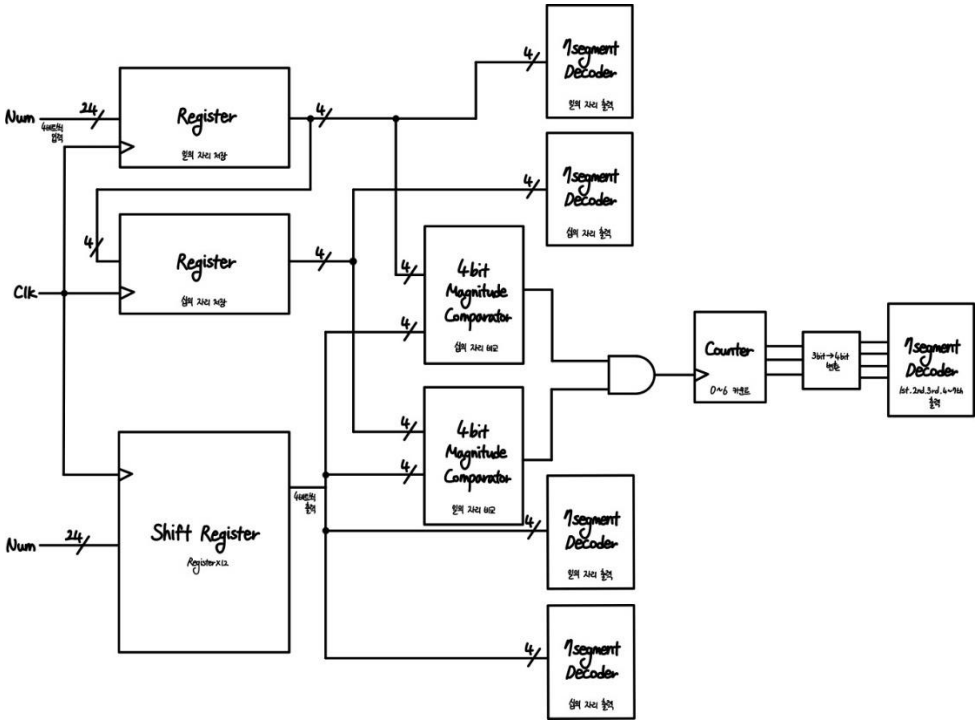


그림 10 – FSM 회로 구조도

4) FSM 세부 회로 설계

(1) 4-Bit Magnitude Comparator

두 개의 4-Bit 이진수를 입력으로 받아 모든 Bit 값이 동일한 경우 1 을, 하나라도 다른 경우 0 을 출력한다.

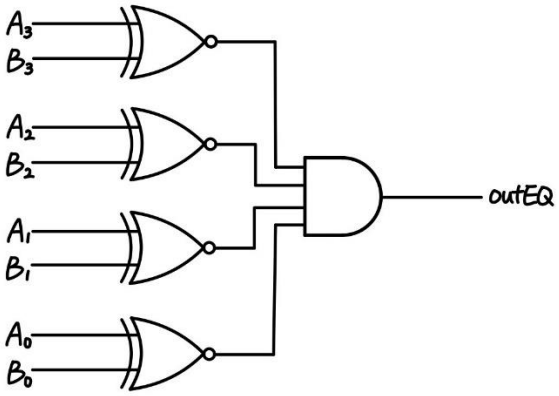


그림 11 – 4-Bit Magnitude Comparator

(2) Register

D FF 4 개를 이용하여 입력값을 저장한다.

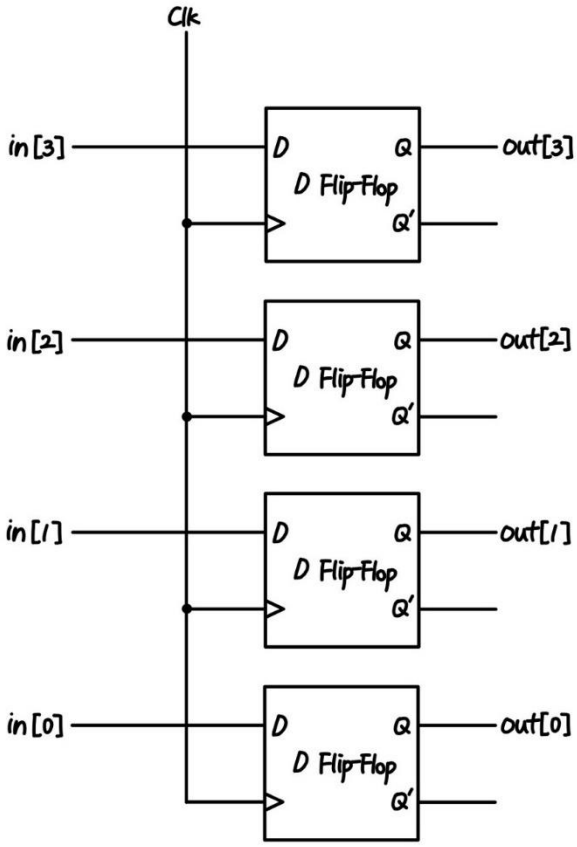


그림 12 – Register

(3) Shift Register

(2)에서 구현한 레지스터 12 개를 활용하여 Shift Register 를 설계한다. Clock pulse 에 맞춰 값이 입력될 때마다 저장되어 있는 값은 다음 레지스터로 값이 이동하며, 총 12 개의 숫자, 즉 두자리로 이루어진 6 개의 숫자를 저장한다.

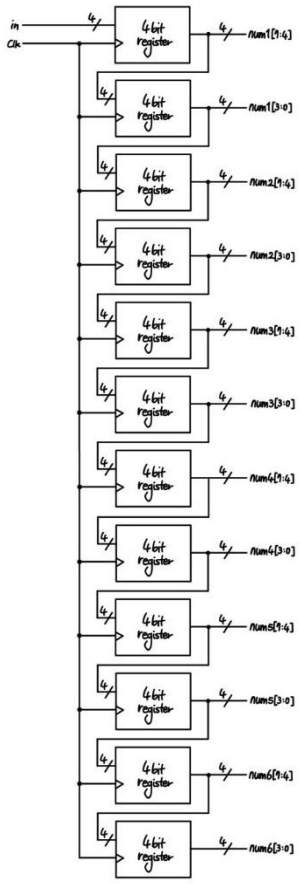


그림 13 – Shift Register

(4) Counter

0 부터 16 까지 카운트하는 회로이다. 4 개의 T FF 을 이용하며, Clock pulse 로는 4-Bit Magnitude Comparator 의 출력값을 이용하여, 맞춘 경우 Comparator 에서 1 이 출력되었을 때, 카운터가 활성화되어 다음 state 로 넘어가도록 설계한다.

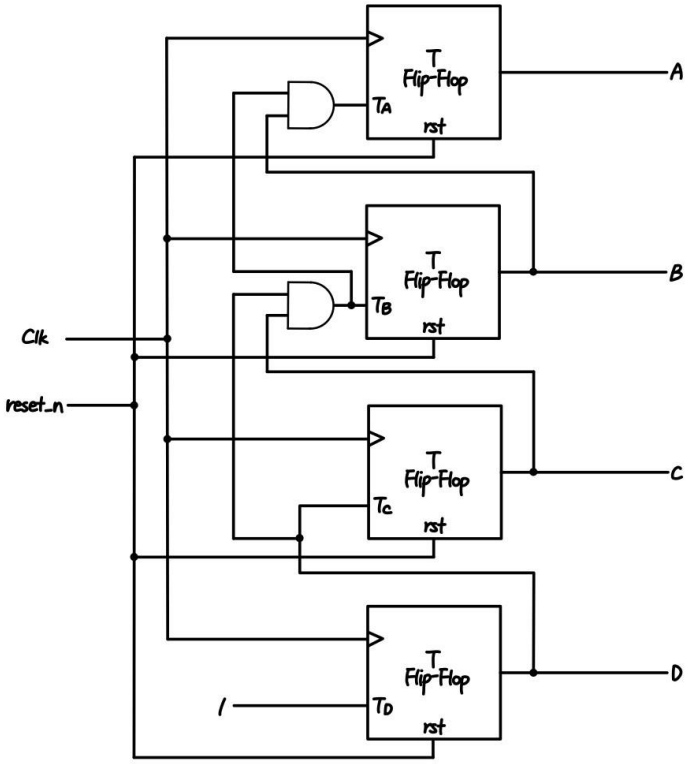


그림 14 - Counter

(5) 7-segment Decoder

입력 받은 4-Bit 에 맞춰 LED 에 표현할 위치를 활성화한다. 0~9 까지의 숫자와 등수 출력 및 상태 출력에 필요한 t, n, d, r, h, E 를 출력할 수 있다. 이때 s 는 5 로 대체한다.

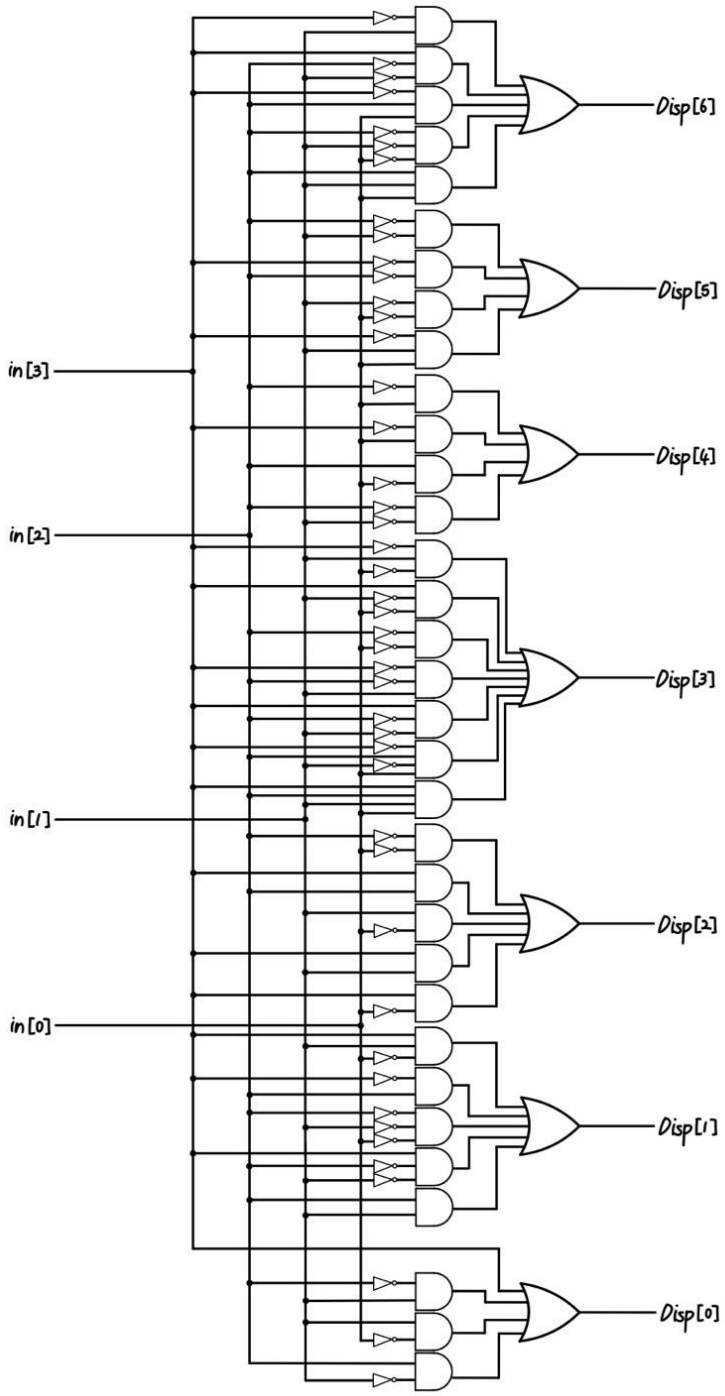
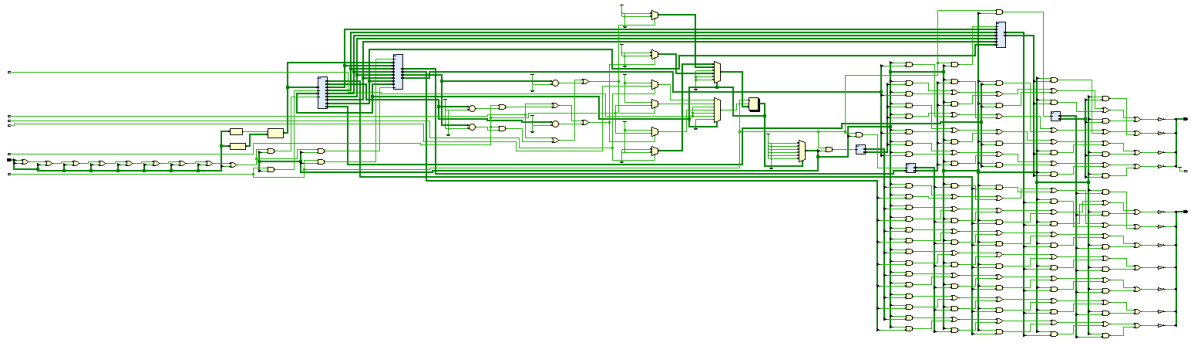


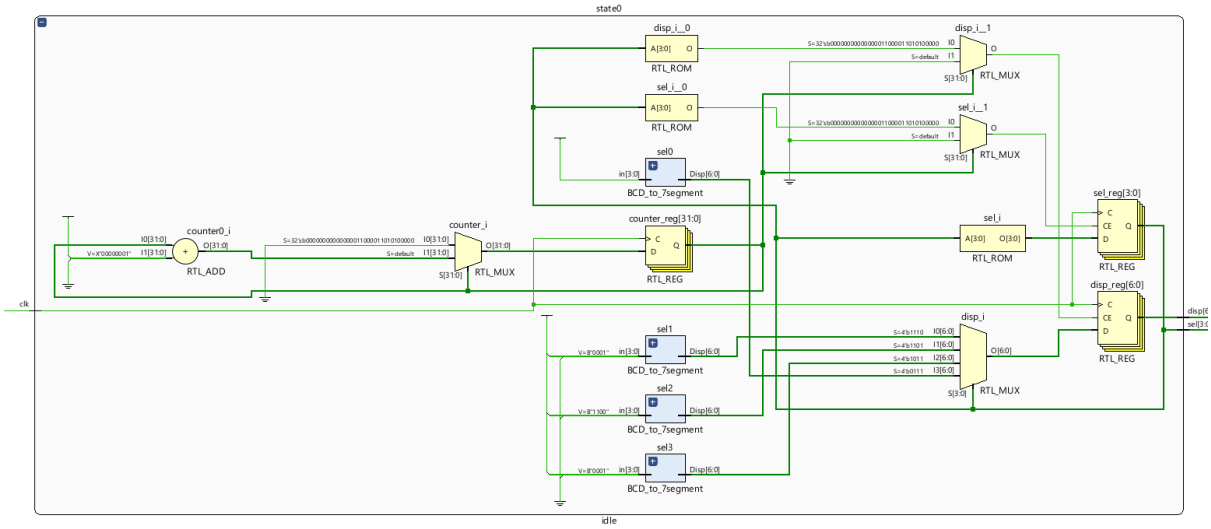
그림 15 – 7-segment Decoder

5) Schematic 회로 확인

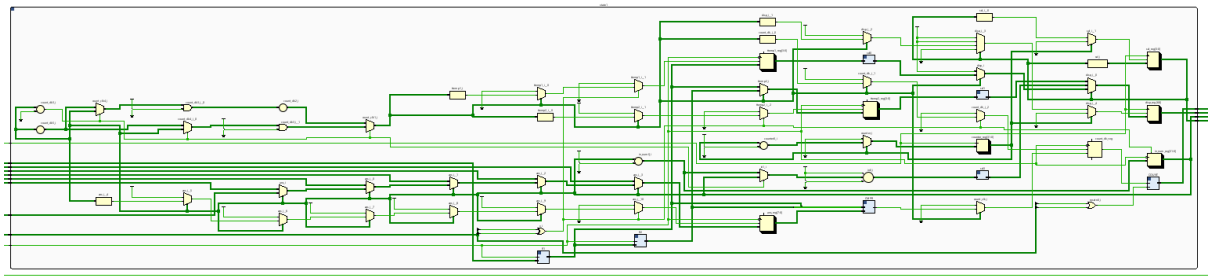
(1) 전체 회로도



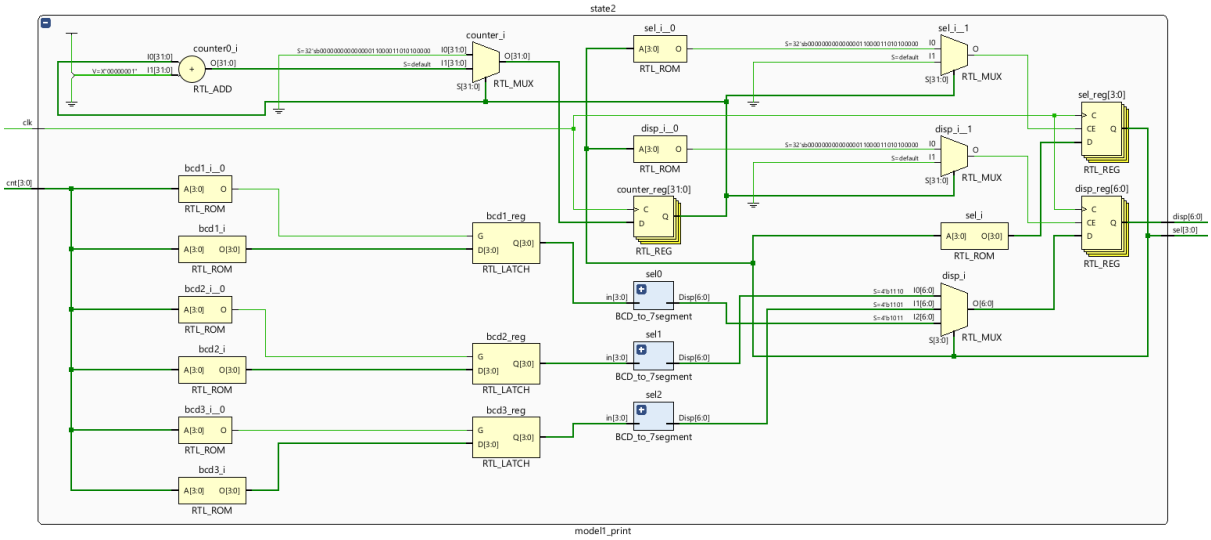
(2) IDLE state



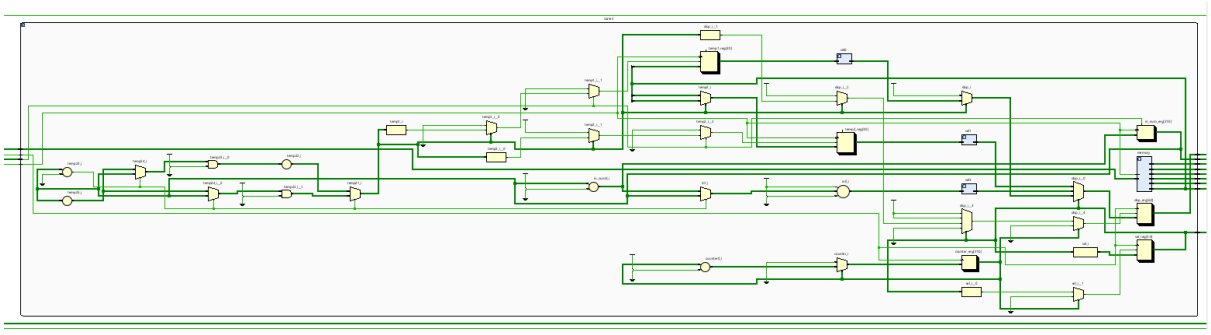
(3) MODE1_RUN state



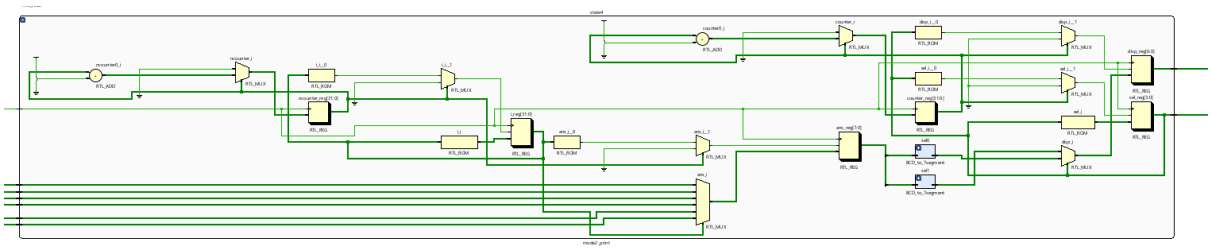
(4) MODE1_PRINT state



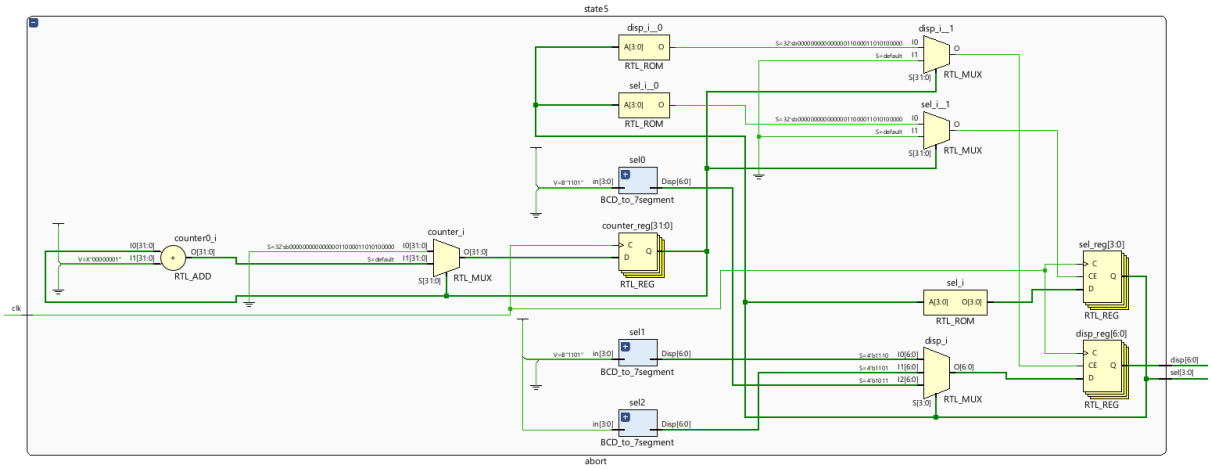
(5) MODE2_RUN state



(6) MODE2_PRINT state

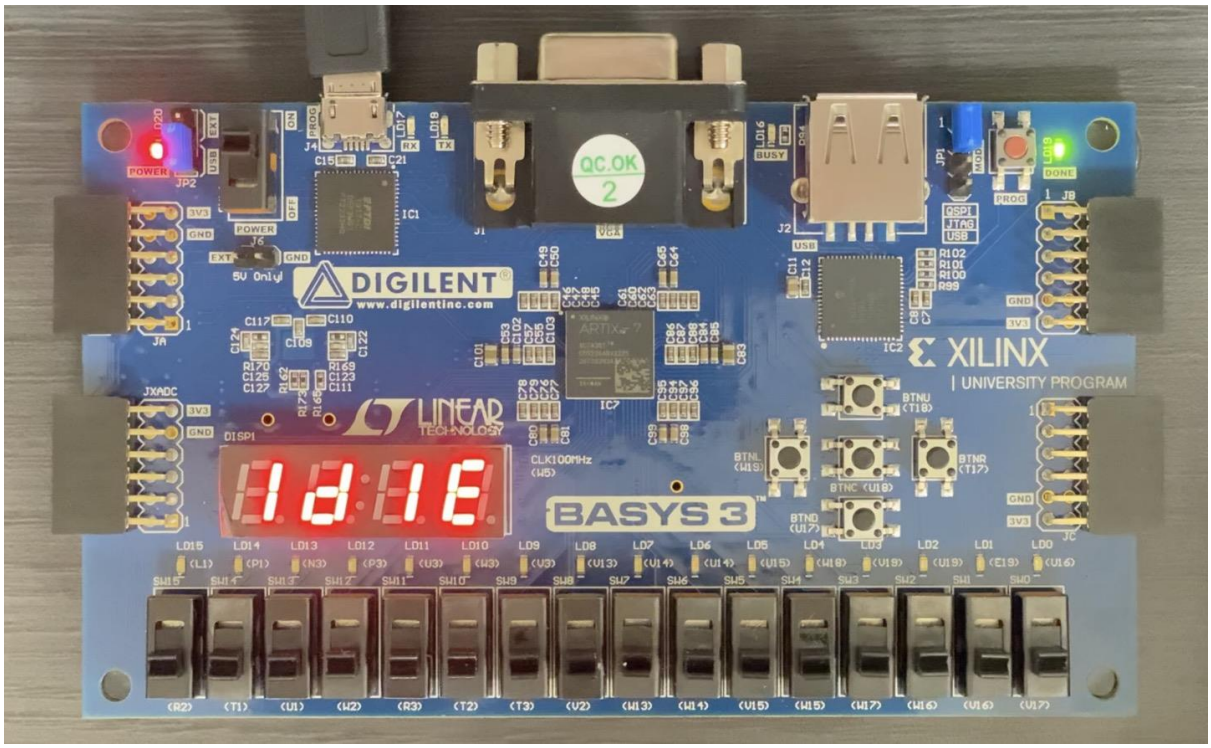


(7) ABORT state

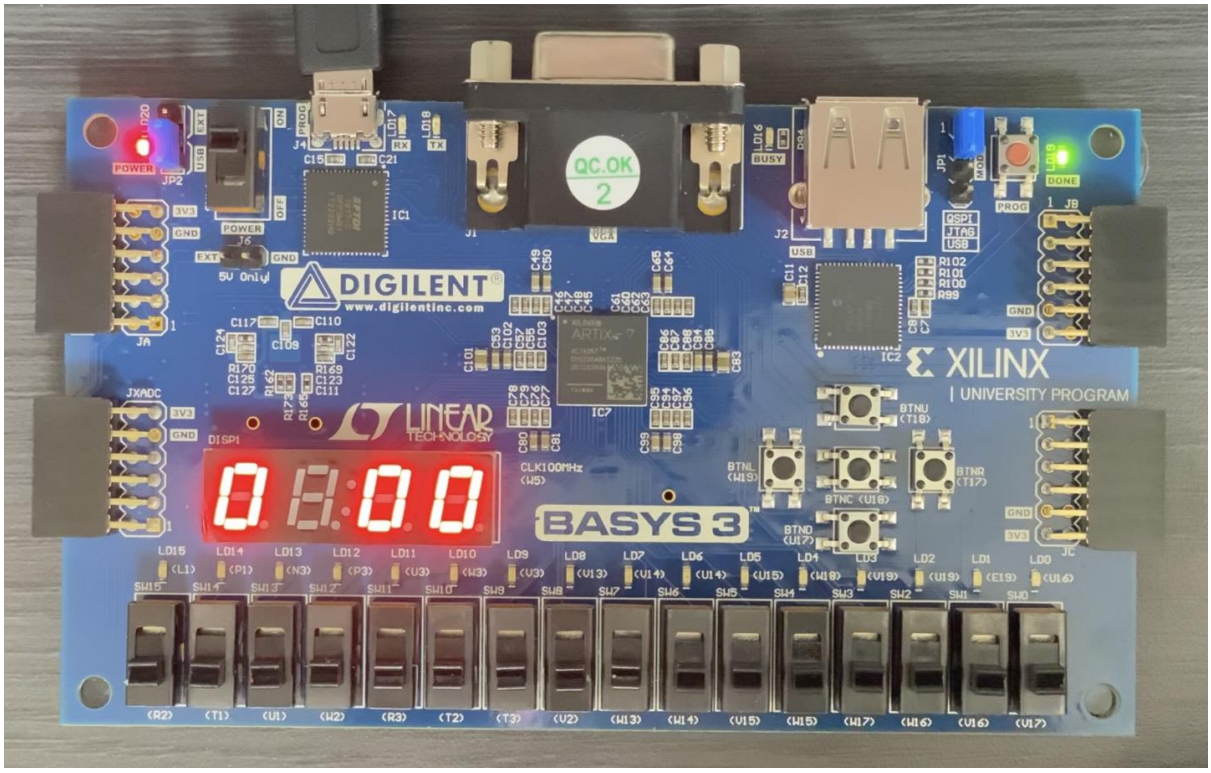


4. 결과

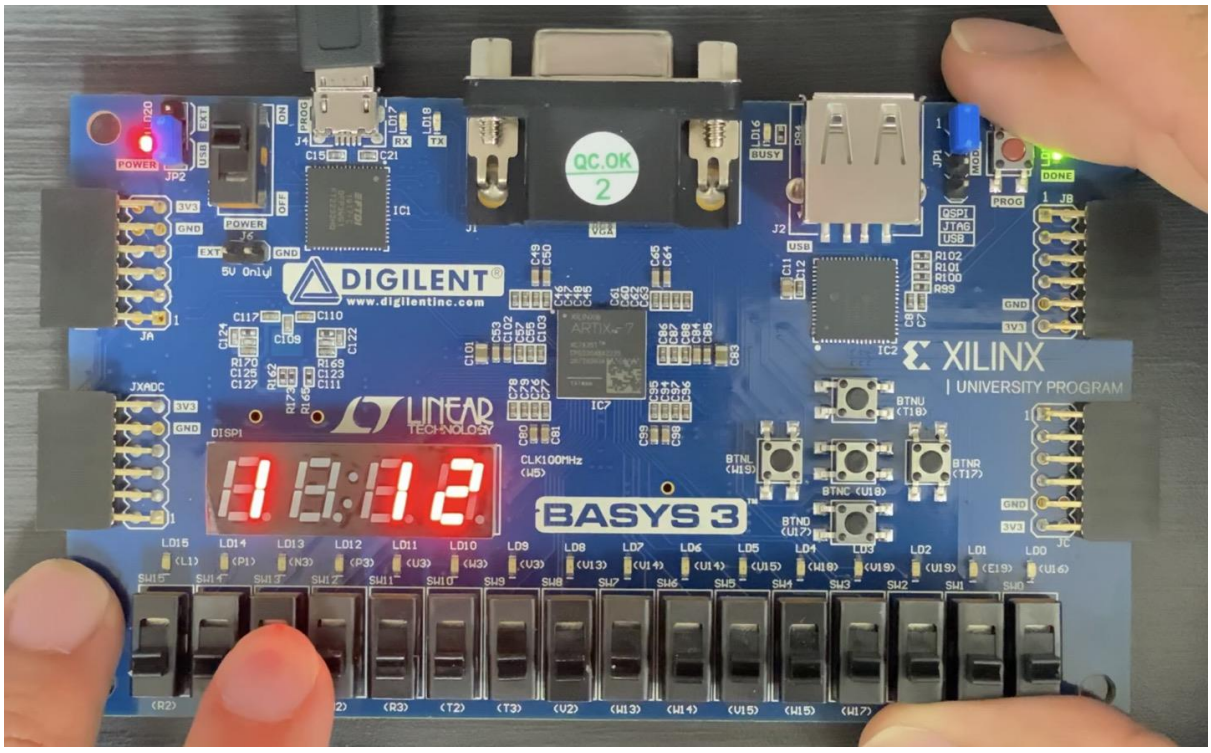
1) IDLE state



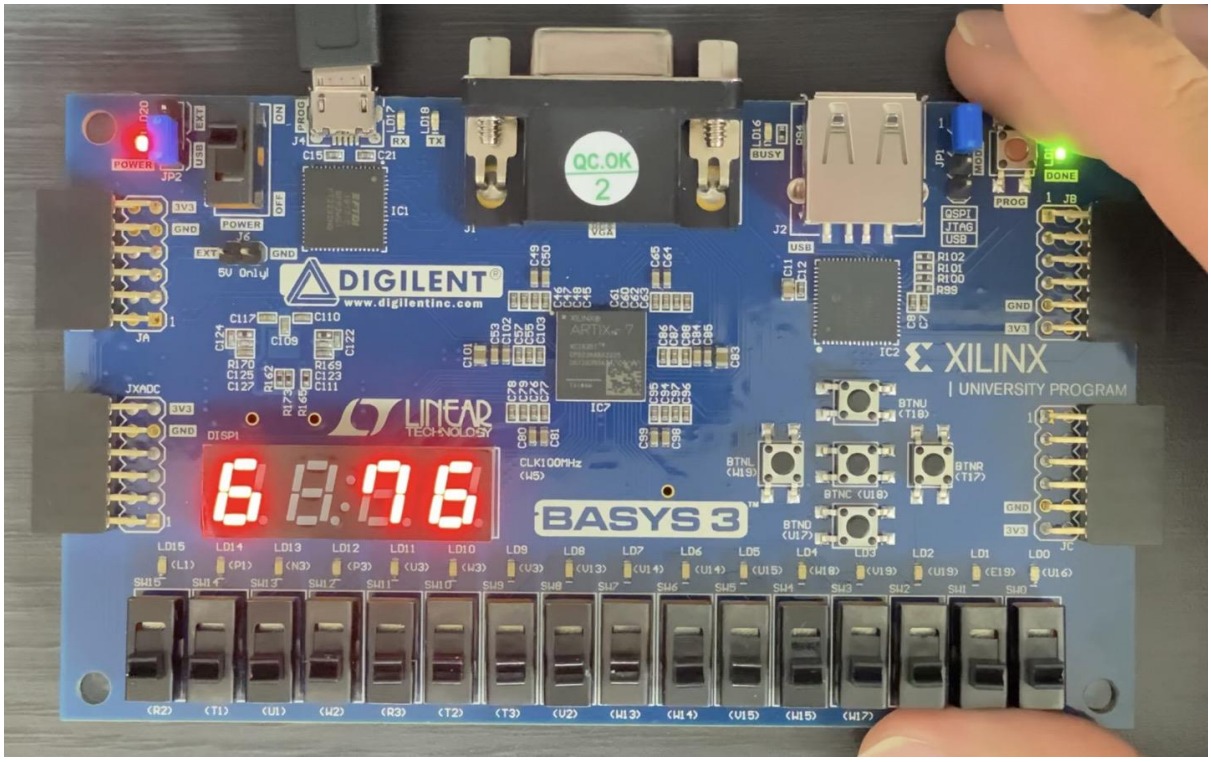
2) MODE2_RUN state



- 숫자 1 개 입력한 경우

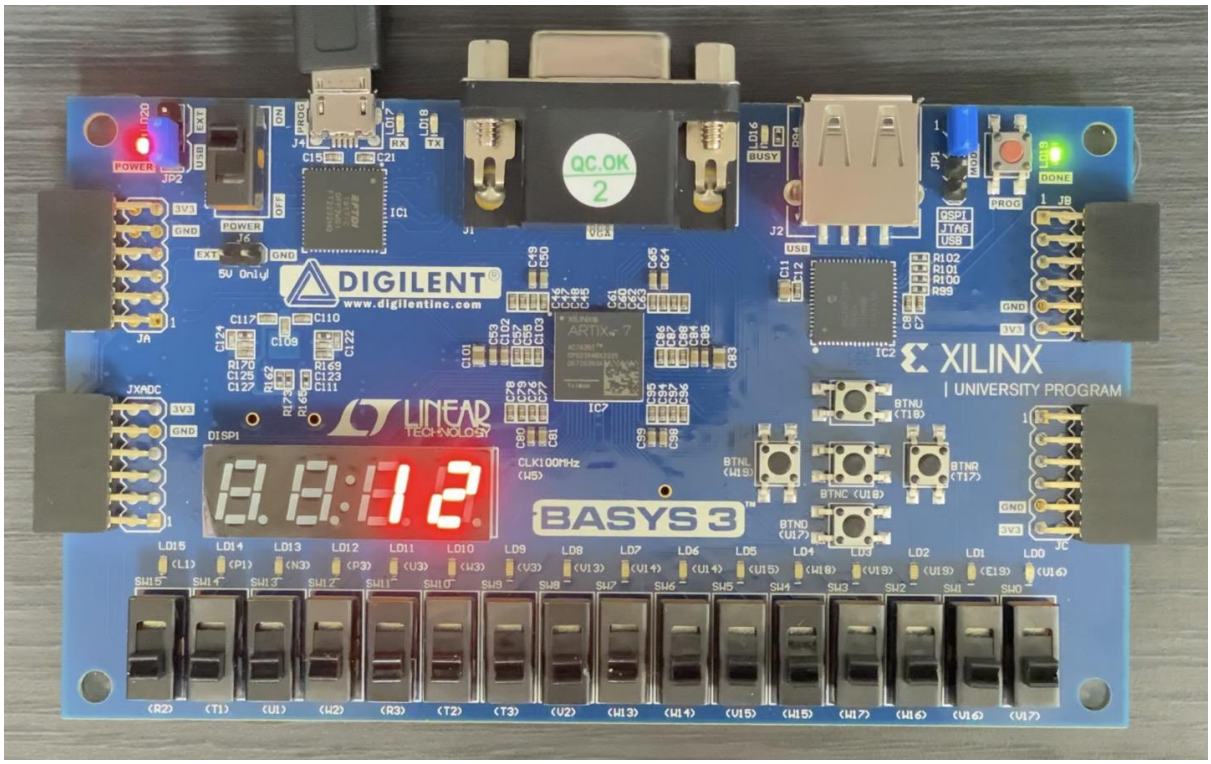


- 숫자 6 개 입력한 경우

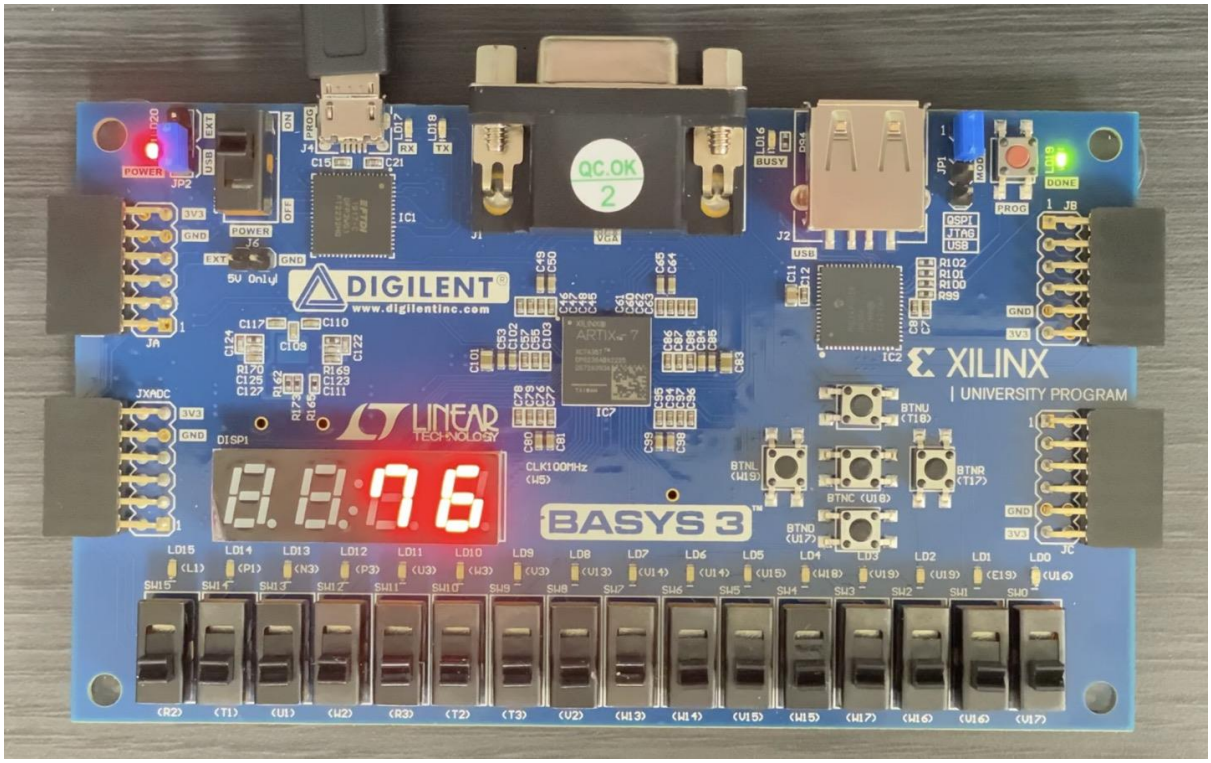


3) MODE2_PRINT state

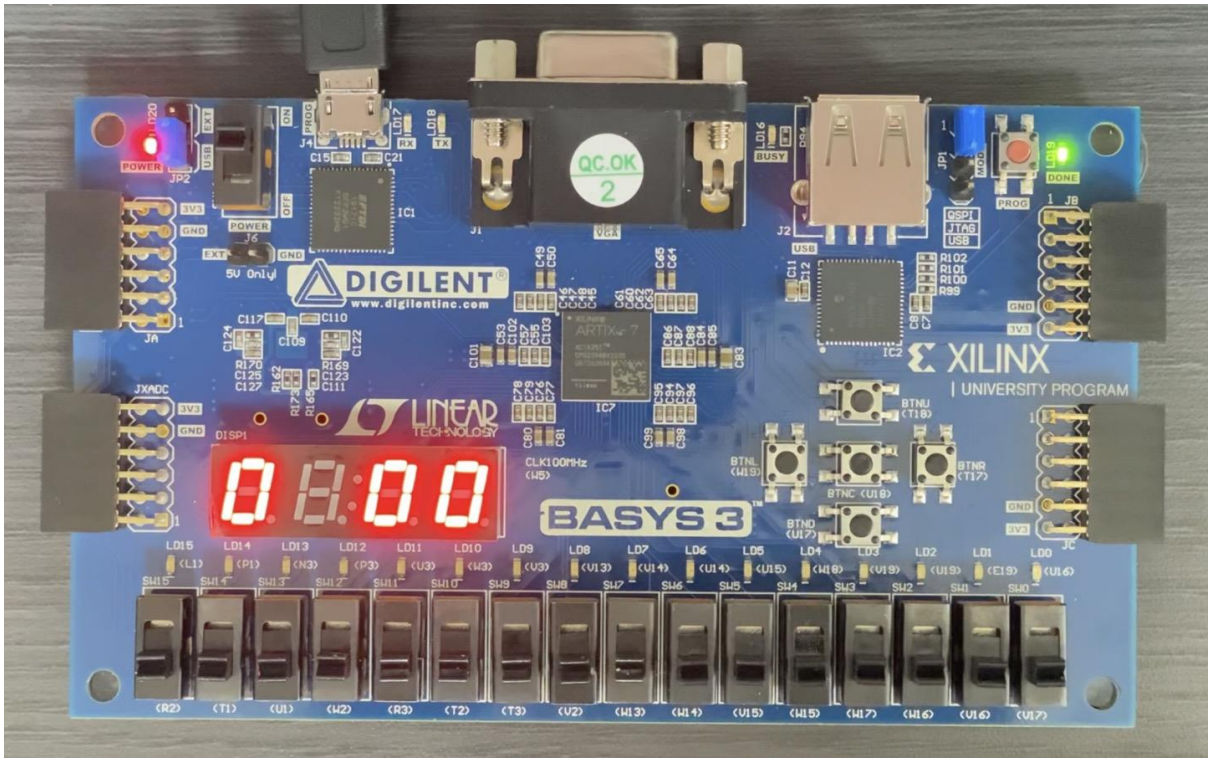
- 첫 번째로 입력한 숫자 12 표시



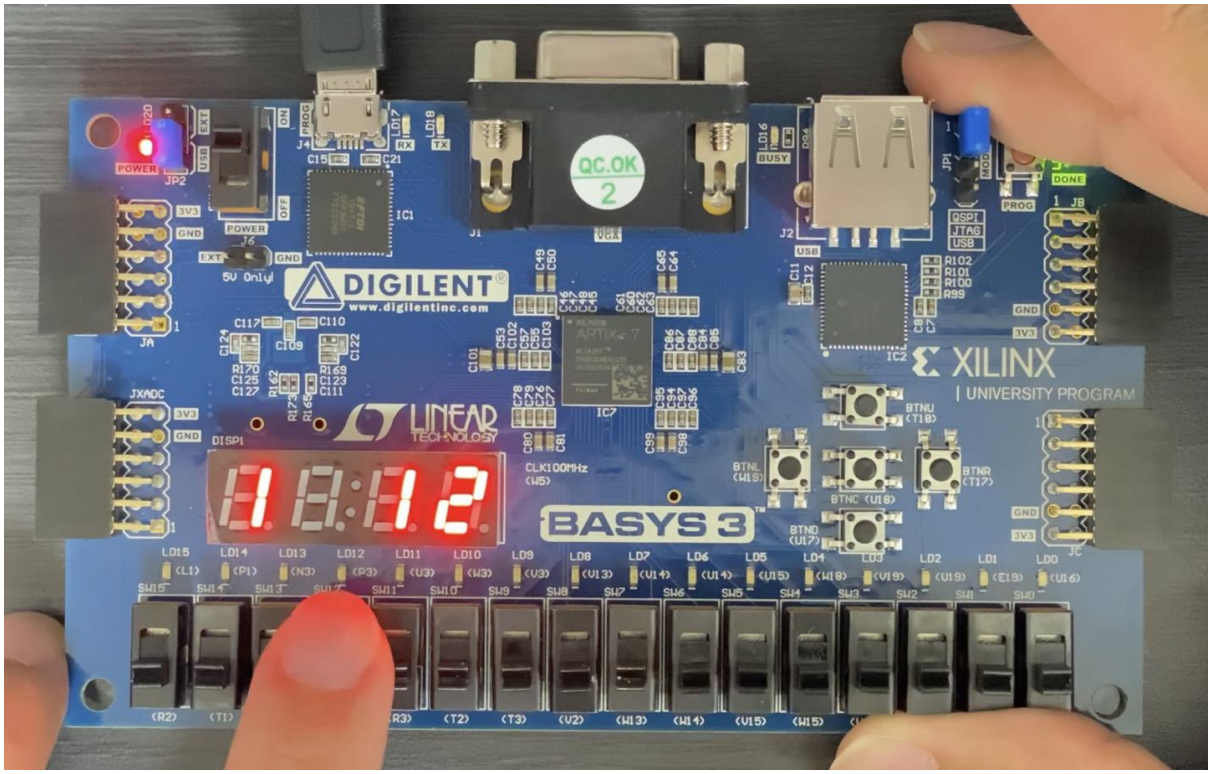
- 여섯 번째로 입력한 숫자 76 표시



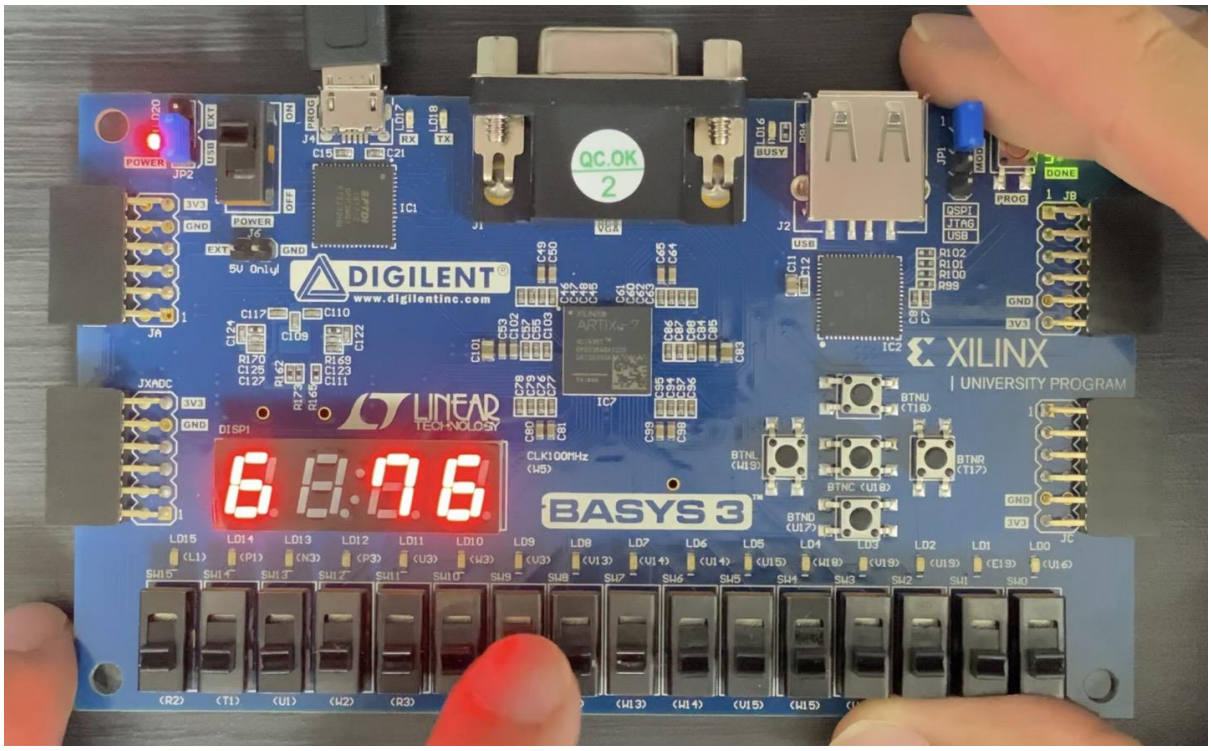
4) MODE1_RUN state



- 숫자 1 개 입력한 경우

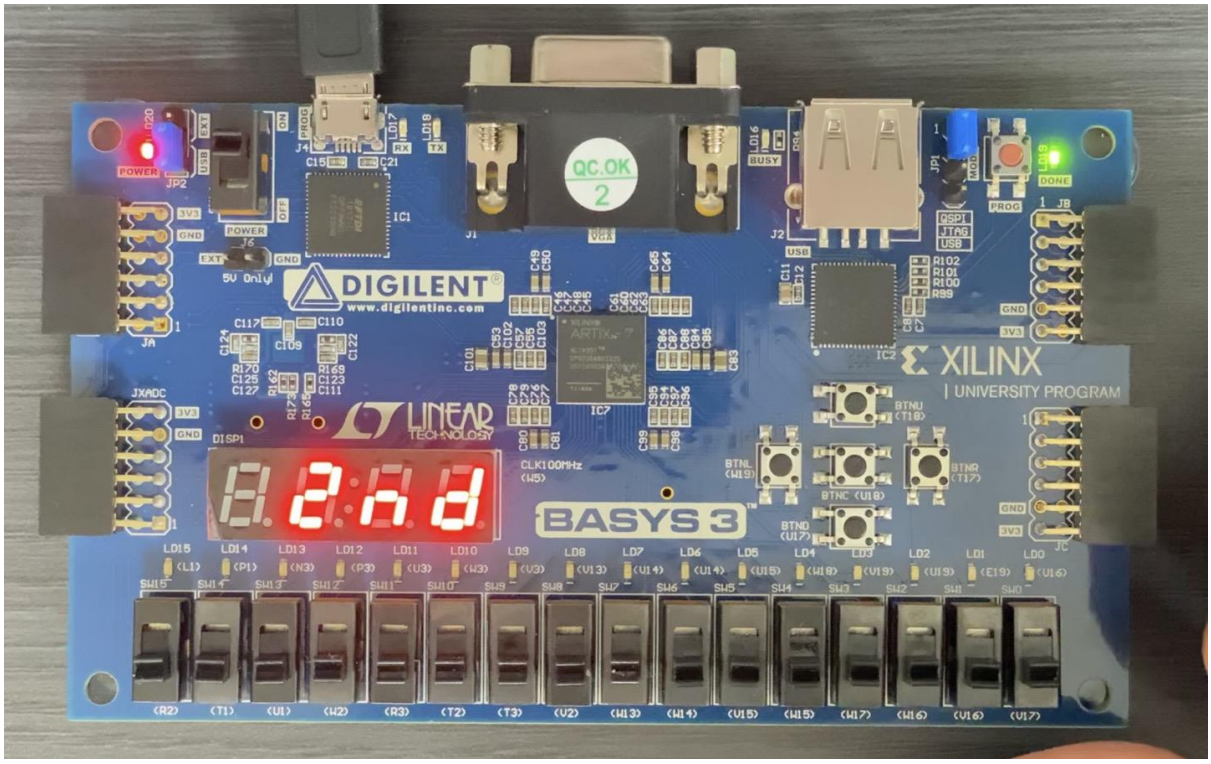


- 숫자 6 개 입력한 경우

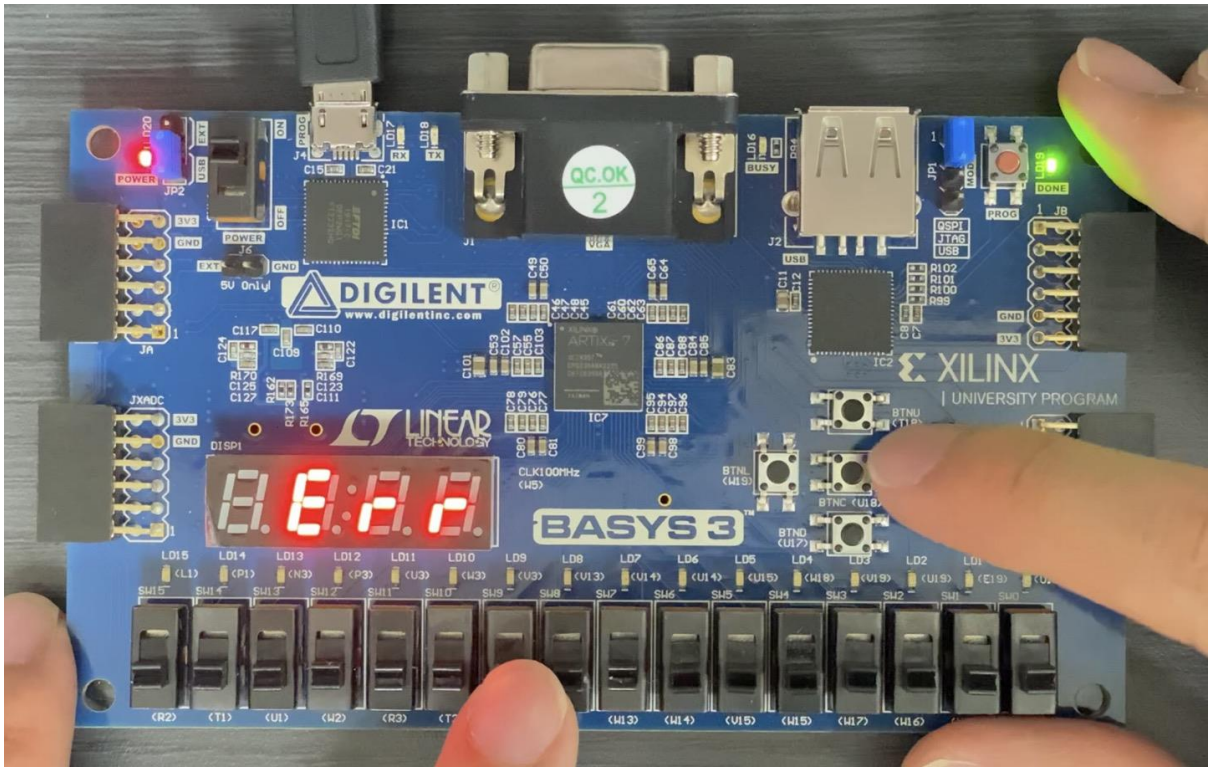


5) MODE1_PRINT state

- 숫자 1 개 틀린 경우



(6) ABORT state



5. 논의

1) 어려웠던 점 및 해결 방법

처음 Verilog 로 코드를 짜는 과정에서 wire 와 reg 의 구분이 어려웠던 것 같다. assign 문을 활용하기 위해서는 wire 로, always, initial 문을 활용하기 위해서는 reg 로 선언해주어야 한다는 것을 알게 되었다.

가장 빈번하게 등장했던 오류로는 Multiple Driver Nets 였다. 이는 multiple output ports 나 multiple continuous assignment 가 한 개의 변수를 drive 하는 경우 발생한다. 이 문제는 변수 이름을 바꿔주어 구분해줌으로써 해결할 수 있었다.

마지막 단계에서는 카운터 초기화를 위해서 state 가 넘어갈 때마다 Clock pulse 를 어떻게 넘겨주어야 그 signal 을 인지하여 카운터 값에 reset_n 의 값을 넘겨줄 수 있을지 많이 헤맸던 것 같다. 이를 해결하기 위해서 공부하는 과정에서 always 문 관련하여 문법을 이해할 수 있었고, always 문에 두 개의 edge signal 을 넘겨줄 수 있고, 그 안에서 if 문을 활용하여 두 개의 signal 을 구분할 수 있다는 것을 알게 되어 원하는 state 로 넘어갈 때마다 reset_n 의 값을 넘겨주는 작동을 구현할 수 있었다.

2) 추가 구현 가능성

shift register 로 구현하여 숫자를 입력할 때마다 기존의 register 에 저장되어 있던 값은 다음 register 로 넘어가게 되고 12 번째 register 를 넘어가면 값이 사라지게 된다. 따라서 6 개의 숫자를 정확히 입력하는 본 FSM 에서는 크게 레지스터를 초기화하는 과정이 필요하지 않았다. 하지만, MODE1_RUN state 와 MODE2_RUN state 로 넘어갈 때마다 초기에 7-segment 에 마지막에 입력한 숫자가 저장된 레지스터의 값으로 표현되기에, 더 정교한 FSM 을 설계하기 위해서는 1 번째와 2 번째 레지스터 값을 초기화하여 7-segment 에 00 이 뜨도록 구현하면 더 깔끔한 모습의 FSM 을 설계할 수 있을 것이라고 생각한다.

MODE1_RUN 과 MODE2_RUN 모두 숫자 6 개를 덜 입력하거나, 더 입력하는 경우 ABORT state 로 넘어가게 되며, 7-segment 에 Err 가 뜨도록 구현하였다. 추가 구현은 아니지만, 다른 방식으로 구현할 수 있는 아이디어를 떠올려보면, 사용자가 6 개의 입력을 다 하지 않은 경우에는 6 개를 모두 입력할 때까지 state 를 넘어가지 못하게 구현해볼 수 있을 것이라 생각하였다.