마피아 게임 설계 보고서

2022. 06. 15

디지털시스템설계(CSED273) 27조

20190936 곽찬영

20200437 김채현

20200794 박건우

1. 프로젝트 주제 및 목적

본 프로젝트에서는 마피아 게임을 FPGA를 통해 구현하고자 한다. 마피아 게임은 마피아와 시민이 대치하면서 진행되는 팀 게임이다. 게임의 룰에 대해 간략히 설명하자면 처음에는 4명의 시민과 2명의 마피아로 게임을 시작한다. 마피아는 서로를 알지만 시민은 마피아가 누구인지 모르기에 마피아로 추정되는 사람을 낮에 투표를 통해 처형하게 된다. 밤이 되면 시민은 더 이상 행동하지 못하고 마피아는 시민을 암살하게 된다. 이때 시민은 마피아를 전부 처형해야 승리하고, 마피아는 마피아가 전체 인원의 과반수가 될 때 승리하는 룰을 가지고 있다. 본 프로젝트에서는 이를 FPGA를 통해 구현하고 현재 상태를 서로 다른 색의 LED를 통하여 표현하여 가시성을 높이고, 이러한 상태를 사용자의 조작을 통해 직접 변화시키는 것에 그 목적을 두고 있다.

2. 구현을 위한 배경지식

- FSM (Finite-state machine)

  FSM(Finite-state machine)이란 전자 논리 회로를 설계하는 데 쓰이는 수학적 모델이다. 유한한 개수의 상태를 가지며, 한 번에 오로지 하나의 상태(state)를 가질 수 있다. 상태는 주어지는 입력(input)에 따라 어떤 상태에서 다른 상태로 전이되거나 다른 출력(output)을 내게 된다.

  FSM에는 Mealy machine과 Moore machine이 존재한다.

  - Mealy machine

Mealy machine은 출력이 현재의 입력과 상태에 의해 바로 결정된다. 따라서 Mealy machine은 입력이 직접 출력에 영향을 주게 된다. 또한 clock을 한 사이클 기다리지 않고, 같은 사이클에서 input을 바로 output에 반영하기 때문에 Moore machine보다 더 빠르다는 장점이 있다.

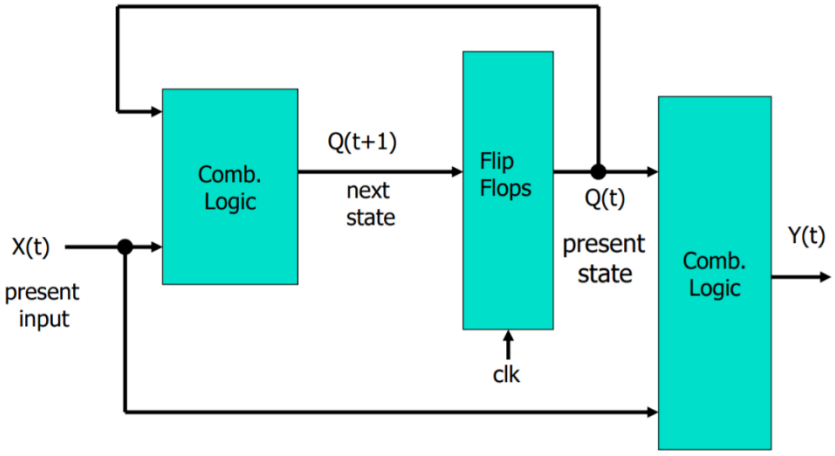


그림 1. Mealy machine 회로도

- Moore machine

Moore machine은 출력이 현재 상태에 의해서만 결정된다. 따라서 Moore machine은 입력이 무조건 순차 회로를 한 번 거쳐 출력된다. 출력이 clock edge에서 바뀌므로, 즉, 입력이 들어온 후 한 사이클 뒤에 변화하기 때문에 Mealy machine보다 사용하기에 더 안전하다는 장점이 있다.

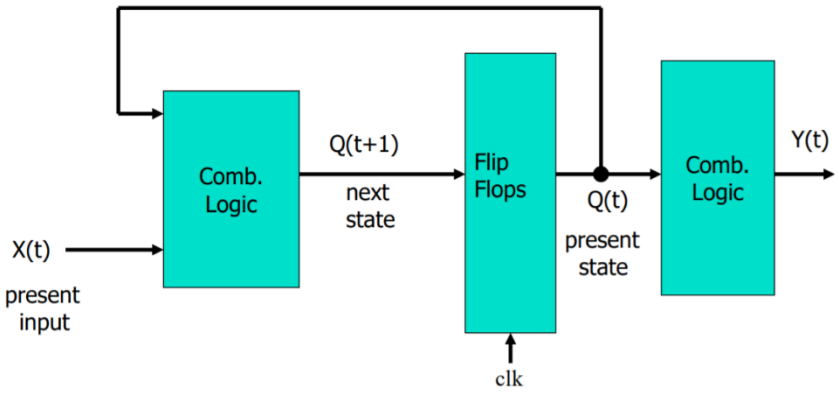


그림 2. Moore machine 회로도

Mealy machine에서는 입력이 출력에 영향을 바로 미치기 때문에 2개 이상의 기계에서 비동기화 문제가 발생할 수 있다. 하지만 Mealy machine이 Moore machine보다 더 적은 개수의 state를 갖는다. 이런 Moore machine과 Mealy machine의 장점을 합친 것이 synchronous mealy machine이다. flip-flop을 사용하여 clock에 맞춰 상태를 동기화함으로써 glitch에 대한 문제를 피할 수 있다.

- Counter

Counter, 즉 계수기는 클록 펄스에 맞춰 상태를 변화시키는 논리 회로이다. 클록 펄스에 따라 일정 시간이 지나면 처음의 상태로 돌아오게 설계하며, 주로 Count-up 또는 Count-down 방식을 사용한다. 계수기의 구현에 따라 다양한 방식으로 출력값과 그 순서를 지정할 수 있다. 계수기를 여러 개 연결하면 더 큰 범위의 수를 세는 계수기도 제작이 가능하다.

- JK Flip-Flop

JK Flip-Flop는 두 개의 입력에 따라 저장된 값을 조정할 수 있는 동기회로이다. 두 개의 입력 J와 K의 조합으로 서로 다른 연산을 할 수 있다. JK Flip-flop 내부에는 JK 래치가 존재하는데, (J, K)가 (0, 0)일 때는 Hold, (1, 0)일 때는 Set, (0, 1)일 때는 Reset, (1, 1)일 때는 Toggle 연산을 수행하여 래치가 저장하는 값을 바꿀 수 있다. 또한 JK Flip-Flop을 사용하면 D Flip-Flop이나 T Flip-Flop도 손쉽게 제작할 수 있다는 면에서 범용성이 매우 높다.

- FPGA

FPGA란 설계 가능 논리 소자와 프로그래밍이 가능한 내부 회로가 포함된 반도체 소자이다. 논리회로를 원하는 의도에 맞춰 동작하게 할 수 있기 때문에 본 프로젝트에서는 HDL인 Verilog를 프로그래밍하여 사용할 예정이다. FPGA의 내부에는 로직을 만들어내는 RAM과 flip-flop이 있어 로직 게이트를 자유자재로 구현할 수 있다. 일반적인 프로세서는 memory에 있는 프로그램을 불러와 해독하여 작업을 실행하지만 FPGA의 경우 아예 프로세서 내부의 회로를 프로그램에 맞게 직접 설계하여 병렬적으로 실행시키므로 압도적인 속도를 낼 수 있다는 장점이

있다.

3. State transition diagram 설계

State에 대한 설명은 다음과 같다.

- State는 게임의 낮/밤 여부와 생존자의 명수에 따라 결정된다.
- 낮과 밤의 State가 구분되며, 낮의 State는 D, 밤의 State는 N으로 표시한다.
- 생존해 있는 마피아의 수와 시민의 수를 State의 이름에 포함하여 각 State가 어떤 의미인지 쉽게 파악할 수 있도록 한다. 예컨대 낮에 마피아 2명과 시민 3명이 생존해 있다면 "D 2:3"으로 표현하며, 밤에 마피아 1명과 시민 2명이 생존해 있다면 "N 1:2"로 표현한다.

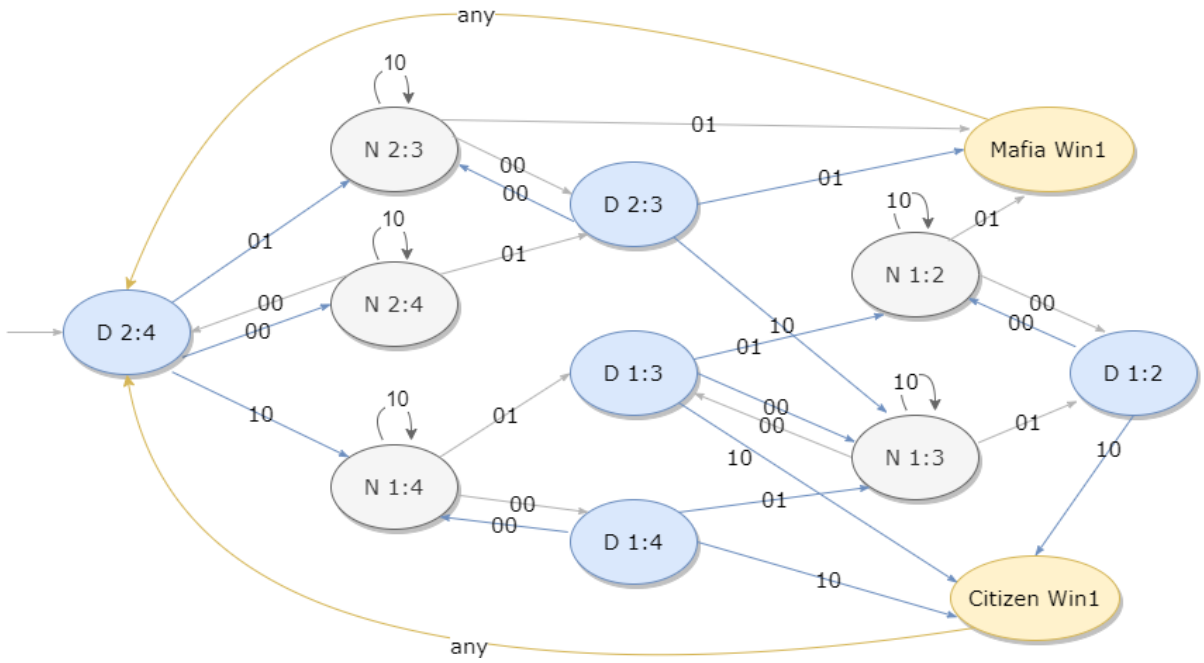


그림 3. State Transition diagram

4. State machine 입출력 및 동작 설명

State Machine의 입력으로는 2비트의 이진 입력과 Enable, Reset이 주어진다. 먼저 사

용자는 Reset 스위치를 올리고 Enable 버튼을 눌러 입력값을 줌으로써 게임을 시작하게 된다. 이후 게임이 시작되면 매 차례에 사용자는 스위치를 통해 Input 값을 조정한 후 Enable 버튼을 눌러 Transition이 일어나게 한다. 입력값은 낮 State와 밤 State에서 서로 다른 의미로 해석된다.

낮/밤	I_1	I_0	의미	State transition
D	0	0	투표 건너뛰기	D n:m → N n:m
D	0	1	시민 처형	D n:m → N n:(m-1)
D	1	0	마피아 처형	D n:m → N (n-1):m
N	0	0	의사의 치료로 아무도 죽지 않음	N n:m → D n:m
N	0	1	시민이 암살당함	N n:m → D n:(m-1)
N	1	0	밤이 계속됨	N n:m → N n:m

표 1. 각 낮/밤에서의 Input의 의미

먼저 게임을 시작하기에 앞서 가장 기본적인 상황을 설명하면, 낮과 밤을 표현하는 흰색 LED 1개, 살아있는 마피아의 숫자를 나타내는 붉은 색 LED 2개, 살아 있는 시민의 숫자를 나타내는 초록색 LED 4개가 있고, 사용자의 입력을 받을 FPGA의 버튼이 있다. 흰색 LED는 낮일 때 불이 들어와 있고, 밤이 되면 불이 꺼진다. 빨간색 LED는 살아 있는 마피아의 숫자에 같게 불이 들어와 있고, 죽을 때마다 가장 밑의 LED부터 하나씩 꺼진다. 녹색 LED 또한 살아 있는 시민의 숫자에 같게 불이 들어와 있고, 죽을 때마다 가장 밑의 LED부터 하나씩 꺼진다.

처음 게임을 시작하면 낮 상태로 시작하게 된다. 낮 시간대에서의 입력은 00, 01, 10 세 가지가 있다. 첫 번째 경우(input 00)는 투표를 건너뛰는 것으로 아무도 죽지 않고 저녁 시간대가 된다. 2번째 경우(input 01)는 시민이 투표로 처형되는 것으로 시민의 숫자를 나타내는 LED 중 가장 오른쪽에 있는 것이 하나 꺼진다. 마지막 경우(input 10)는 마피아가 투표로 처형되는 것으로 마피아의 숫자를 나타내는 LED 중 가장 오른쪽에 있는 것이 꺼지게 된다.

다음으로 낮 시간대에서 입력이 끝난다면 밤 상태가 되게 된다. 밤 시간대에도 동일하게 00, 01, 10의 3가지의 입력이 있다. 첫 번째 경우(input 00)는 마피아가 시민을 암살하려 하였지만 의사가 치료하여 아무도 죽지 않은 경우이다. 이러한 경우에는 마피아, 시민

을 나타내는 LED의 변화 없이 시간대만 다시 낮으로 바뀌게 된다. 두 번째 경우(input 01)는 시민이 마피아에게 암살당하는 경우이다. 이러한 경우에는 살아있는 시민을 나타내는 LED 중에서 가장 오른쪽에 있는 것의 불이 꺼지게 되면서 다시 낮 시간대가 된다. 마지막 경우(input 10)는 밤이 계속되는 경우인데, 이러한 경우에는 빨간색, 흰색, 초록색 LED 모두 변화가 없고 다시 한번 input을 받게 된다.

이렇게 계속 낮 상태와 밤 상태를 반복하면서 마피아가 모두 죽어 시민의 승리 상태가 되면 빨간색 LED의 불이 모두 꺼지고 초록색 LED의 불이 모두 켜진다. 또한 FPGA의 스크린을 통해 "CITIZEN WIN"이라는 문구를 띄우고 어떤 입력이 주어지든 처음 상태로 돌아가게 된다. 마찬가지로, 마피아가 과반수를 차지하게 될 경우 마피아의 승리 상태가 되어 빨간색 LED의 불이 모두 켜지고 초록색 LED의 불이 모두 꺼진다. 더불어 FPGA의 스크린을 통해 "MAFIA WIN"이라는 문구가 출력되고, 다음 입력이 주어질 때 처음 상태로 돌아가게 된다.

만약 게임 도중 알맞지 않은 입력값인 11이 주어지면 상태는 변화하지 않으며, FPGA의 스크린에 "INVALID"이라는 문구가 출력되며 다시 입력값을 받는 상태로 돌아간다. 또한 게임 중간에 Reset 입력을 주게 되면 게임은 처음 상태 (낮, 시민 4명, 마피아 2명)으로 돌아가게 된다.

5. 상태 전이표 작성 및 식 간단화

위와 같은 동작을 구현하기 위해 다음과 같이 상태 전이도를 구성하였다.

4-bit 값인 ABCD는 현재 어떤 상태에 위치해 있는지를 의미한다. A는 낮과 밤을 나타낸다. A의 값이 1이면 낮, 0이면 밤이며, 이에 따라 흰색 LED의 개수도 변화한다. B는 마피아의 명수를 나타낸다. B가 1이면 마피아 2명이 모두 생존해 있음을 의미하고, 0이면 마피아가 단 한 명만 생존해 있음을 뜻한다. CD는 시민의 명수를 나타낸다. CD가 11이면 시민 4명이 모두 생존해 있는 것이며, 10이면 3명, 01이면 2명만 생존해 있음을 나타낸다. 이 규칙에는 두 가지 예외가 있는데, 마피아가 이기는 경우와 시민이 이기는 경우이다. ABCD = 0000은 마피아가 과반수가 되어 승리한 경우이고, ABCD = 0100은 마피아가 모두 사망하여 시민이 승리한 경우이다. 나머지 상태는 유효하지 않은 상태로 간주한다.

		$A^+B^+C^+D^+$			

Label	ABCD	XY=00	XY=01	XY=10	Light	Red	Green
MafWin	0000	1111	1111	1111	0	11	0000
N 1:2	0001	1001	0000	0001	0	10	1100
N 1:3	0010	1010	1001	0010	0	10	1110
N 1:4	0011	1011	1010	0011	0	10	1111
CitWin	0100	1111	1111	1111	0	00	1111
Invalid	0101	X	X	X	X	X	X
N 2:3	0110	1110	0000	0110	0	11	1110
N 2:4	0111	1111	1110	0111	0	11	1111
Invalid	1000	X	X	X	X	X	X
D 1:2	1001	0001	0000	0100	1	10	1100
D 1:3	1010	0010	0001	0100	1	10	1110
D 1:4	1011	0011	0010	0100	1	10	1111
Invalid	1100	X	X	X	X	X	X
Invalid	1101	X	X	X	X	X	X
D 2:3	1110	0110	0000	0010	1	11	1110
D 2:4	1111	0111	0110	0011	1	11	1111

표 2. 무어 머신 상태전이표

위의 상태전이표를 바탕으로 다음과 같은 과정을 거쳐 식을 간단화하였다.

1) XY = 00일 때

Present State				Next State				J				K			
A	B	C	D	A^+	B^+	C^+	D^+	J_A	J_B	J_C	J_D	K_A	K_B	K_C	K_D
0	0	0	0	1	1	1	1	1	1	1	1	X	X	X	X
0	0	0	1	1	0	0	1	1	0	0	X	X	X	X	0
0	0	1	0	1	0	1	0	1	0	X	0	X	X	0	X
0	0	1	1	1	0	1	1	1	0	X	X	X	X	0	0
0	1	0	0	1	1	1	1	1	X	1	1	X	0	X	X
0	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
0	1	1	0	1	1	1	0	1	X	X	0	X	0	0	X
0	1	1	1	1	1	1	1	1	X	X	X	X	0	0	0
1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X

1	0	0	1	0	0	0	1	X	0	0	X	1	X	X	0
1	0	1	0	0	0	1	0	X	0	X	0	1	X	0	X
1	0	1	1	0	0	1	1	X	0	X	X	1	X	0	0
1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	0	1	1	0	X	X	X	0	1	0	0	X
1	1	1	1	0	1	1	1	X	X	X	X	1	0	0	0

$\mathrm{J_A} = 1,\ K_A = 1, \mathrm{K_B} = 0, K_C = 0, K_D = 0$

AB | CD	00	01	11	10
00	1	1	1	1
01	1	X	1	1
11	X	X	X	X
10	X	X	X	X

$\mathrm{J_A} = 1$

AB | CD	00	01	11	10
00	1	0	0	0
01	X	X	X	X
11	X	X	X	X
10	X	0	0	0

$\mathrm{J_B} = C'D'$

AB | CD	00	01	11	10
00	1	0	X	X
01	1	X	X	X
11	X	X	X	X
10	X	0	X	X

$\mathrm{J_C} = D'$

AB | CD	00	01	11	10
00	1	X	0	X
01	1	X	0	X
11	X	X	0	X
10	X	X	0	X

$\mathrm{J_D} = \mathrm{C}'$

이를 정리하면 다음과 같은 식을 얻는다.

XY=00	
$J_A = 1$ $J_B = C'D'$ $J_C = D'$ $J_D = D'$	$K_A = 1$ $K_B = 0$ $K_C = 0$ $K_D = 0$

2) XY=01일 때

Present State				Next State				J				K			
A	B	C	D	A^+	B^+	C^+	D^+	J_A	J_B	J_C	J_D	K_A	K_B	K_C	K_D
0	0	0	0	1	1	1	1	1	1	1	1	X	X	X	X
0	0	0	1	0	0	0	0	0	0	0	X	X	X	X	1
0	0	1	0	1	0	0	1	1	0	X	1	X	X	1	X
0	0	1	1	1	0	1	0	1	0	X	X	X	X	0	1
0	1	0	0	1	1	1	1	1	X	1	1	X	0	X	X
0	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
0	1	1	0	0	0	0	0	0	X	X	0	X	1	1	X
0	1	1	1	1	1	1	0	1	X	X	X	X	0	0	1
1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	0	0	1	0	0	0	0	X	0	0	X	1	X	X	1
1	0	1	0	0	0	0	1	X	0	1	1	1	X	X	X
1	0	1	1	0	0	1	0	X	0	X	X	1	X	0	1
1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	0	0	0	0	X	X	X	0	1	1	1	X
1	1	1	1	0	1	1	0	X	X	X	X	1	0	0	1

$K_A = 1, K_D = 1$

AB | CD	00	01	11	10
00	1	0	1	1
01	1	X	1	0
11	X	X	X	X
10	X	X	X	X

$$J_A = C'D' + CD + B'D'$$

AB | CD	00	01	11	10

00	1	0	0	0
01	X	X	X	X
11	X	X	X	X
10	X	0	0	0

$J_B = C'D'$

AB | CD	00	01	11	10
00	1	0	X	X
01	1	X	X	X
11	X	X	X	X
10	X	0	X	1

$J_C = D'$

AB | CD	00	01	11	10
00	1	X	X	1
01	1	X	X	0
11	X	X	X	0
10	X	X	X	1

$J_D = B' + C'$

AB | CD	00	01	11	10
00	X	X	X	X
01	0	X	0	1
11	X	X	0	1
10	X	X	X	X

$K_B = CD'$

AB | CD	00	01	11	10
00	X	X	0	1
01	X	X	0	1
11	X	X	0	1
10	X	X	0	X

$K_C = D'$

이를 정리하면 다음과 같은 식을 얻는다.

XY=01

$J_A = C'D' + CD + B'D'$	$K_A = 1$
$\mathrm{J_B} = C'D'$	$\mathrm{K_B} = CD'$
$\mathrm{J_C} = D'$	$\mathrm{K_C} = D'$
$\mathrm{J_D} = B' + C'$	$K_D = 1$

3) XY=10일 때

Present State				Next State				J				K			
A	B	C	D	A^+	B^+	C^+	D^+	$\mathrm{J_A}$	$\mathrm{J_B}$	$\mathrm{J_C}$	$\mathrm{J_D}$	$\mathrm{K_A}$	$\mathrm{K_B}$	$\mathrm{K_C}$	$\mathrm{K_D}$
0	0	0	0	1	1	1	1	1	1	1	1	X	X	X	X
0	0	0	1	0	0	0	1	0	0	0	X	X	X	X	0
0	0	1	0	0	0	1	0	0	0	X	0	X	X	0	X
0	0	1	1	0	0	1	1	0	0	X	X	X	X	0	0
0	1	0	0	1	1	1	1	1	X	1	1	X	0	X	X
0	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
0	1	1	0	0	1	1	0	0	X	X	0	X	0	0	X
0	1	1	1	0	1	1	1	0	X	X	X	X	0	0	0
1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	0	0	1	0	1	0	0	X	1	0	X	1	X	X	1
1	0	1	0	0	1	0	0	X	1	X	0	1	X	1	X
1	0	1	1	0	1	0	0	X	1	X	X	1	X	1	1
1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	0	0	1	0	X	X	X	0	1	1	0	X
1	1	1	1	0	0	1	1	X	X	X	X	1	1	0	0

$K_\mathrm{A} = 1$

AB | CD	00	01	11	10
00	1	0	0	0
01	1	X	0	0
11	X	X	X	X
10	X	X	X	X

$J_A = C'\mathrm{D}'$

AB | CD	00	01	11	10
00	1	0	0	0

01	X	X	X	X
11	X	X	X	X
10	X	1	1	1

$$J_B = A + C'D'$$

AB | CD	00	01	11	10
00	1	0	X	X
01	1	X	X	X
11	X	X	X	X
10	X	0	X	X

$$J_C = D'$$

AB | CD	00	01	11	10
00	1	X	X	0
01	1	X	X	0
11	X	X	X	0
10	X	X	X	0

$$J_D = C'$$

AB | CD	00	01	11	10
00	X	X	X	X
01	0	X	0	0
11	X	X	1	1
10	X	X	X	X

$$K_B = A$$

AB | CD	00	01	11	10
00	X	X	0	0
01	X	X	0	0
11	X	X	0	0
10	X	X	1	1

$$K_C = AB'$$

AB | CD	00	01	11	10
00	X	0	0	X
01	X	X	0	X
11	X	1	1	X
10	X	X	0	X

$$\mathrm{K}_D = AB$$

이를 정리하면 다음과 같은 식을 얻는다.

XY=10	
$J_A = C'D'$	$K_A = 1$
$\mathrm{J_B} = A + C'D'$	$\mathrm{K_B} = A$
$\mathrm{J_C} = D'$	$\mathrm{K_C} = AB'$
$\mathrm{J_D} = C'$	$K_D = AB$

따라서 간단화한 전체식은 다음과 같으며,

XY=00		XY=01	
$\mathrm{J_A} = 1$	$K_A = 1$	$J_A = C'D' + CD + B'D'$	$K_A = 1$
$\mathrm{J_B} = C'D'$	$\mathrm{K_B} = 0$	$\mathrm{J_B} = C'D'$	$\mathrm{K_B} = CD'$
$\mathrm{J_C} = D'$	$\mathrm{K_C} = 0$	$\mathrm{J_C} = D'$	$\mathrm{K_C} = D'$
$\mathrm{J_D} = C'$	$K_D = 0$	$\mathrm{J_D} = B' + C'$	$K_D = 1$
XY=10		XY=11	
$J_A = C'D'$	$K_A = 1$	$\mathrm{J_A} = 0$	$K_A = 0$
$\mathrm{J_B} = A + C'D'$	$\mathrm{K_B} = A$	$\mathrm{J_B} = 0$	$\mathrm{K_B} = 0$
$\mathrm{J_C} = D'$	$\mathrm{K_C} = AB'$	$\mathrm{J_C} = 0$	$\mathrm{K_C} = 0$
$\mathrm{J_D} = C'$	$K_D = AB$	$\mathrm{J_D} = 0$	$K_D = 0$

이를 X와 Y에 대해 다시 간단화하면 다음 식을 얻는다.

$\mathrm{J_A} = X'Y' + C'D' + (CD + B'D')\mathrm{Y}$	$K_A = 1$
$\mathrm{J_B} = C'D' + \mathrm{AX}$	$\mathrm{K_B} = AX + CD'Y$

$\mathrm{J_C} = D'$ $\mathrm{J_D} = C' + B'Y$	$\mathrm{K_C} = AB'X + D'Y$ $K_D = AB'X + Y$

출력은 다음과 같이 간단화할 수 있다.

$\mathrm{Mwin} = \mathrm{A'B'C'D'}$는 마피아가 승리한 상태를 의미한다.

$\mathrm{Cwin} = \mathrm{A'BC'D'}$는 시민이 승리한 상태를 의미한다.

R과 G는 각각 빨간색과 초록색 LED를 의미한다. 인풋에 따른 출력은 다음과 같다.

Light	Red 1	Red 2	Green 1	Green 2	Green 3	Green 4
A	1	B	1	1	C	CD

단, Cwin일 경우 모든 Green 값이 1이 되고, 모든 Red 값이 0이 된다. 반대로 Mwin일 경우 모든 Green 값이 0이 되고, 모든 Red 값이 1이 된다.

6. 부품 및 사용처

FPGA 외에 필요한 부품은 다음과 같다.

- Red LED

  현재 state의 마피아 수를 표시하기 위해서 Red LED를 사용한다.

- Green LED

  현재 state의 시민 수를 표시하기 위해서 Greed LED를 사용한다.

- White LED

  현재 state가 낮인지 밤인지를 표시하기 위해서 White LED를 사용한다.

- Resister

LED에 저항이 필요하다. 저항 계산은 다음과 같다. FPGA의 출력으로 나오는 전압을 측정해본 결과 약 1.5V이었으며, 100 Ω 저항을 이용하여 연결하였다.

7. 세부 구현 및 모듈 설명

Vivado를 이용하여 다음과 같은 모듈을 정의하고 사용하였다.

- Final_project Module

이 모듈은 top level 모듈로, FPGA 버튼과 스위치로 입력을 받고, Seven-Segment Display와 LED 등을 출력한다. 받은 입력을 바탕으로 내부 모듈인 FSM과 Display 등을 실행하며, 현재 State에 해당하는 LED에 불이 들어오도록 전원을 공급한다. Input으로 스위치와 연결된 2bit의 Switch_Input, Clock과 연결된 CLK, Botten Center과 연결된 Enable, 스위치와 연결된 RESET을 받고, Output으로 LED들과 7 segment 출력에 필요한 ssSel, ssDisp를 출력한다.

- FSM Module

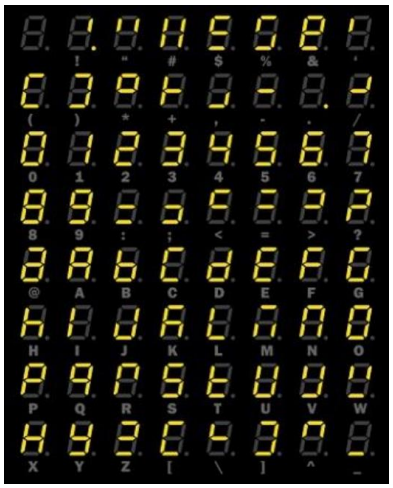
FSM은 Finite State Machine을 의미한다. 이 모듈은 JK 플립플롭을 바탕으로 앞서 설명한 마피아 게임의 State Machine을 구현한다. FSM는 2비트의 입력을 받아 현 상태를 나타내는 4비트의 출력을 반환한다. 또한 clock을 input으로 받게 되는데, 이 clock은 Final_project 모듈에서 한 번의 버튼 클릭에 한 번씩만 state가 변하도록 enable을 clock으로 받아들인다. 이렇게 input으로 받은 enable은 state를 결정하는 JK 플립플롭의 clock으로 들어가 state가 변하게 하는 trigger 역할을 한다. 만약 Reset이 1일 경우 상태가 초기화되어 1111을 출력한다. 즉, 마피아 2명 시민 4명인 낮으로 초기화된다. 그 외의 경우에는 input이 00일 때는 낮 밤만 바뀐다. 그리고 01인 경우에는 시민이 죽어서 시민의 숫자가 줄어든다. 그리고 input이 01일 때는 마피아가 죽어서 마피아의 숫자가 줄어든다. 단, 밤에 01이 input으로 들어올 때는 state가 변하지 않고 유지된다. 마지막으로 11 input은 don't care가 되도록 플립플롭의 J, K 값을 앞서 설명한 대로 구현하여 input으로 넣어주었다.

- Display Module

Display 모듈은 각 상황에 알맞은 출력을 7 Segment Display를 통해 출력한다. 예를 들어 마피아가 이긴 경우에는 MAFIA WIN, 시민이 이긴 경우에는 CITIZEN WIN, 유효하지 않은 입력이 주어진 경우에는 INVALID를 화면에 순차적으로 나타낸다. 이외의 경우에는 남은 시민의 명수와 마피아의 명수를 출력하도록 한다. 상태마다 출력해야 하는 문자열이 다르므로, FSM에서 상태와 입력을 받아서 출력할 값을 불러와야 한다. 출력할 값은 기본 단위가 8비트이므로 알맞은 멀티플렉서를 정의하여 사용한다. 이를 구현하기 위해 여러 과정을 거치며 내부 모듈을 정의하여 사용한다.

FPGA의 7 Segment Display는 한 번에 네 개의 문자를 다 출력하는 것을 지원하지 않는다. 따라서 사람의 눈에 보이지 않을 만큼 빠르게 번갈아 출력하는 일이 필요하다. 따라서 1110->1101->1011->0111->1110... 의 ring 카운터를 사용하여 출력하는 자리가 바뀔 때마다 출력하는 글자가 바뀌도록 해야 한다. MUX에 ring 카운터를 select로 주어 ssDisp의 값으로 ssDispBuf0~3이 번갈아가며 선택되도록 한다.

각 알파벳과 숫자 출력형식은 다음 그림을 따른다.



- DisplayScore Module

현재 남아있는 "마피아 : 시민"의 숫자를 출력해주는 module이다. Top module인 Final_project에서 현재 State가 C+B'D이면서 Switch input이 11이 아닐때 11111111을, 그 외의 경우에는 00000000 저장하는 Display_Score_Enable을 Input으로 받는다. 또한 State에 따라 점수 출력이 달라지므로 input으로 State와 CLK을 받는다. MUX_dis의 input 값으로 parameter와 Enable을 and 연산한 값을 넣으므로 만약 애초에 Enable이 0일 경우에는 모든 숫자 display에 해당하는 wire가 00000000의 값을 갖는다. MUX_dis를 이용하

여 각 state에 따라 display될 8bit 값을 그대로 갖거나, 혹은 00000000을 갖게 한다. 남은 마피아의 숫자는 2번째 bit(AN1)에, 시민의 숫자는 4번째 bit(AN3)에 출력한다. 따라서 1번째 bit(AN0)과 3번째 bit(AN2)는 항상 글씨가 아무것도 없는 상태를 저장하면 되고, 2번째 bit와 4번째 bit는 각 MUX_dis의 출력값을 OR연산을 통해 각 State에 맞게 저장되게 한다.

- DisplayWinner Module

게임이 끝난 경우 마피아가 승리하였는지, 시민이 승리하였는지를 출력하는 module이다. Top module인 Final_project에서 현재 State가 0000 혹은 0100이면서 Switch input이 11이 아닐 때 11111111을, 그 외의 경우에는 00000000 저장하는 Display_Winner_Enable을 Input으로 받는다. 또한 State가 0000일 때와 0100일 때 서로 다른 Winner를 출력해야 하므로 State의 두 번째 bit를 받아온다. 이외에도 CLK과 RESET을 입력 받는다. MUX_dis를 이용하여 B가 1이면 MAFIA WIN 알파벳에 해당하는 wire 값이 00000000이 되게 하고, CITIZEN WIN에 해당하는 wire 값은 parameter 값을 그대로 갖게 한다. B가 0인 경우는 반대로 CITIZEN WIN 알파벳에 해당하는 wire 값은 000000000, MAFIA WIN에 해당하는 wire 값은 parameter 값 그대로를 갖는다. 하지만 MUX_dis의 input 값으로 parameter와 Enable을 and 연산한 값을 넣으므로 만약 애초에 Enable이 0일 경우에는 모든 알파벳에 해당하는 wire가 00000000의 값을 갖는다.

긴 문자열을 출력하기 위해 디스플레이의 출력값을 한 칸씩 이동시키는 과정이 필요하다. FPGA의 클록 속도가 400MHz에 달하므로 사람의 눈으로 움직임을 인지하려면 더 느린 클록을 만드는 것이 필수이다. 이때 $\ln(400 \cdot 10^6) \cong 28.58$이므로 기존 클록보다 2^{28}배 느린 클록이 필요하다. 따라서 JK 플립플롭을 통해 구현한 16진 계수기 여러 개를 연립하여 hexa_8 이라는 module의 Frequency Divider를 제작하였다. hexa_8 counter에서 출력되는 count를 mux의 select로 넣어 각 시간대 별로 7 segment에 display 될 4bit를 선택해준다. 이 4bit는 결국 state가 0000 혹은 0100이 아닌 경우에는 항상 0000을 저장한다.

- DisplayInvalid Module

Switch input으로 11이 들어왔을 때 Invalid input이므로 INVALID를 출력하는 module이다. Top module인 Final_project에서 현재 Switch input이 11인 경우 11111111을, 그 외의

경우에는 00000000 저장하는 Display_Invalid_Enable을 Input으로 받는다. Display될 4bit를 선택하는 방식은 위의 DisplayWinner와 동일하다.

8. 구현 결과 및 고찰

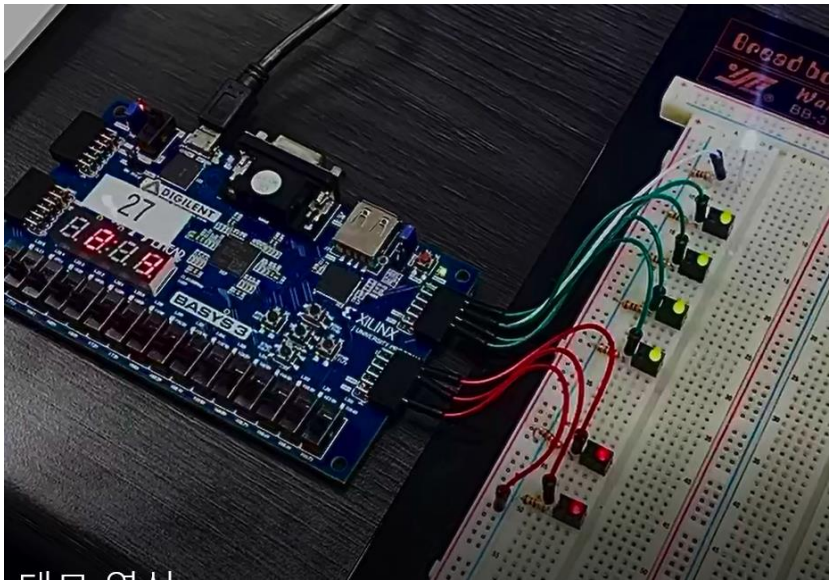


그림 5. FPGA로 구현한 State Machine. 흰색 LED는 낮을, 초록색 LED는 시민 명수를, 빨간색 LED는 마피아 명수를 나타낸다.

위의 기계로 실제 동작을 검사하며 구현이 제대로 이루어졌는지 확인하였으며, 몇 번에 오류 수정을 거쳐 각 입력에 따라 모든 동작이 정상적으로 이루어짐을 확인하였다.

구현 과정에서 if문이나 always문을 쓰지 않고 모든 동작을 구현하여야 했기에 7 Segment Display의 구현이 예상보다 더 오래 걸렸다. 이 과정에서 추가로 Ring 계수기와 Frequency Divider, 8비트 멀티플렉서를 비롯한 여러 모듈을 추가로 구현하였으며, 출력이 예상한 대로 나오지 않는 경우가 잦아서 많은 시행착오를 겪었다. 심지어 테스트 벤치의 시뮬레이션 상으로는 완벽하게 돌아가는 코드도 실제로는 정상 작동하지 않는 경우도 있었다. 하지만 회로가 예상한 대로 작동하지 않아 막혀 있을 때 조교님들께 많은 도움을 받아 문제를 해결할 수 있었다. 이번 프로젝트 내내 랩실에서 상주하며 적극적으로 도움을 주신 조교님들께 감사의 말씀을 전한다.