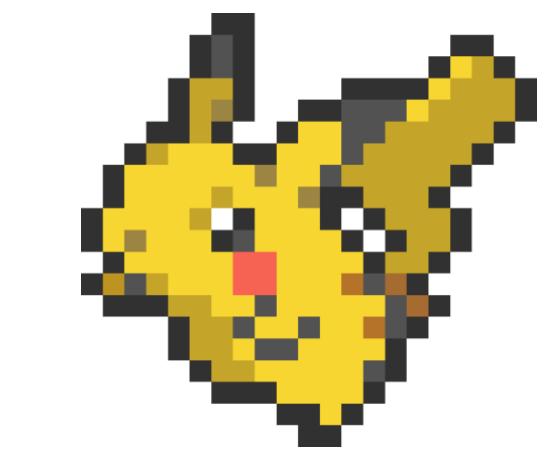




FPGA Real-Time Dice Board Game Based on I2C & VGA



10팀 : 곽서유, 황석현, 한지윤, 진우석

index

- 1. Introduction**
- 2. Game Logic Module (I2C master)**
- 3. Dice Detector & VGA Display Module**
- 4. I2C Slave Module**
- 5. 동작영상 및 고찰 (TroubleShooting / 팀소개)**

01

Introduction

Development Environment

EDA Tool

AMD Xilinx Vivado Design Suite 2020.2



HDL Language

SystemVerilog (IEEE 1800-2017)



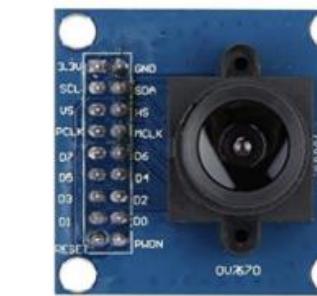
Board

Digilent Basys 3 (100 MHz on-board clock)



Camera

OV7670

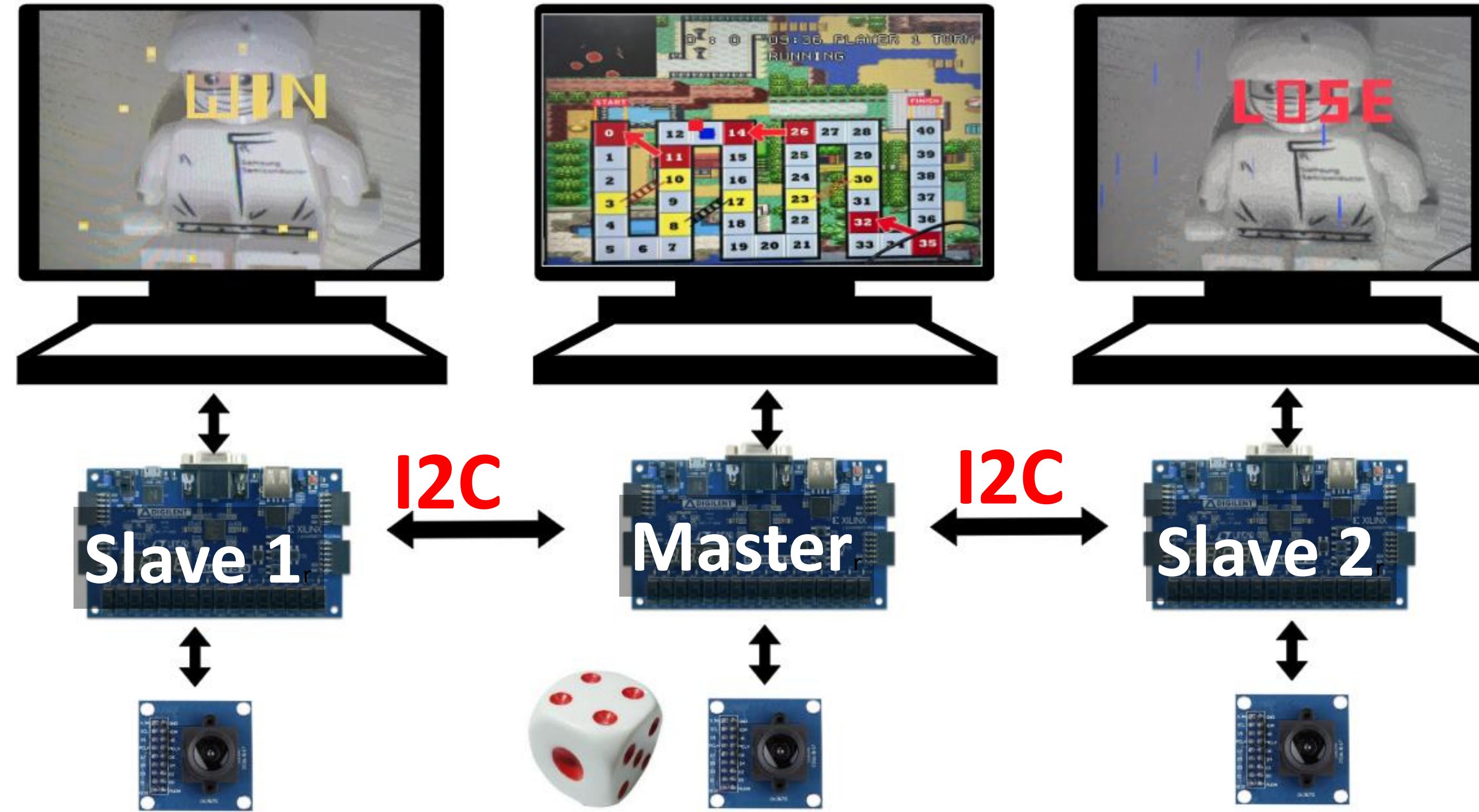


interface

I2C protocol (Master-Slave game data transfer)

SCCB protocol (OV7670 camera register control)

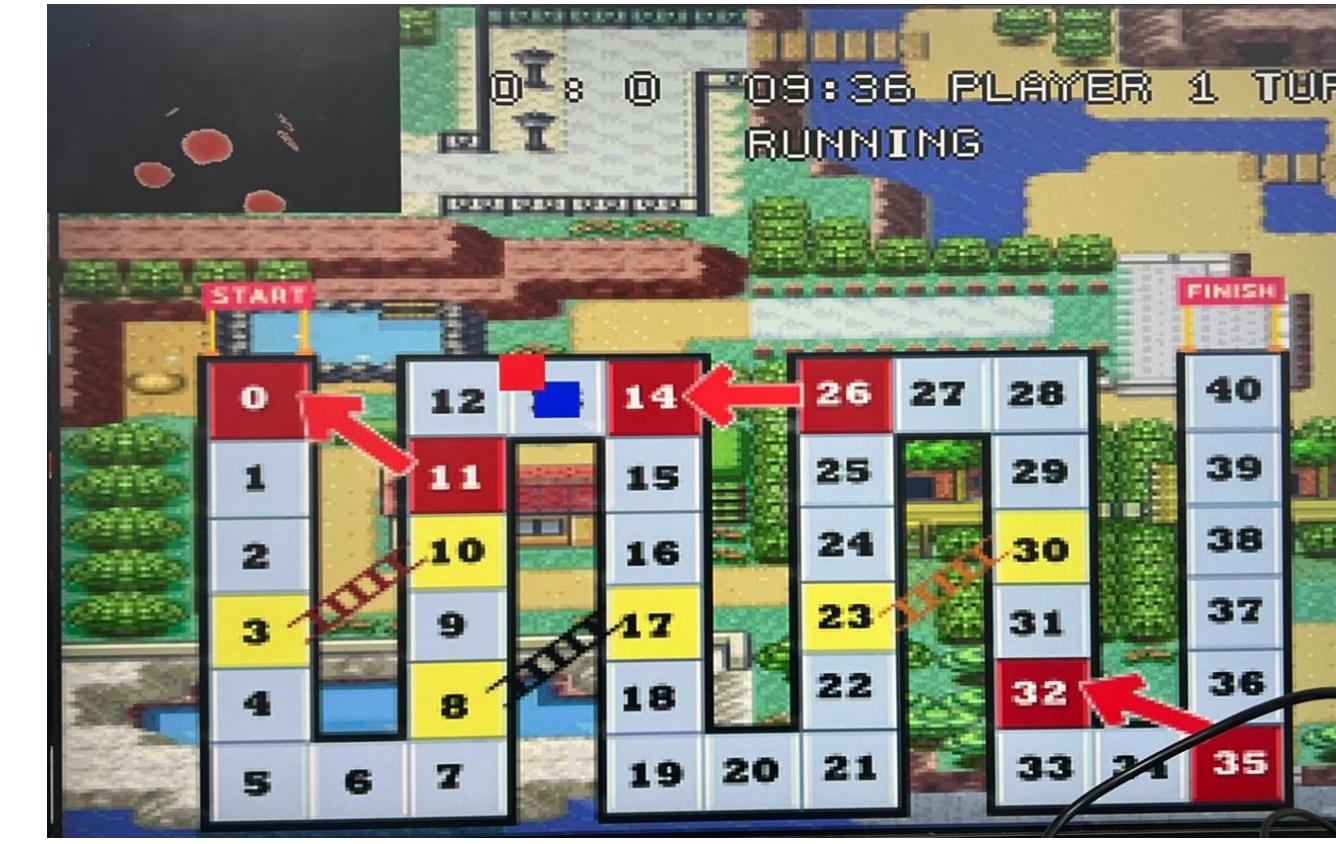
OVERVIEW



주사위를 인식한 값을 기반으로 Master에서 게임 로직을 수행
→ 그 결과를 **I2C 통신**을 통해 2-Player Slave로 전달
→ 실시간 영상 필터와 상태 표현을 적용하는
FPGA 기반 VGA Dice 보드게임

OVERVIEW

Game Rule



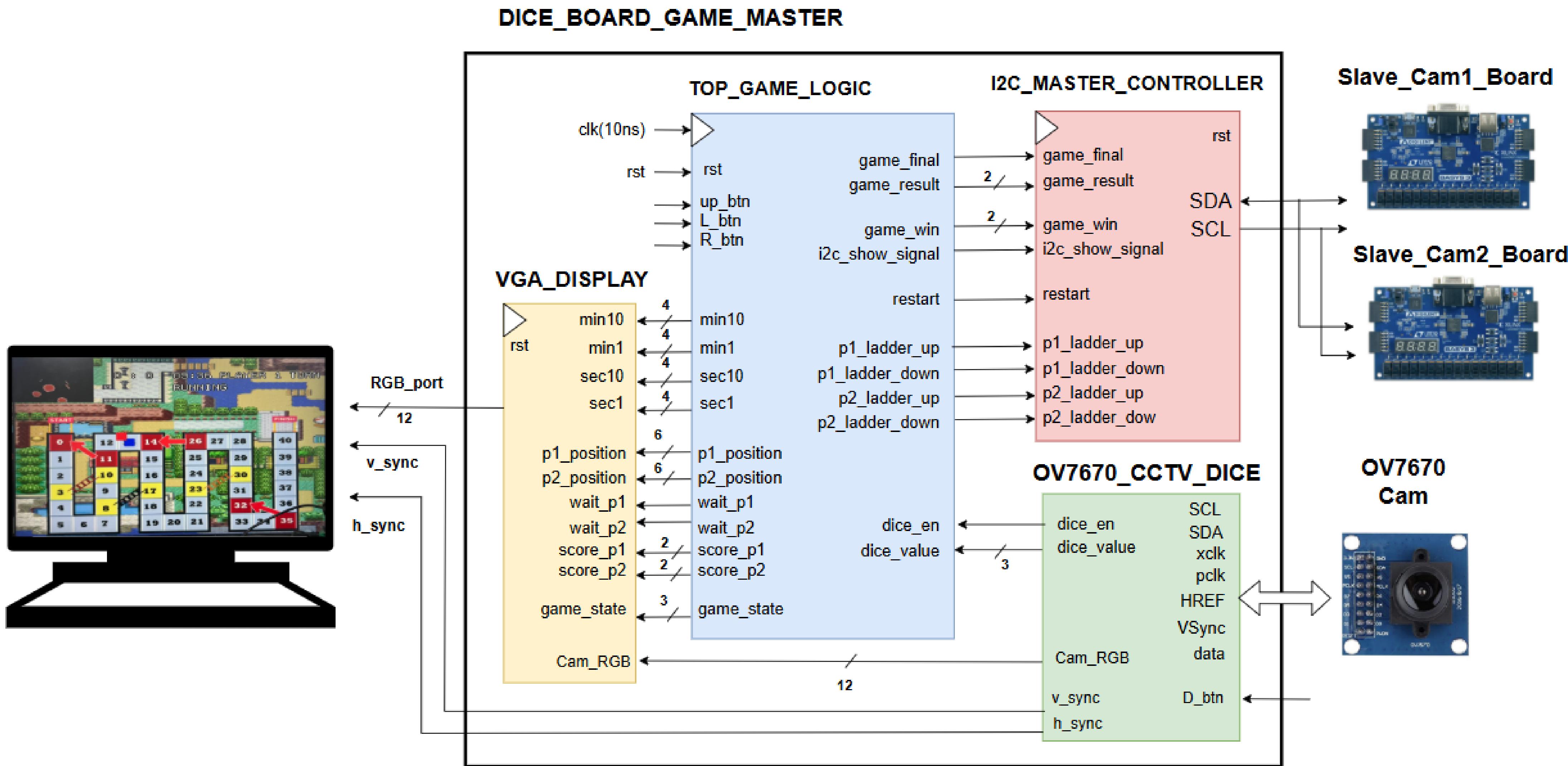
- 주사위 인식 : 플레이어가 굴린 주사위의 빨간 눈금을 시스템이 실시간으로 판독.

- 1) 자동 이동 : 인식된 눈금 수만큼 모니터 상의 말이 자동으로 이동.
- 2) 사다리 : 특정 UP 칸에 도착 시, 사다리를 타고 더 높은 칸으로 전진. + slave 골든 효과 적용.
 : 특정 DOWN 칸에 도착 시, 뱀을 타고 이전 칸으로 후퇴. + slave 모자이크 효과 및 흔들림 적용.

- 점수 획득 및 승리 조건

- Player1 → Player2 순으로 게임진행 : 2명이 다 주사위를 던지면 상태(누가 앞서 있는지 + 누가 게임을 이겼는지)
- 점수 획득 : 말이 Finish Line(40)에 도착하면 라운드가 종료되며 스코어 1점 획득. 이후 시작 지점으로 말이 자동 이동.
- 최종 승리 : 총 2점의 스코어를 먼저 획득하는 플레이어가 승리 (restart 가능)

BLOCK_DIAGRAM (MASTER BOARD)



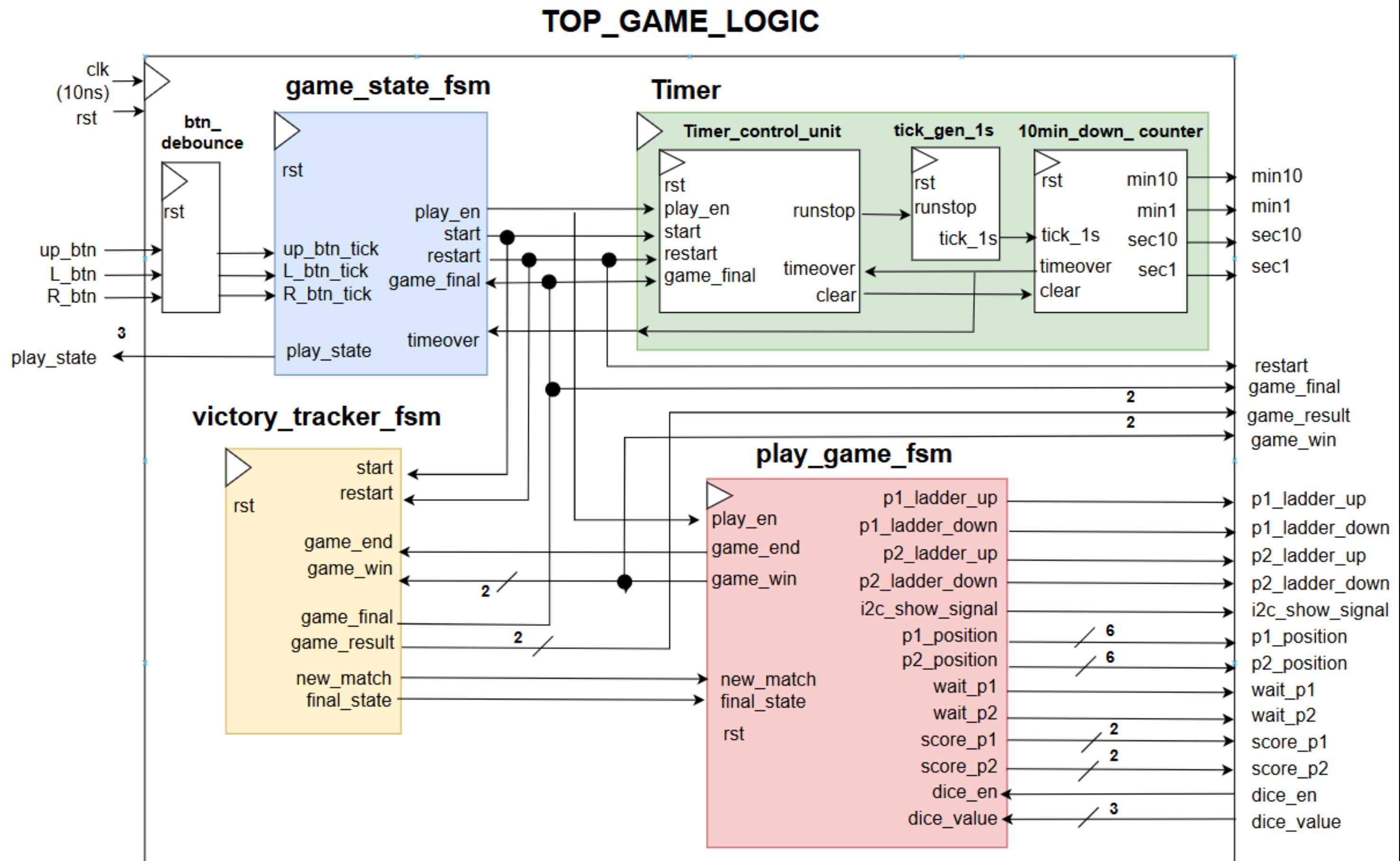
Please enter the title of the presentation

Simple Presentation Design Kit

02

Game Logic / i2c Master Controller

Game_Logic_BLOCK_DIAGRAM



보드의 버튼 입력과 OV7670 DICE 모듈을
통해 들어오는 주사위 정보를 기반으로 동작하는
게임 로직 통합 제어 시스템

구성요소

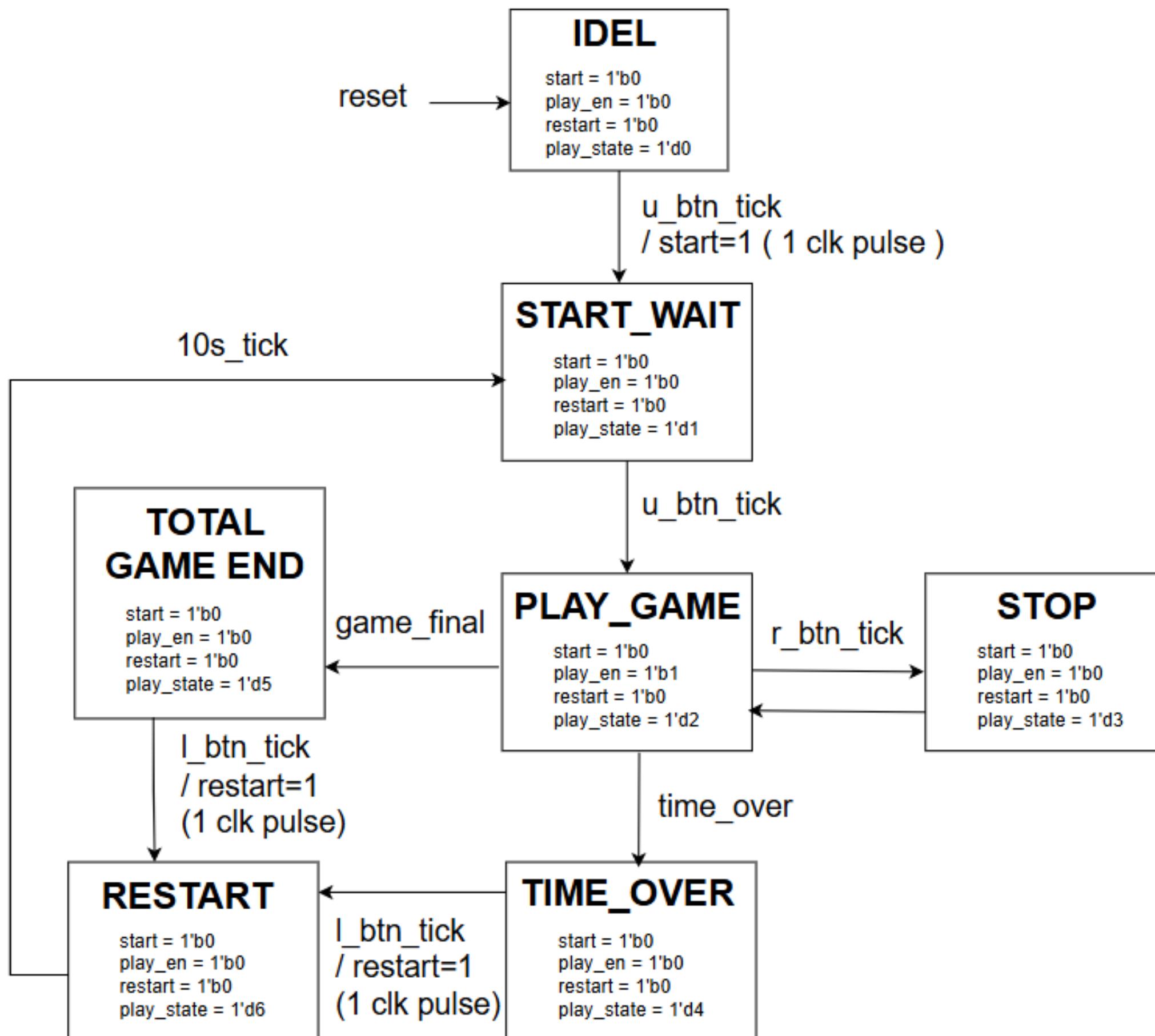
서로 다른 역할을 수행하는
4개의 FSM 모듈로 구성

→ 각 FSM은 서로 input/output 신호를
주고받으며 전체 게임 흐름을 제어

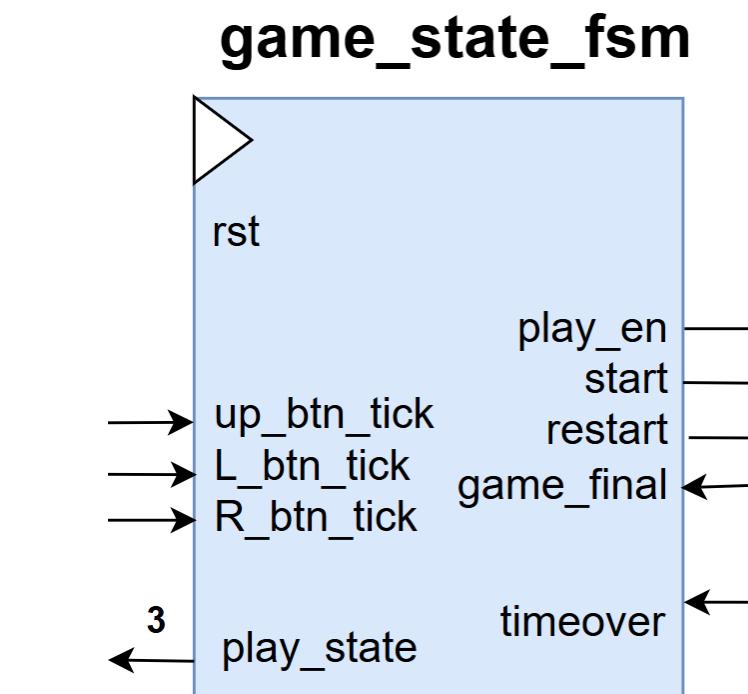
이외 기능

- 1) i2c master controller event 생성
- 2) VGA display를 위한 신호 생성

GAME_STATE

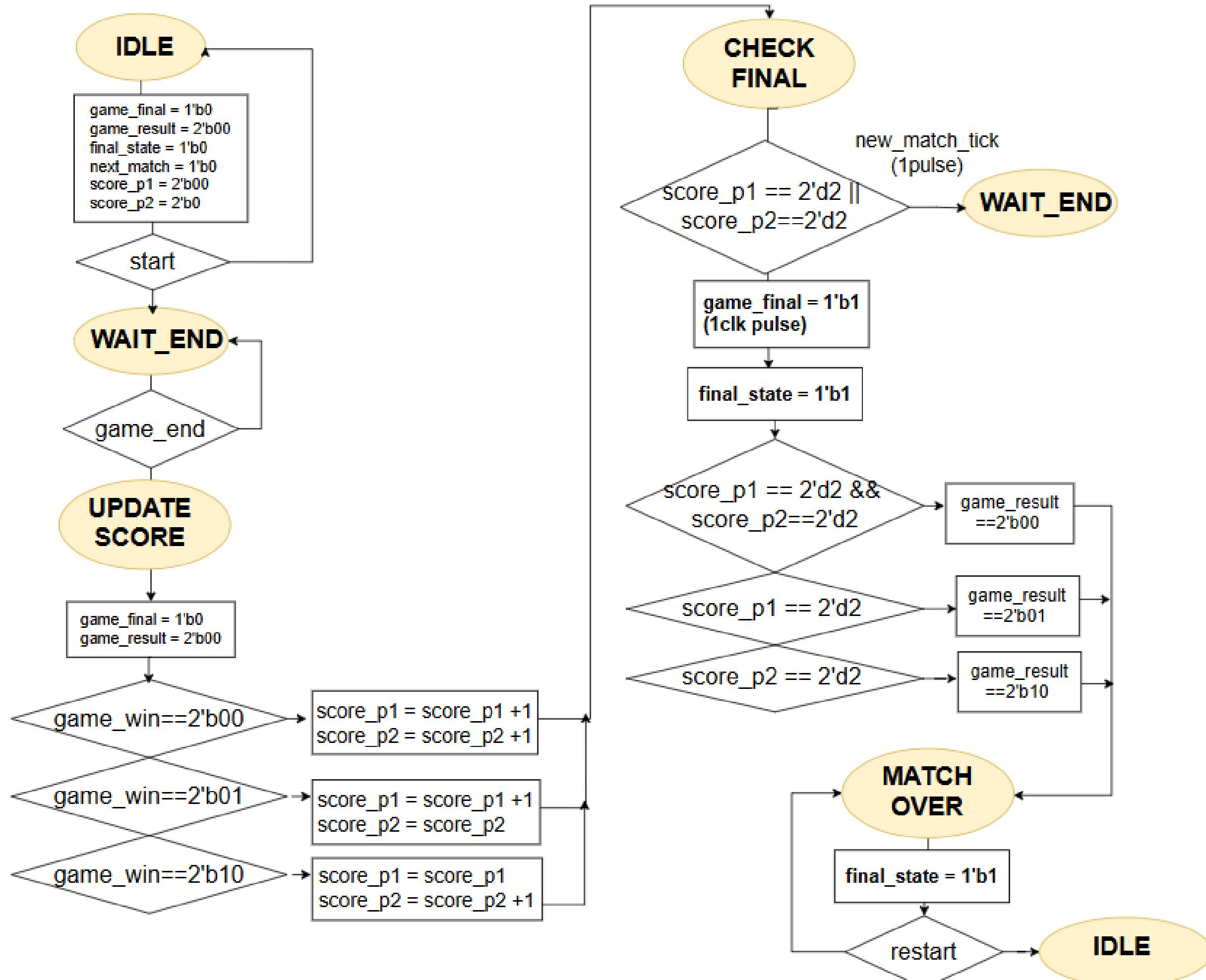


GAME_STATE



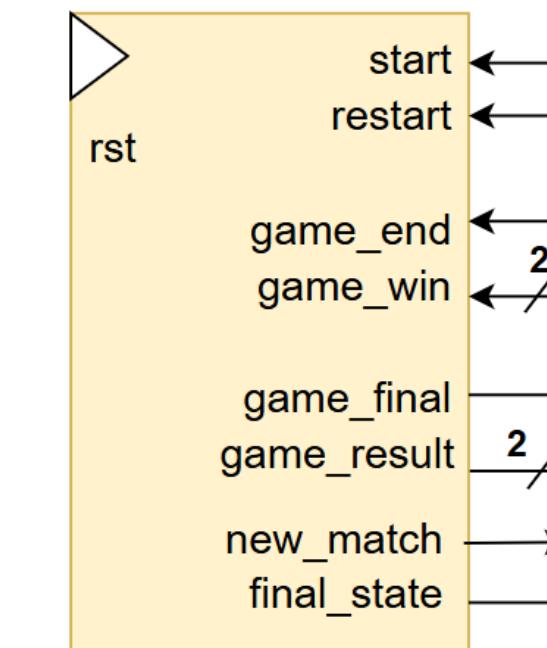
사용자 입력 버튼(게임시작, 멈춤, 다시시작)과
game logic 내부 3판2승제 및 time_over 신호를 기반으로
게임의 진행 상태를 나타내는 모듈

VICTORY_TRACKER



VICTORY_TRACKER

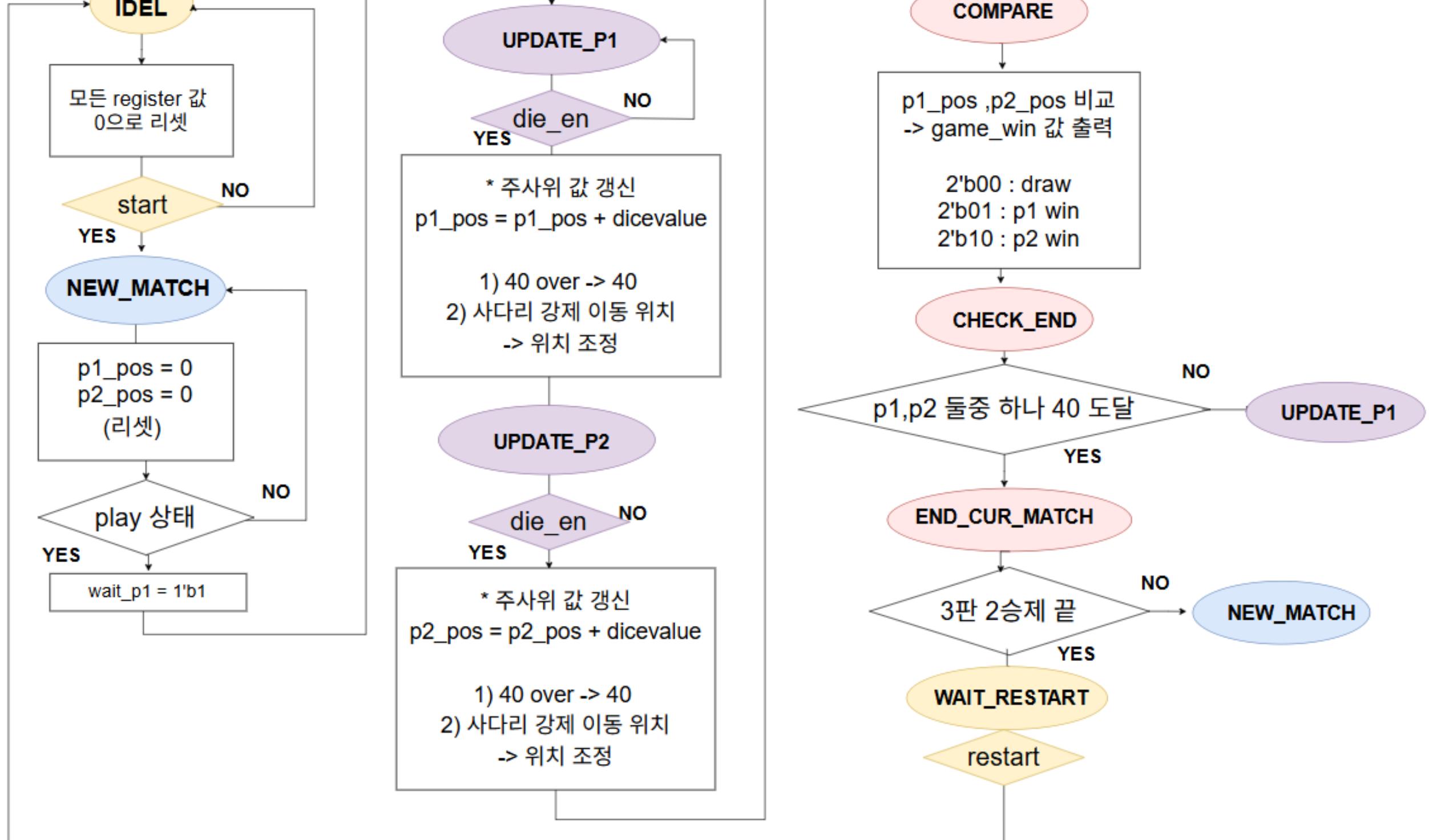
victory_tracker_fsm



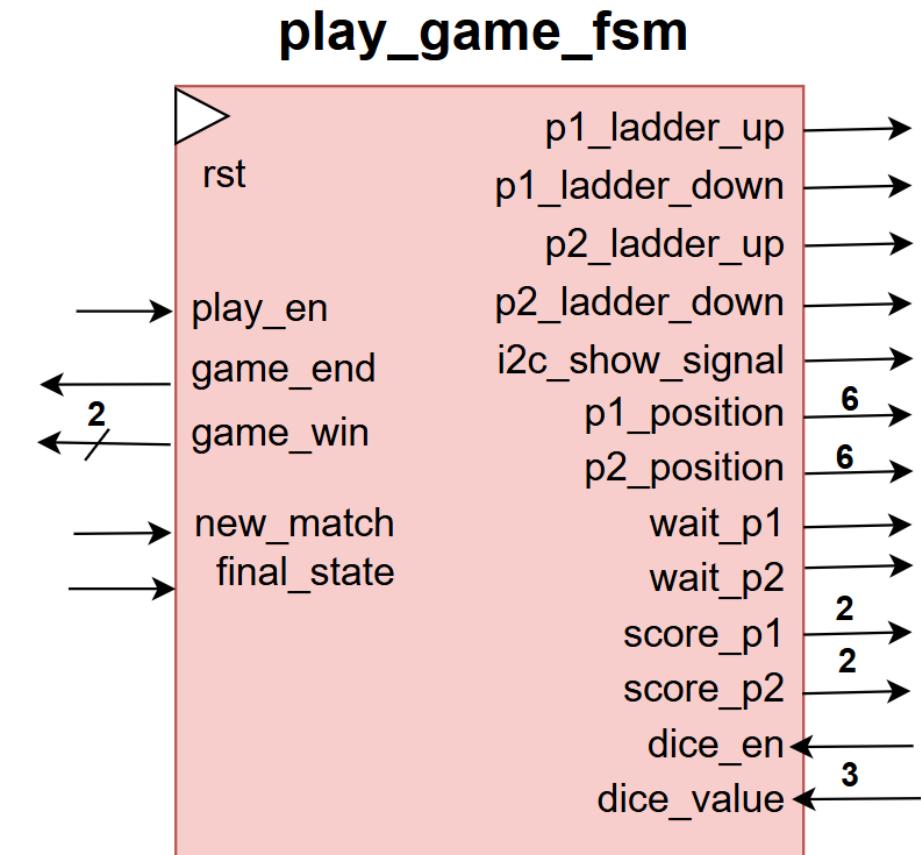
PLAY_GAME 모듈과
GAME_STATE 모듈로 부터 오는 신호를 기반으로

3판 2승제가 끝났는지 확인 및
3판 2승제에 대한 결과 계산을
처리하는 모듈

PLAY_GAME



PLAY_GAME

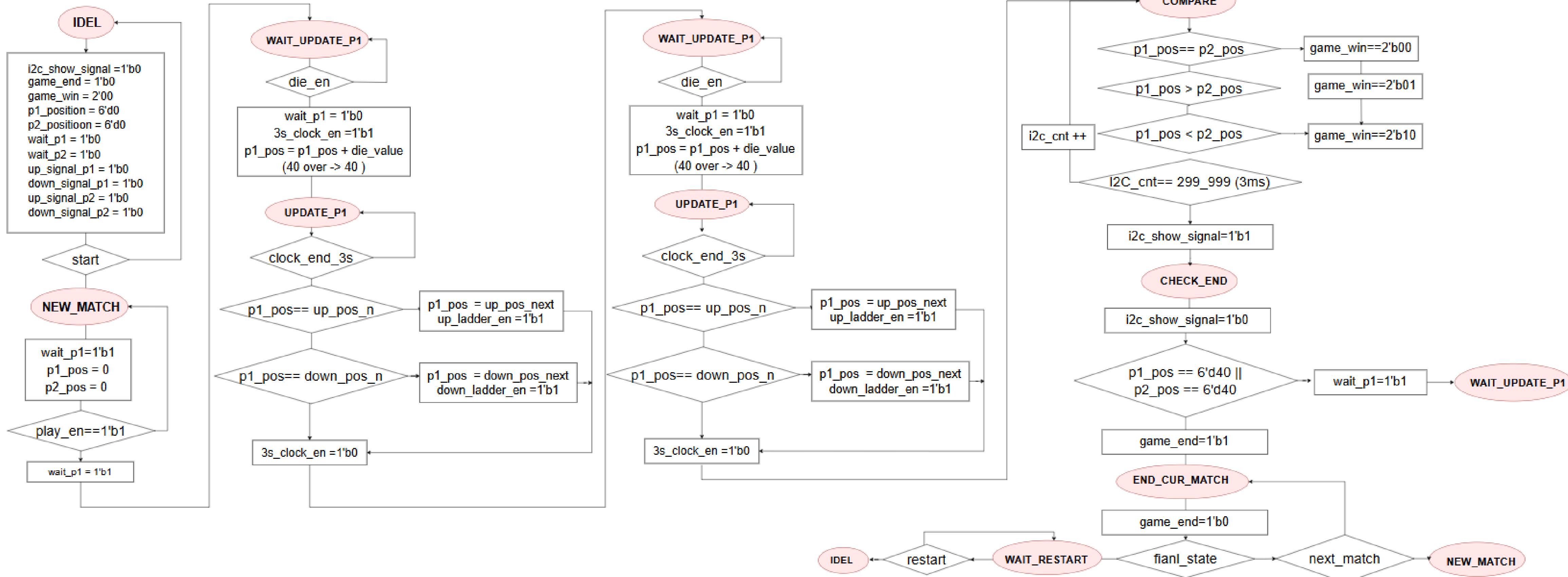


주사위 Cam 입력과
VICTORY_TRACKER로 부터 오는 신호를 기반으로

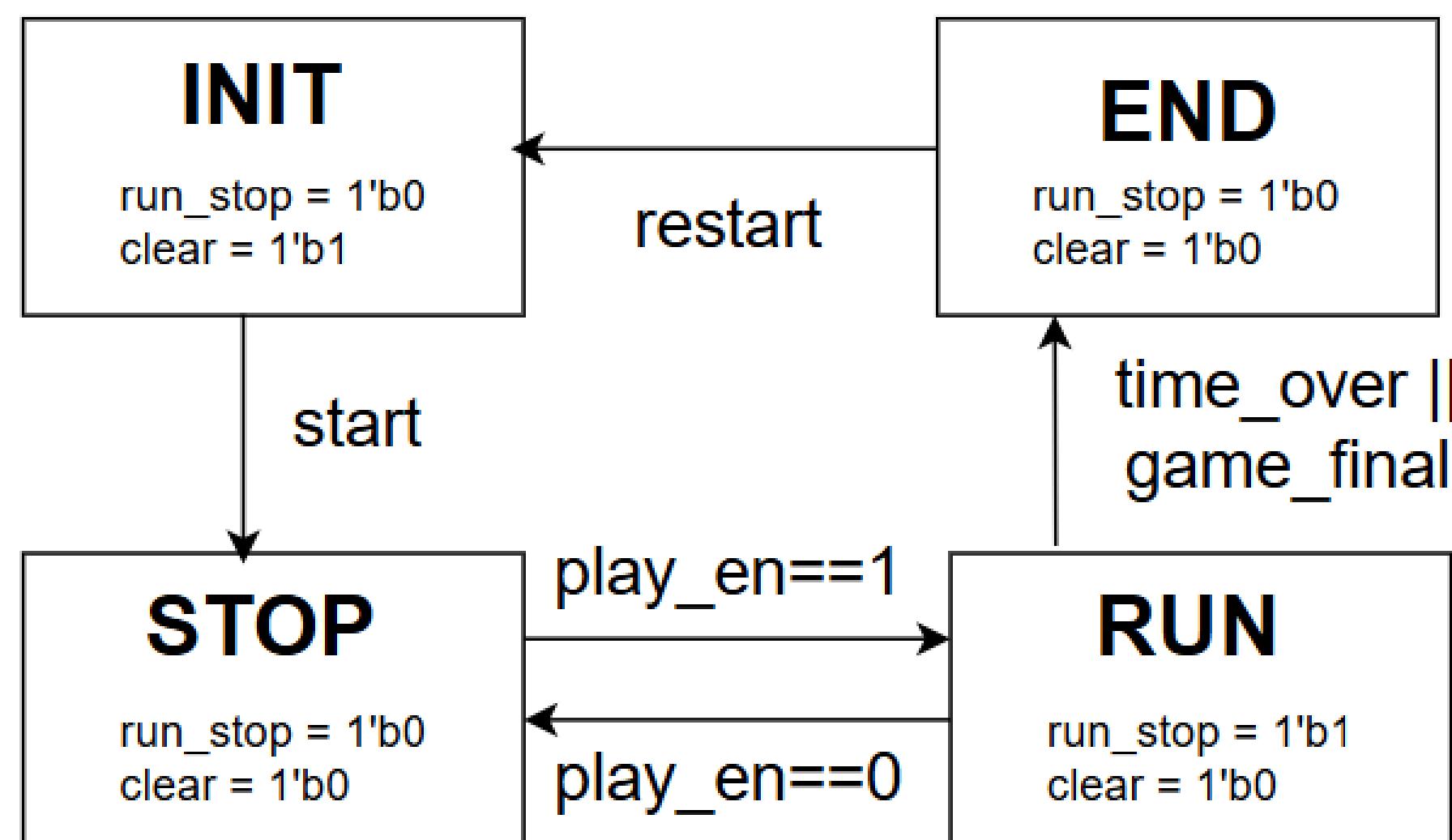
두 플레이어의 위치를 실시간 계산하고,
게임 진행부터 승부 판정까지
한 게임의 흐름을 제어하는 컨트롤 로직

- 주사위 값(dice_value)에 따른 플레이어 위치 업데이트
- 사다리/벌칙 구간에 따른 위치 보정
- 승부 비교 및 game_win 결과 산출

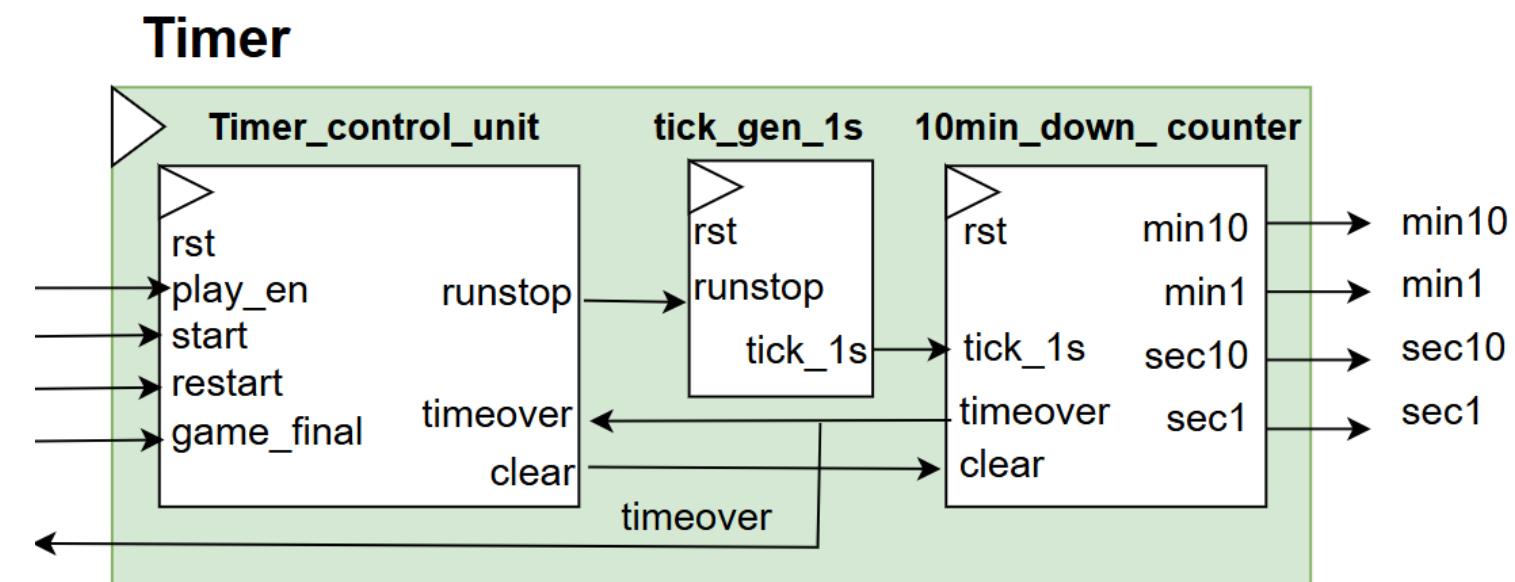
PLAY GAME



TIMER

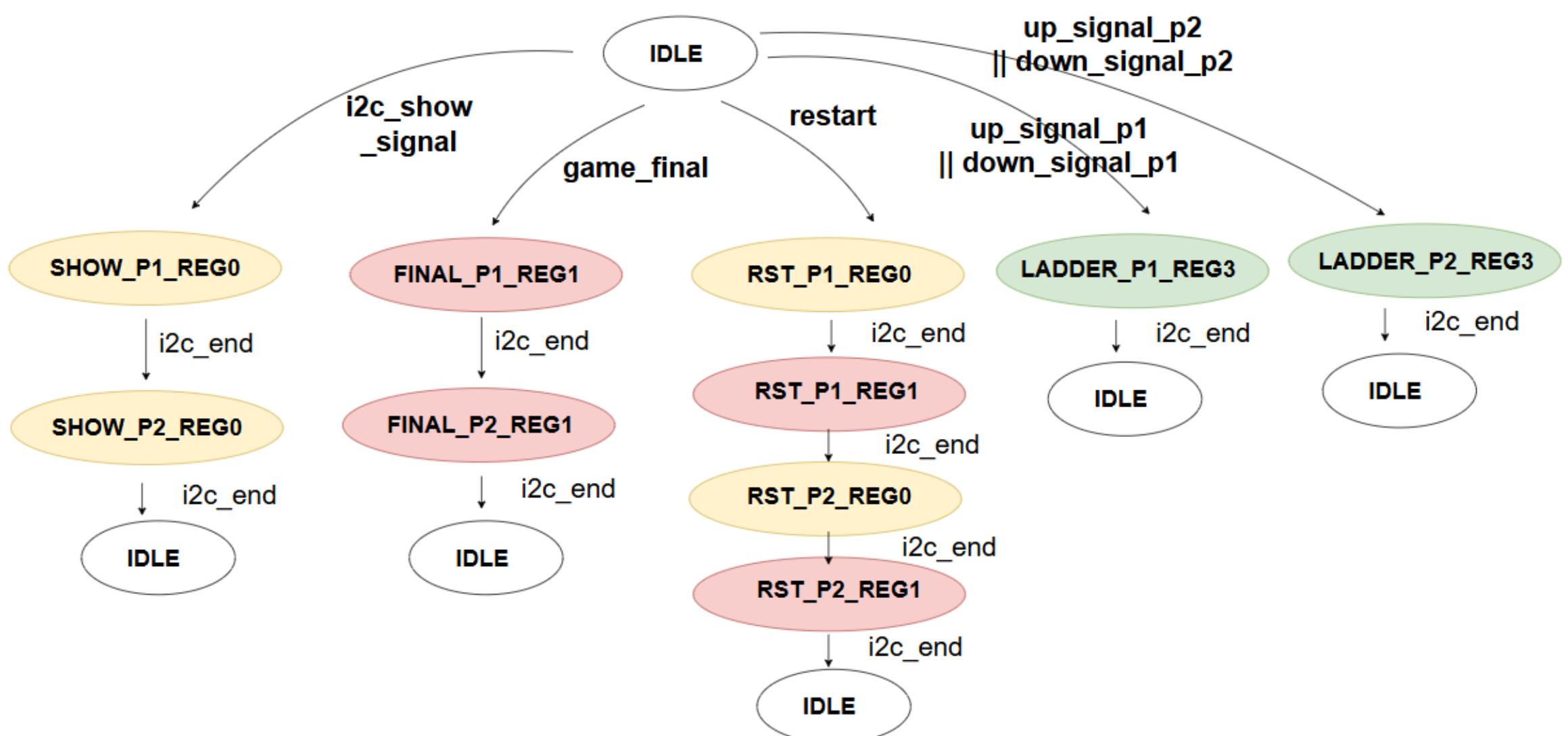
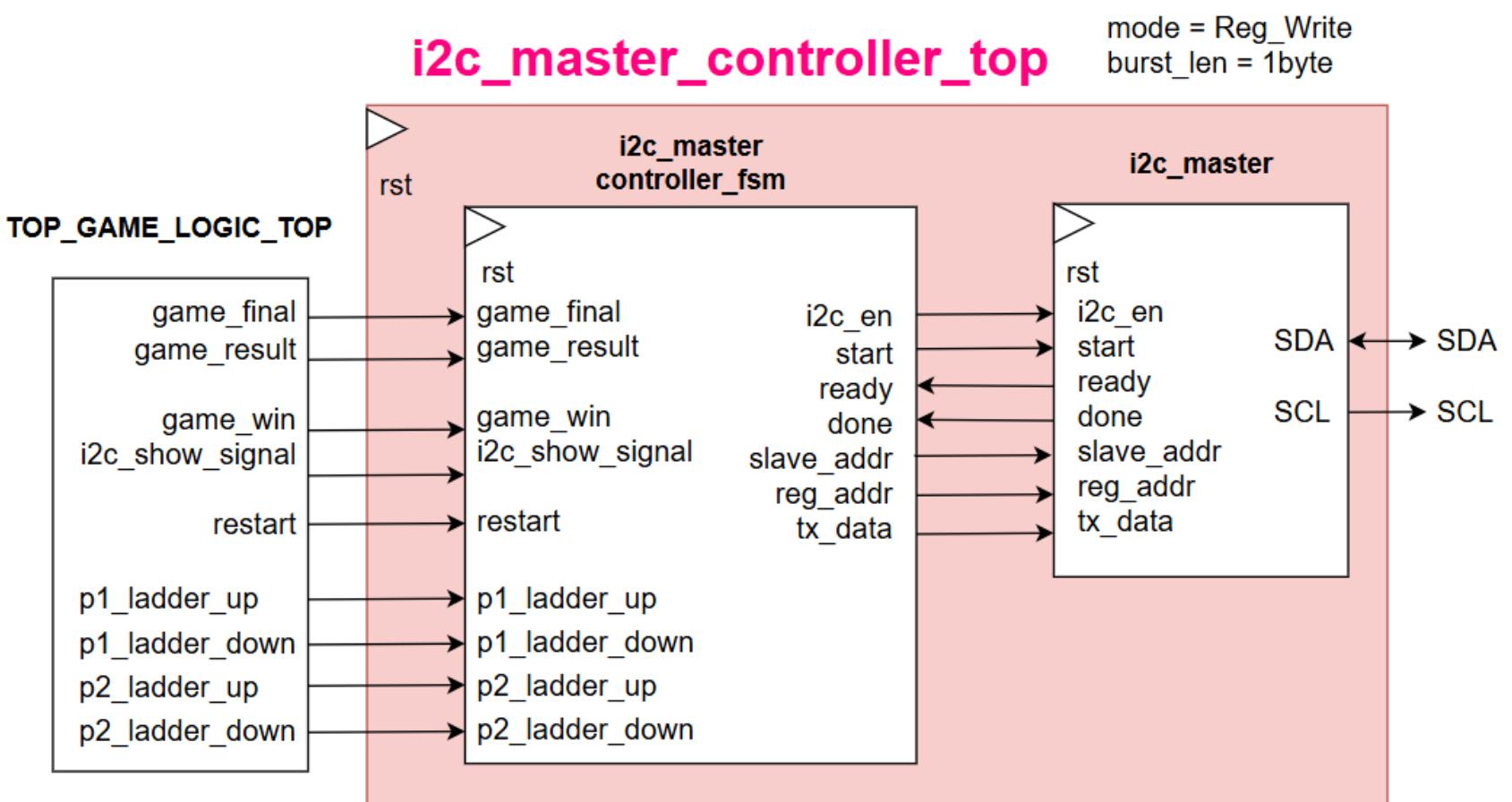


TIMER



TIMER는 게임 시간(10분 카운트다운)을 관리하는 모듈로
게임의 진행 상태(play_en), 시작(start), 재시작(restart),
게임 종료(game_final) 신호에 따라
시간 생성, 카운트, 정지, 초기화 제어

I2C_Master_Controller



I2C MASTER CONTROLLER

게임 FSM에서 발생하는 이벤트
(주사위 표시, 사다리 이동, 게임 종료 등)를

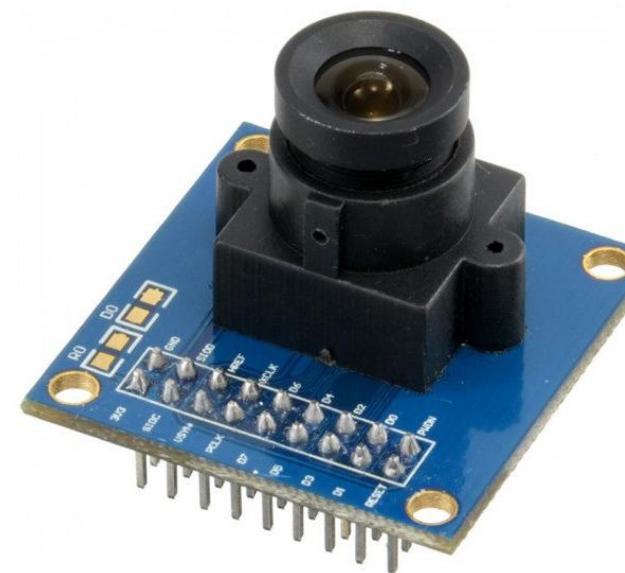
I2C Slave 장치에게 전송하기 위해
설계된 전용 통신 제어 모듈

- PLAYER 1 Slave → PLAYER 2 Slave 순서로 i2c통신
- 현재 누가 이기는 중인지에 대한 정보 : Register 1
- 3판2승제에 대한 정보 : Register 2
- Ladder event에 대한 정보 : Register 3

03

Dice Detector & VGA Display

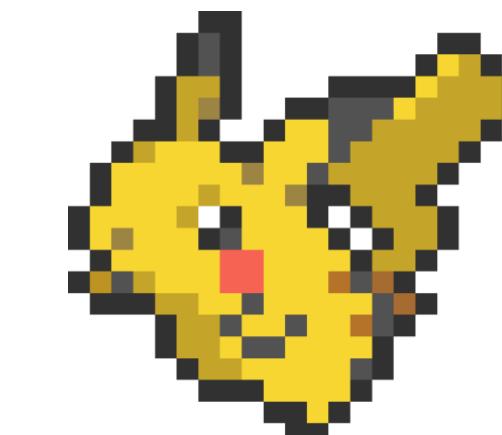
VGA & Dice Detector



01

Camera : OV7670

- Resolution : QQVGA (160 x 120)
- Format : RGB565



02

VGA Display

- Resolution : 640 x 480 @ 60Hz
- Display Area : 화면 좌측 160 x 120 영역에 카메라 영상 출력

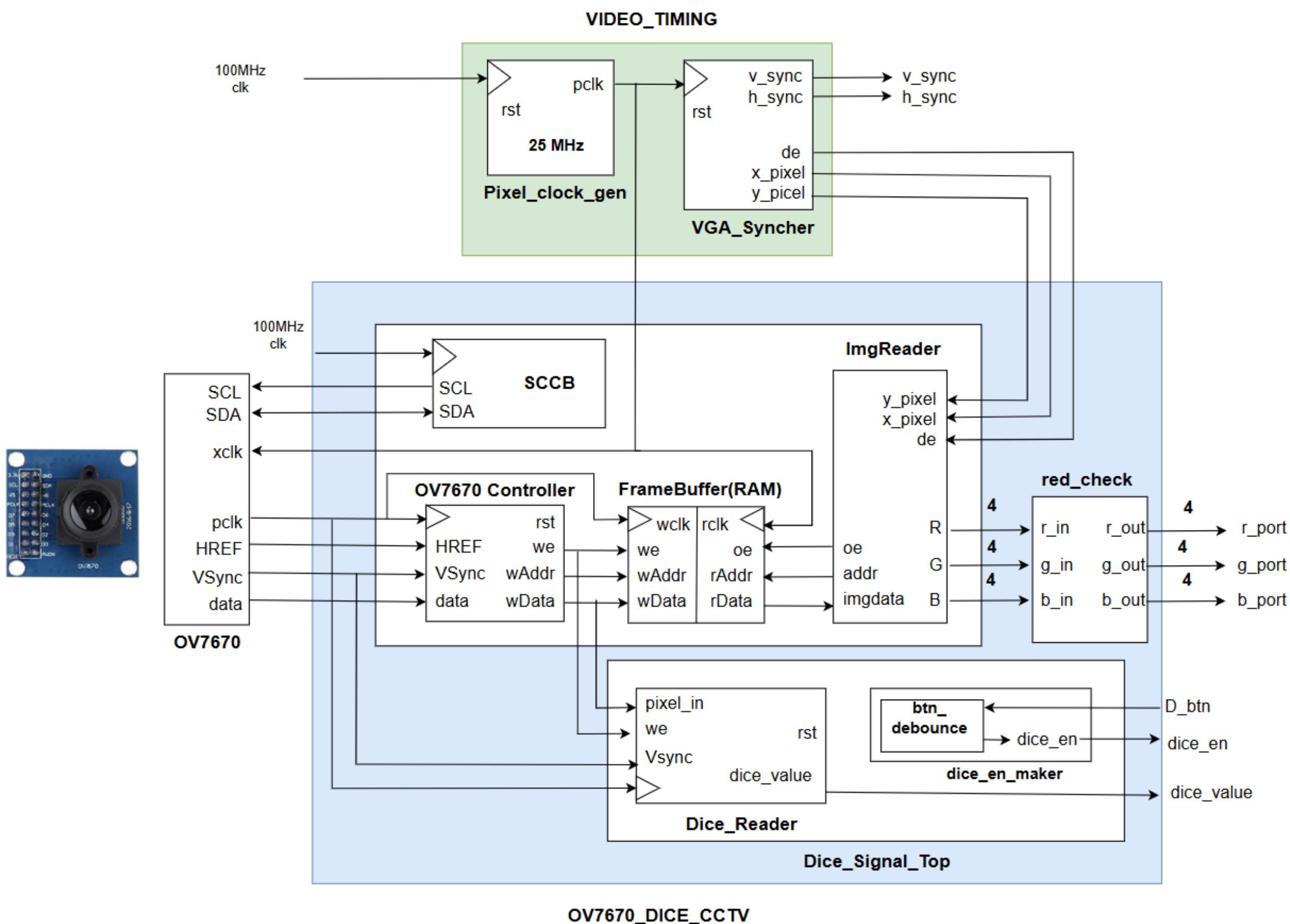


03

Main Function

- 실시간 영상 출력
- **red_check** — 주사위 RGB 중 Red Pixel 만 출력
- **Dice_Reader** — Red Pixel Counting 하여 주사위 눈금 판단

VGA & Dice Detector



OV7670 카메라 영상을 받아
빨간 Pixel 개수를 인식하여 주사위의 눈금을 실시간으로
인식하고 game logic에 신호를 전달
및 모니터에 영상 출력

OV7670 : SCCB 모듈을 통해 초기화되고 QVGA 데이터를 FPGA로 전송.

SCCB : OV7670 camera register 초기화

OV7670 Controller : 카메라의 HREF, VSYNC 타이밍에 맞춰 데이터 캡처.

짝수 pixel만 샘플링하여 데이터 양 1/4로 축소.

FrameBuffer(RAM) : 축소된 QQVGA 이미지를 저장하는 RAM.

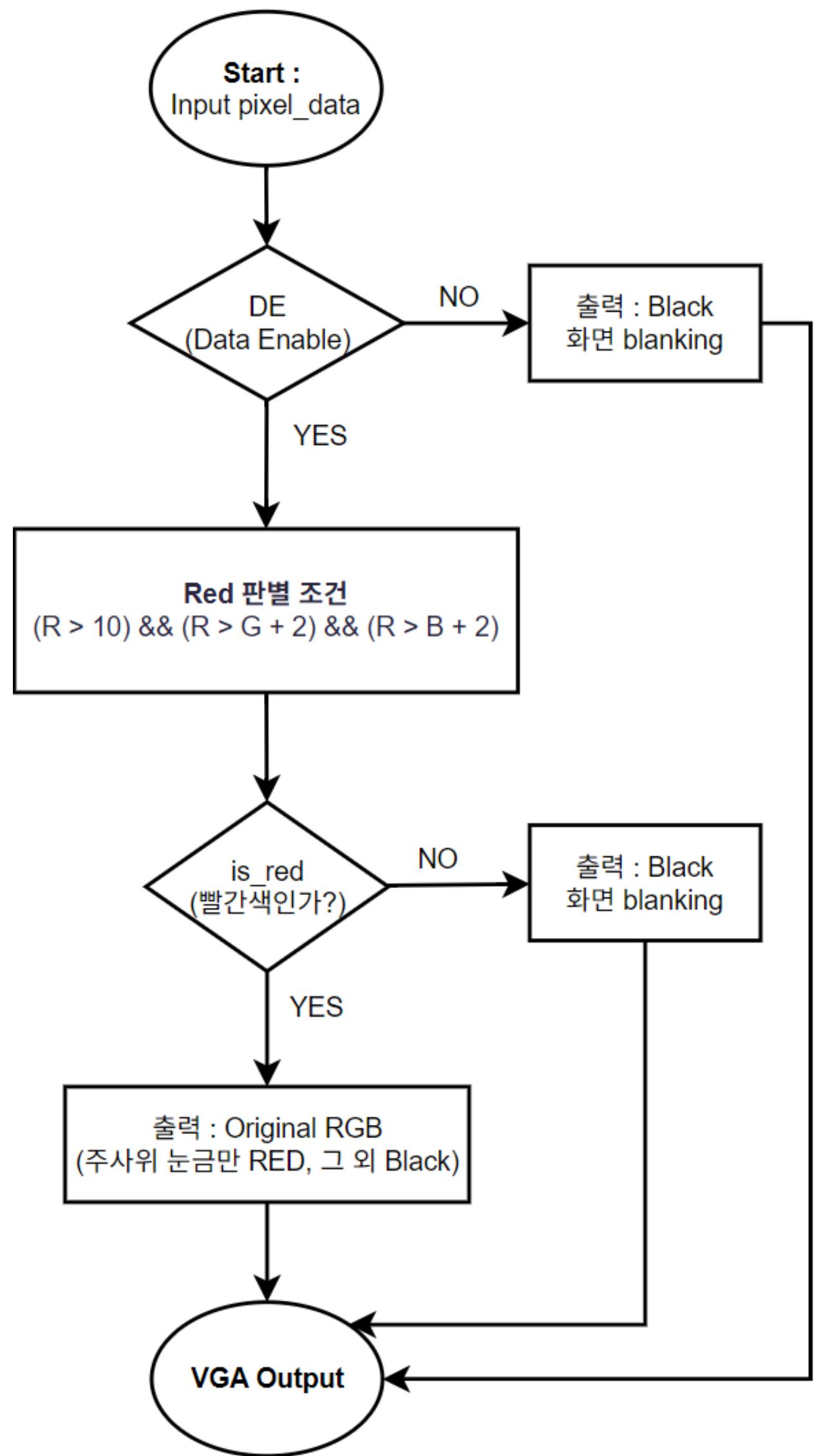
ImgReader : VGA 타이밍(x, y)에 맞춰 RAM 주소를 생성하고 데이터를 읽어옴.

모니터 출력 구간(160x120) 설정.

red_check : 읽어온 pixel[0] 빨간색인지 판별하고 모니터 출력 색상 결정.

Dice_Reader : 빨간색 pixel 수를 한 프레임 동안 카운트하여 주사위 눈금(1~6) 출력.

RED_CHECK



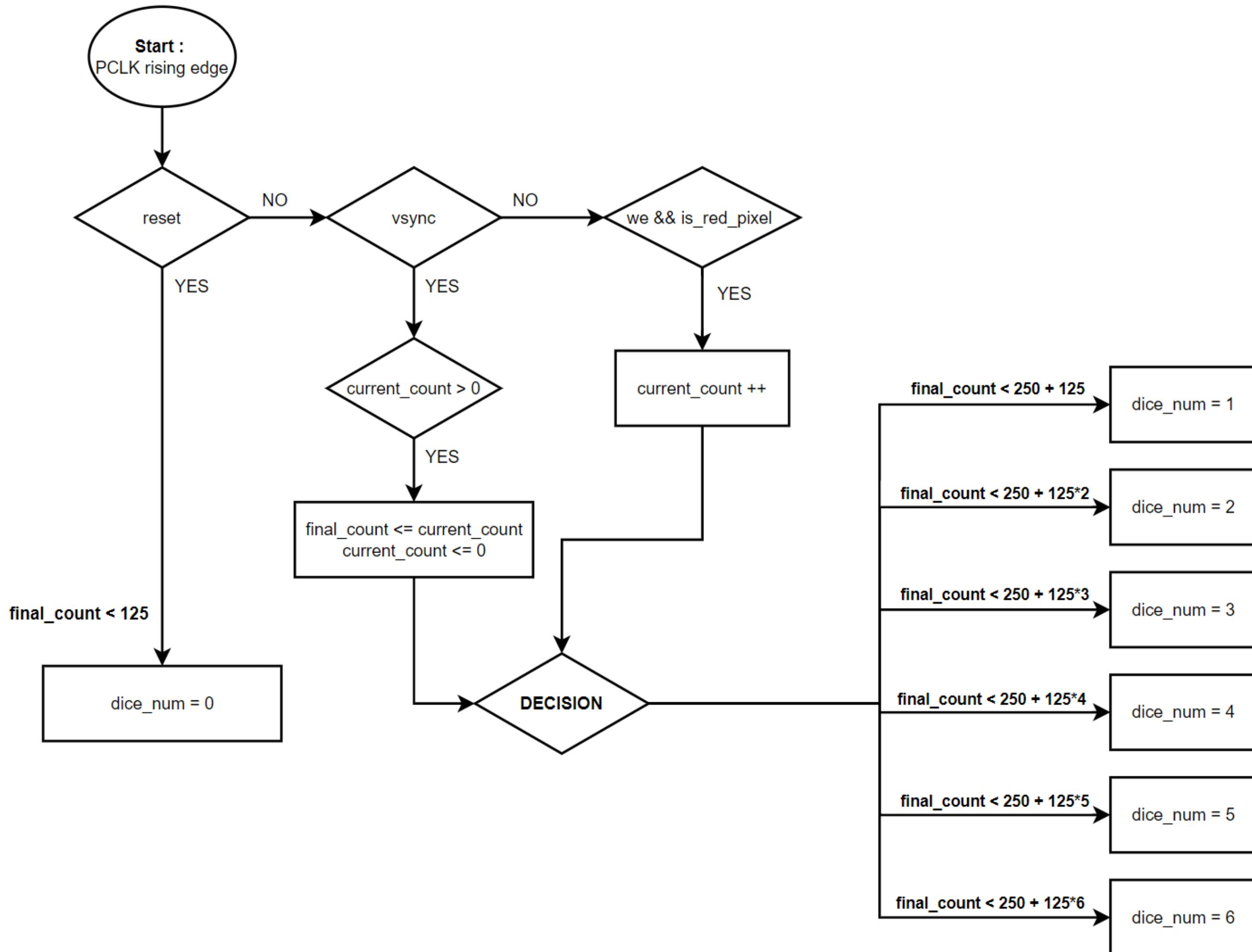
- **역할**

: 모니터에 "무엇을 보여줄지" 결정.

- **로직**

1. DE 확인 : 화면 출력 구간인지 판단하여, 맞으면 화면 출력하고 아니라면 검은색 처리.
2. Red 판별 : $(R > G) \text{ && } (R > B)$ 조건으로 빨간색인지 확인.
3. 출력 : 빨간색이면 원본 색상 출력, 아니면 검은색으로 Masking

DICE_READER



- **역할**

: 주사위 눈금을 "어떻게 셀지" 결정.

- **로직**

1. Pixel이 들어올 때마다 빨간색이면 Count + 1.
2. VSYNC 타이밍(프레임 끝)에 현재 카운트 값 저장.
이때 count > 0 조건 확인해 노이즈 필터링.
3. 출력 : 최종 카운트 값을 125 단위로 조건을 나누어 주사위 눈금 판별.

VGA_DISPLAY

Dice
Detector
출력영상



SCORE상태



Timer
(10분제약)

09:36
RUNNING

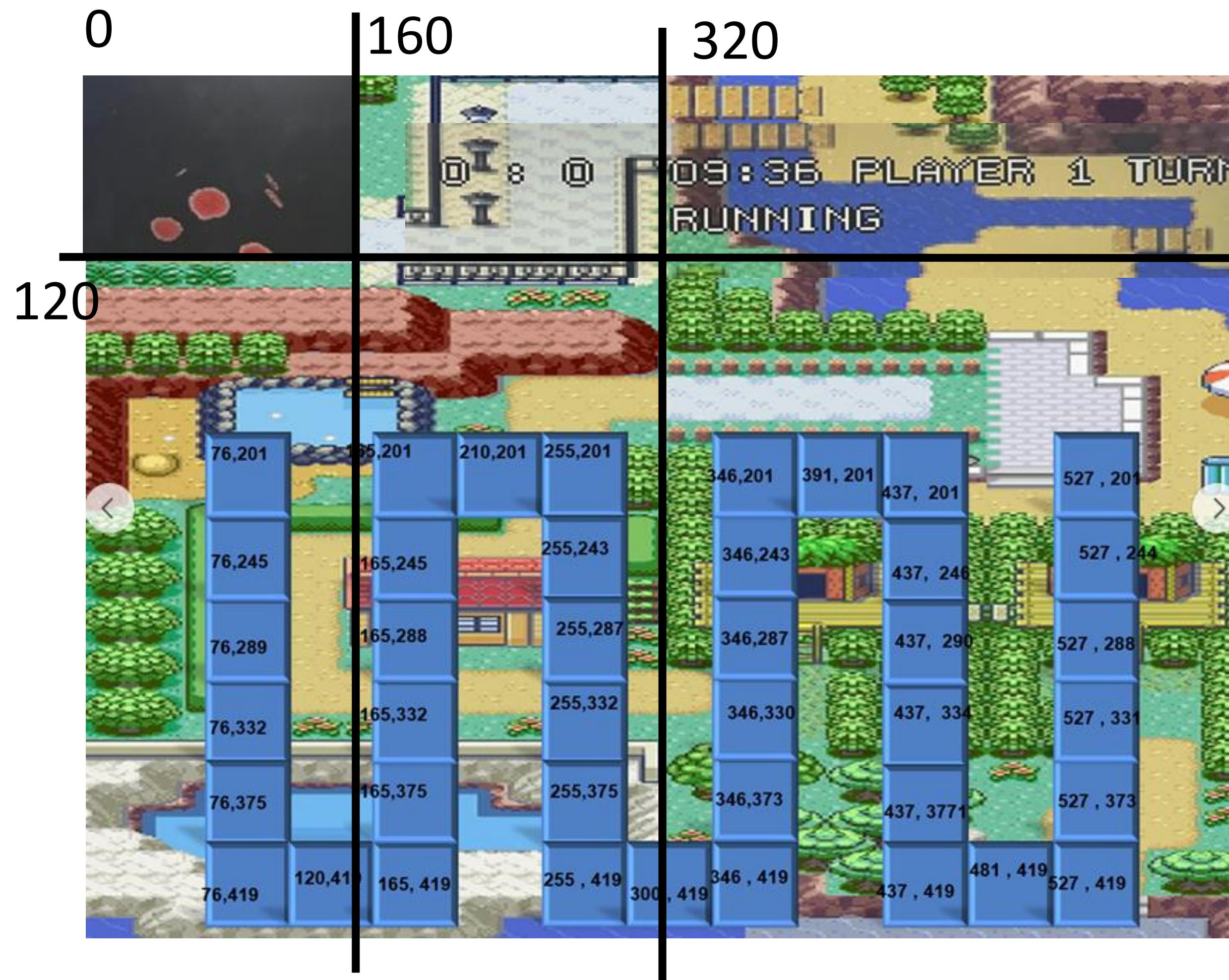
어떤 플레이어의 turn 인지에 대해

현재
게임상태

player 1(빨간)
player 2(블루)
위치



VGA_DISPLAY



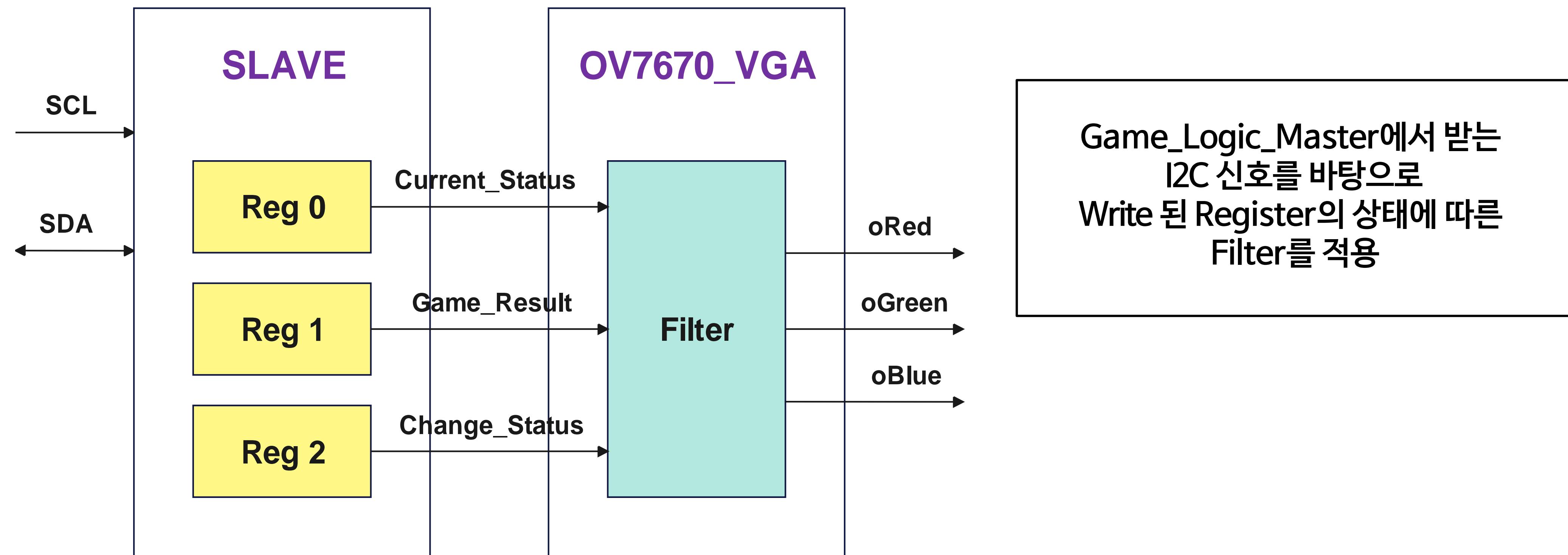
- **픽셀 카운트 판정 기반 문자 제어**
: GAME Logic 모듈로 부터 오는 실시간 게임정보 출력
(SCORE/ 시간 / 게임진행상태 / player turn)

04

Game Player Slave / Filter

GAME_PLAYER_SLAVE

Game_Play_Slave Block Diagram

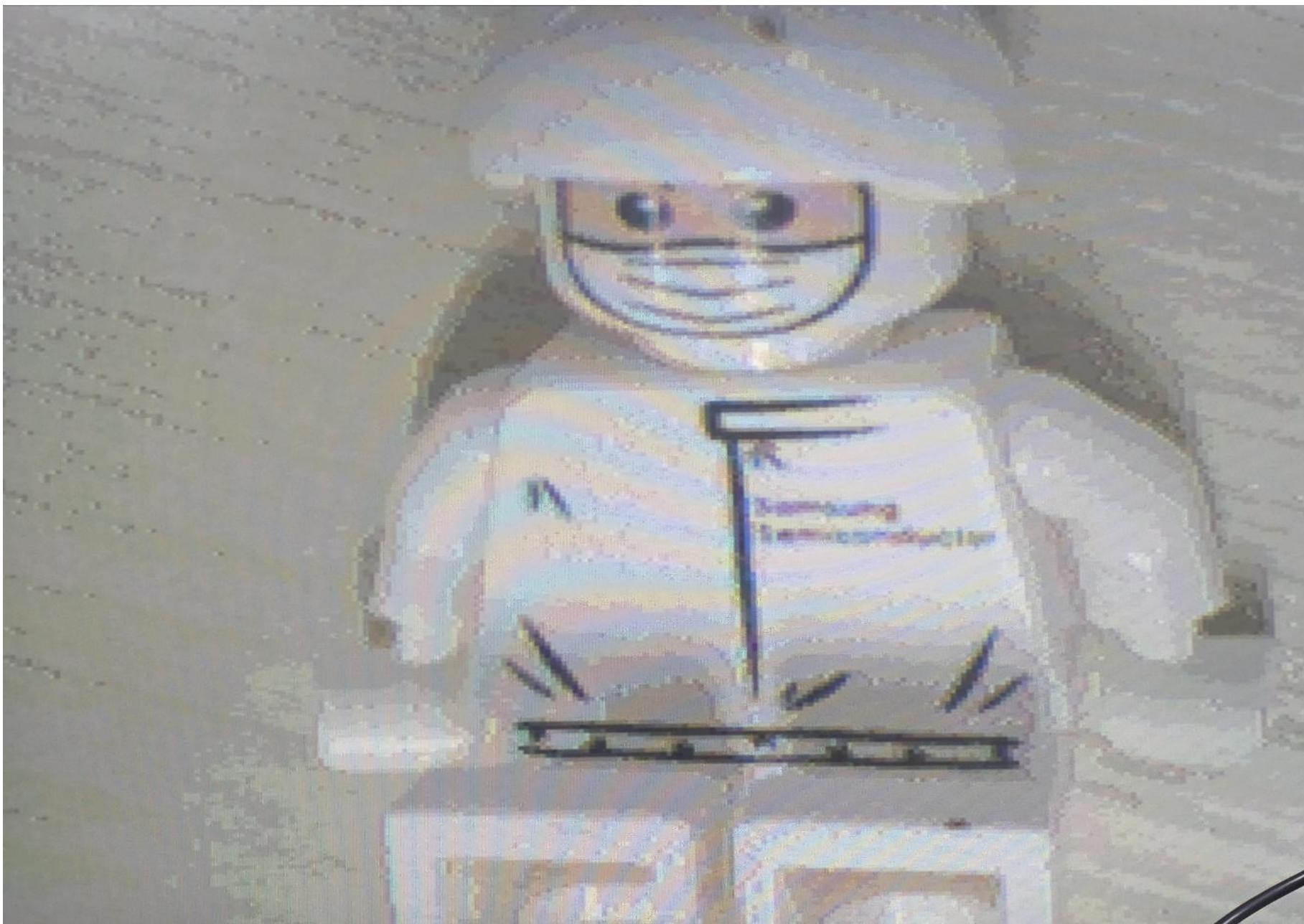


Slave Reg0 : 현재 상태

게임 진행 중 상태 확인

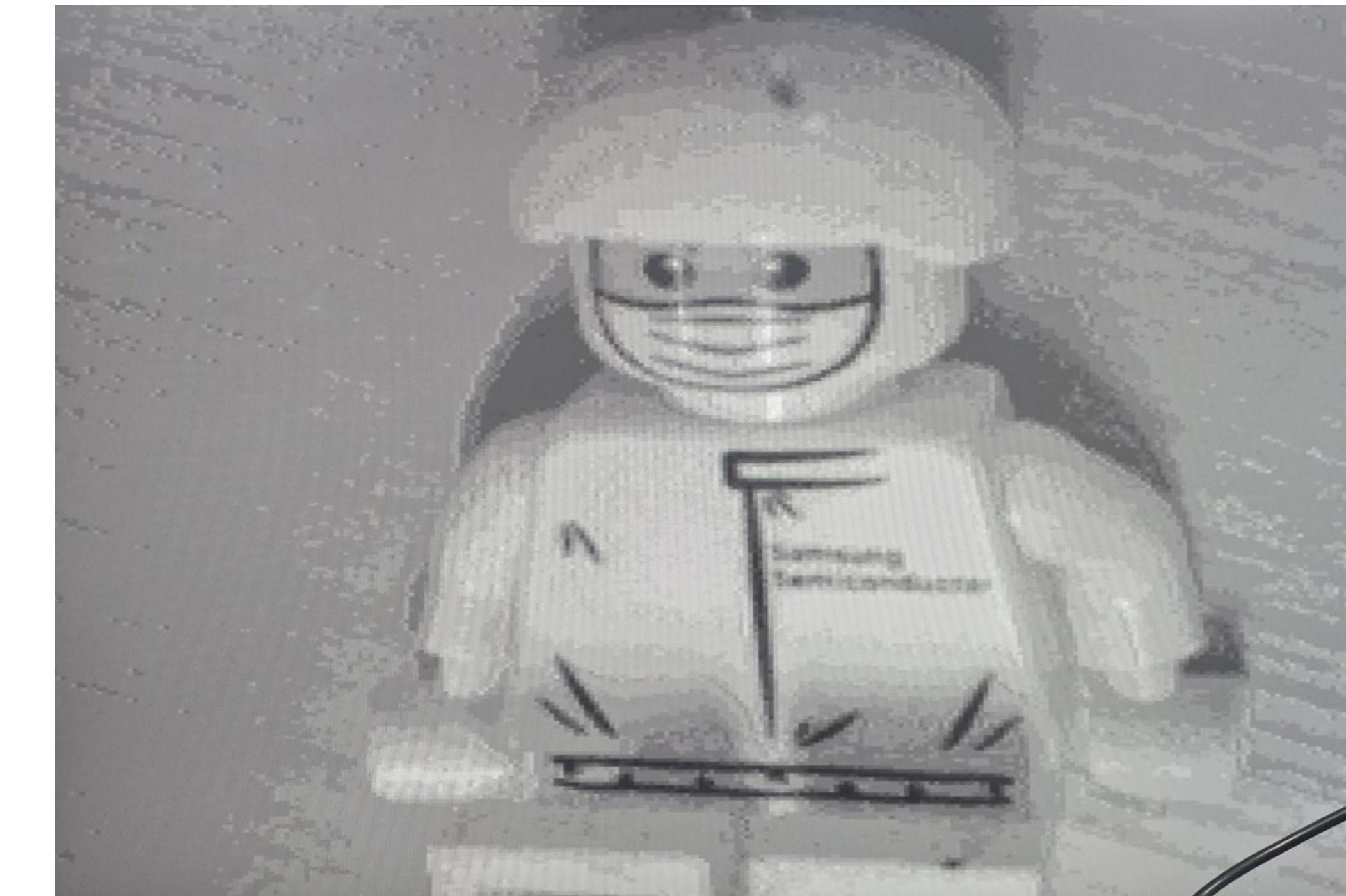
01

앞선 상황 (Normal)



02

뒤쳐진 상황 (Gray)

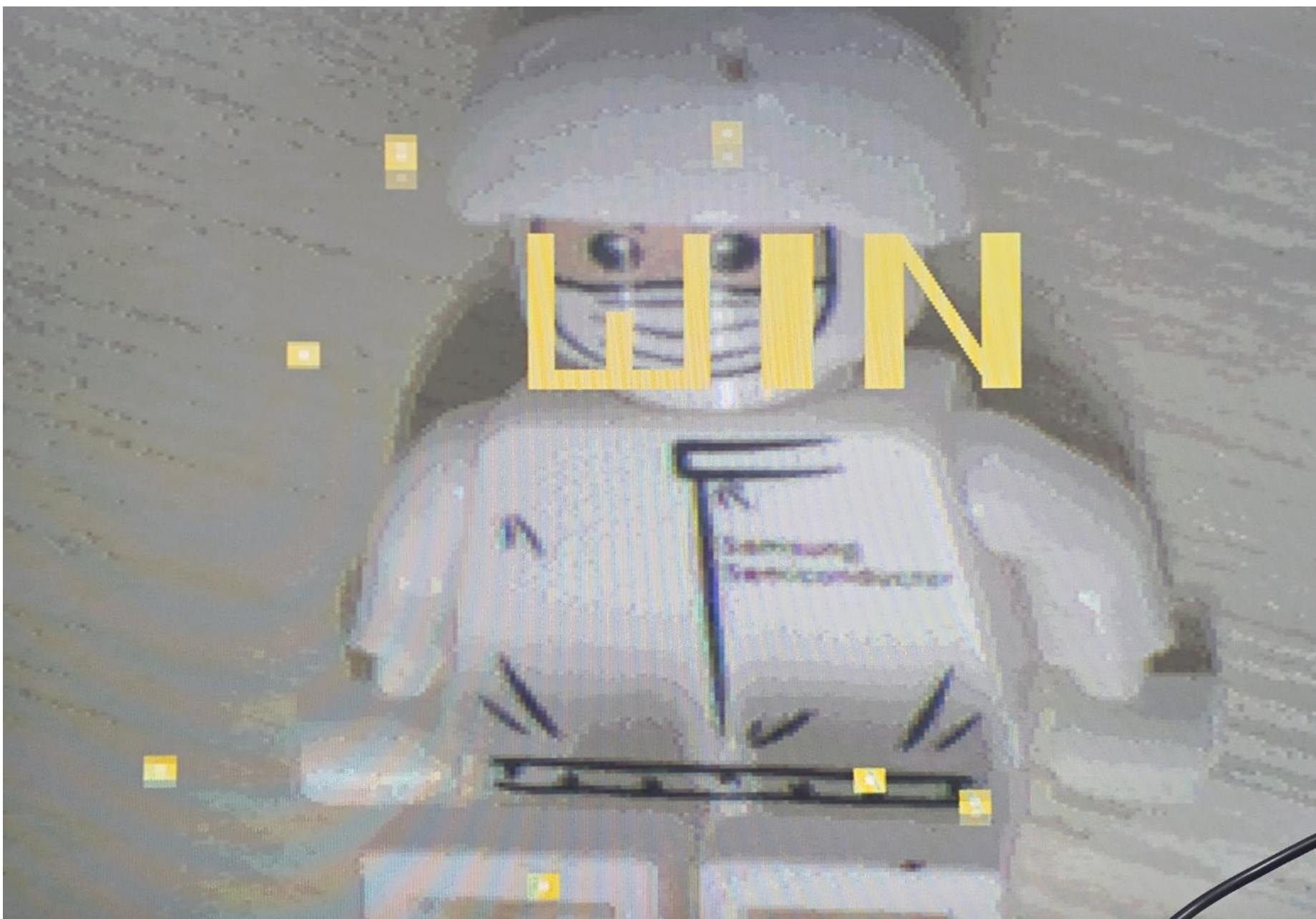


Slave Reg1 : 게임 종료

게임 종료 후 3판 2승제에 대한 player의 승/패 확인

02

승리자



03

패배자



Slave Reg2 : 게임 중 상태 변화

게임 중 함정/지름길에 따른 효과 (2초 지속 후 리셋)

01

함정



02

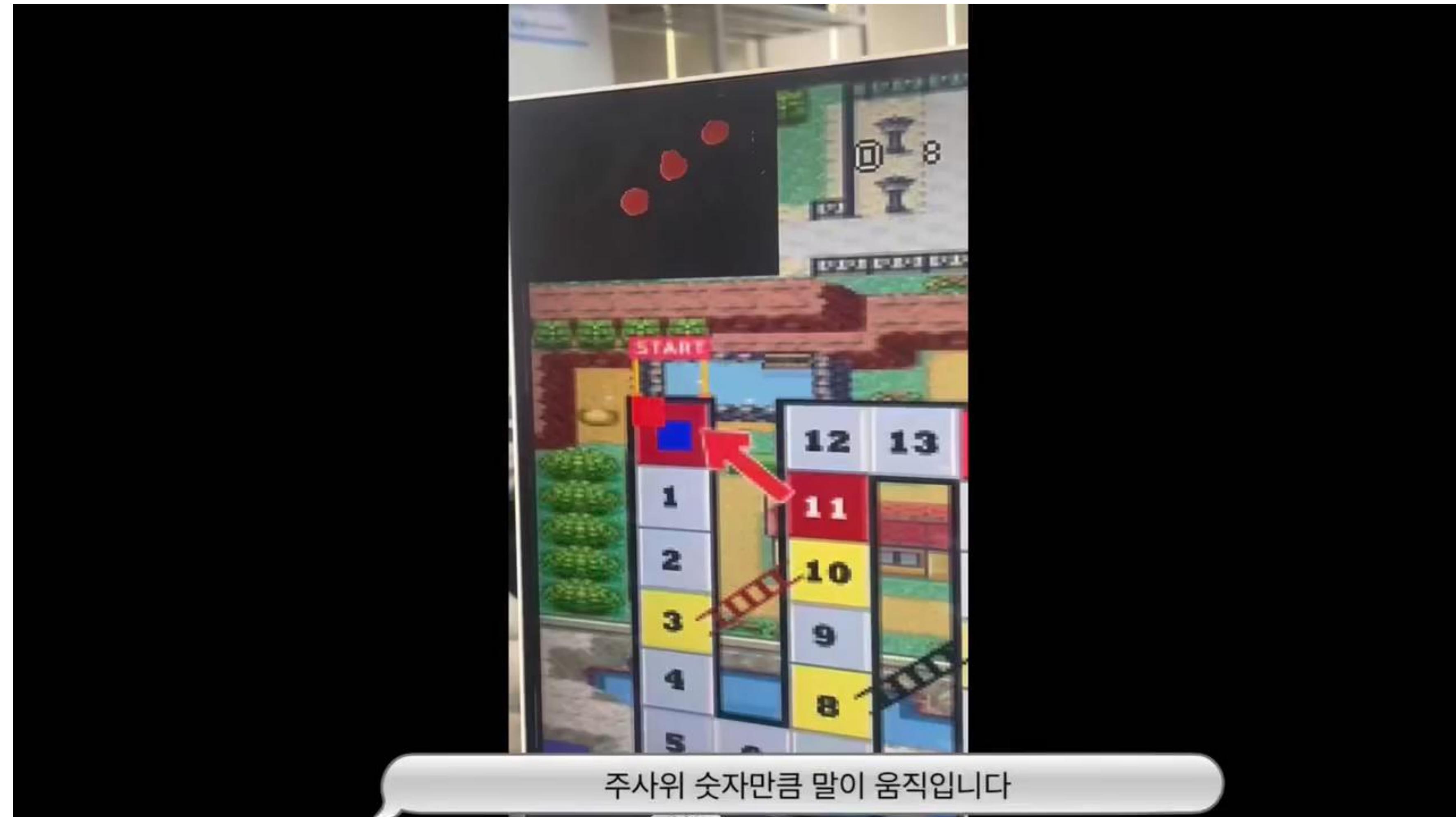
지름길



05

Game Video

Master & Slave 전체 확인 영상



Master & Slave 전체 확인 영상

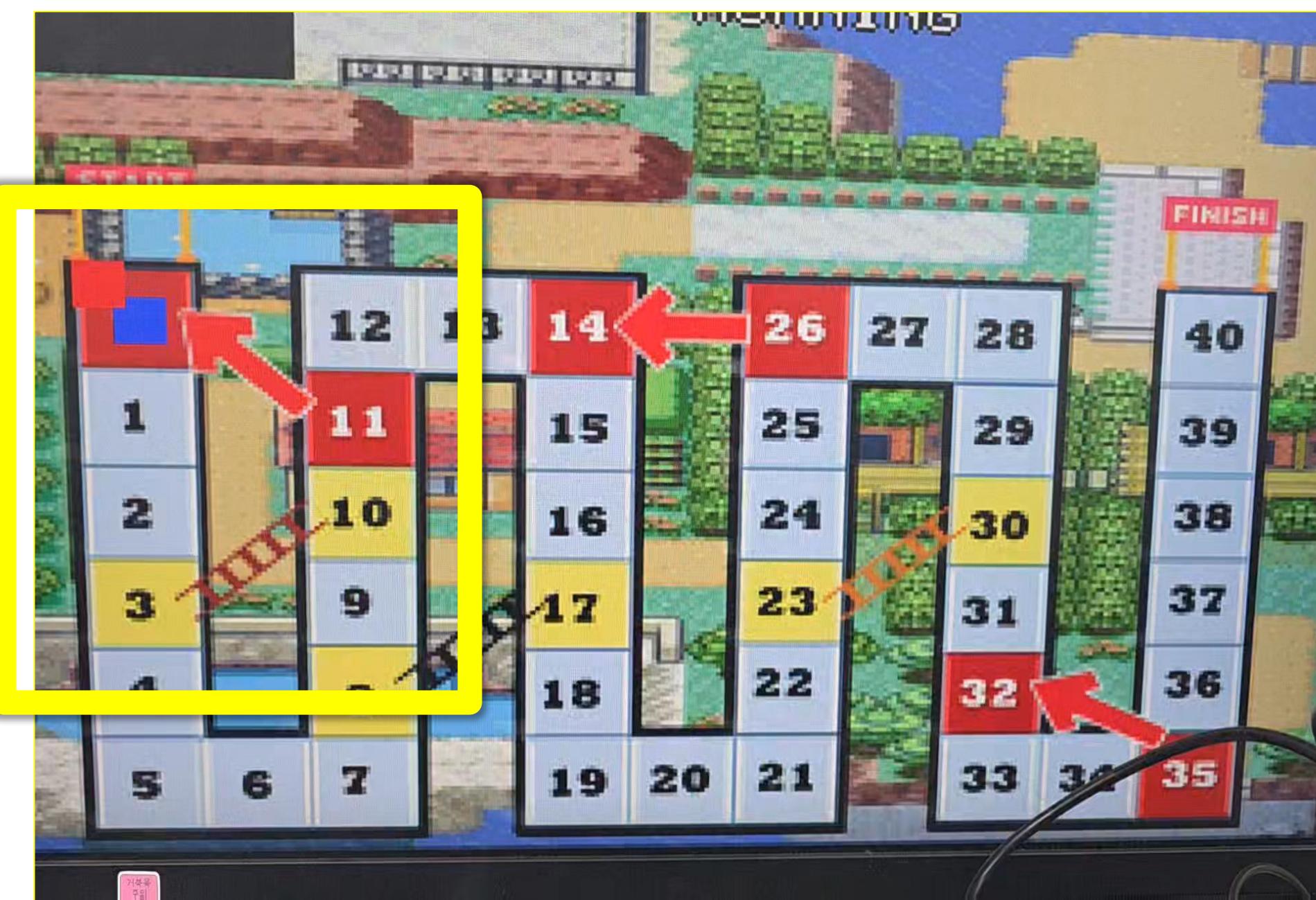


게임 중 상태 변화

게임 중 지름길/함정에 따른 움직임

01

지름길



02

함정



06

Trouble Shooting

Trouble Shooting (1) : @Dice reader

문제상황

빨간 Pixel 수만큼 LED가 켜지도록 설정하여 주사위 픽셀을 디버깅하는 도중 LED에 불이 들어오지 않는 상황이 발생

문제원인

If (vsync) 되는 순간,
저장과 리셋이 동시에 발생하면서 경합 발생.

→ final_count에 유효한 값이 저장되기 전에
0으로 리셋된 값이 먼저 반영되었음.

```
always_ff @(posedge pclk) begin
    if (reset) begin
        current_count <= 0;
        final_count <= 0;
    end else if (vsync) begin
        final_count <= current_count;
        current_count <= 0;
    end else begin
        if (we && is_red_pixel) begin
            current_count <= current_count + 1;
        end
    end
end
```

문제해결

유효 데이터 안전장치 추가.

→ 빨간 pixel을 세었을 때
바로 초기화하지 않고
pixel count값을
업데이트 하도록 설정.

```
always_ff @(posedge pclk) begin
    if (reset) begin
        current_count <= 0;
        final_count <= 0;
    end else begin
        if (vsync) begin
            if (current_count > 0) begin
                final_count <= current_count;
                current_count <= 0;
            end
        end else begin
            if (we && is_red_pixel) begin
                current_count <= current_count + 1;
            end
        end
    end
end
```



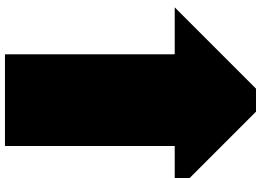
Trouble Shooting (2) : @ I2C Controller

문제상황

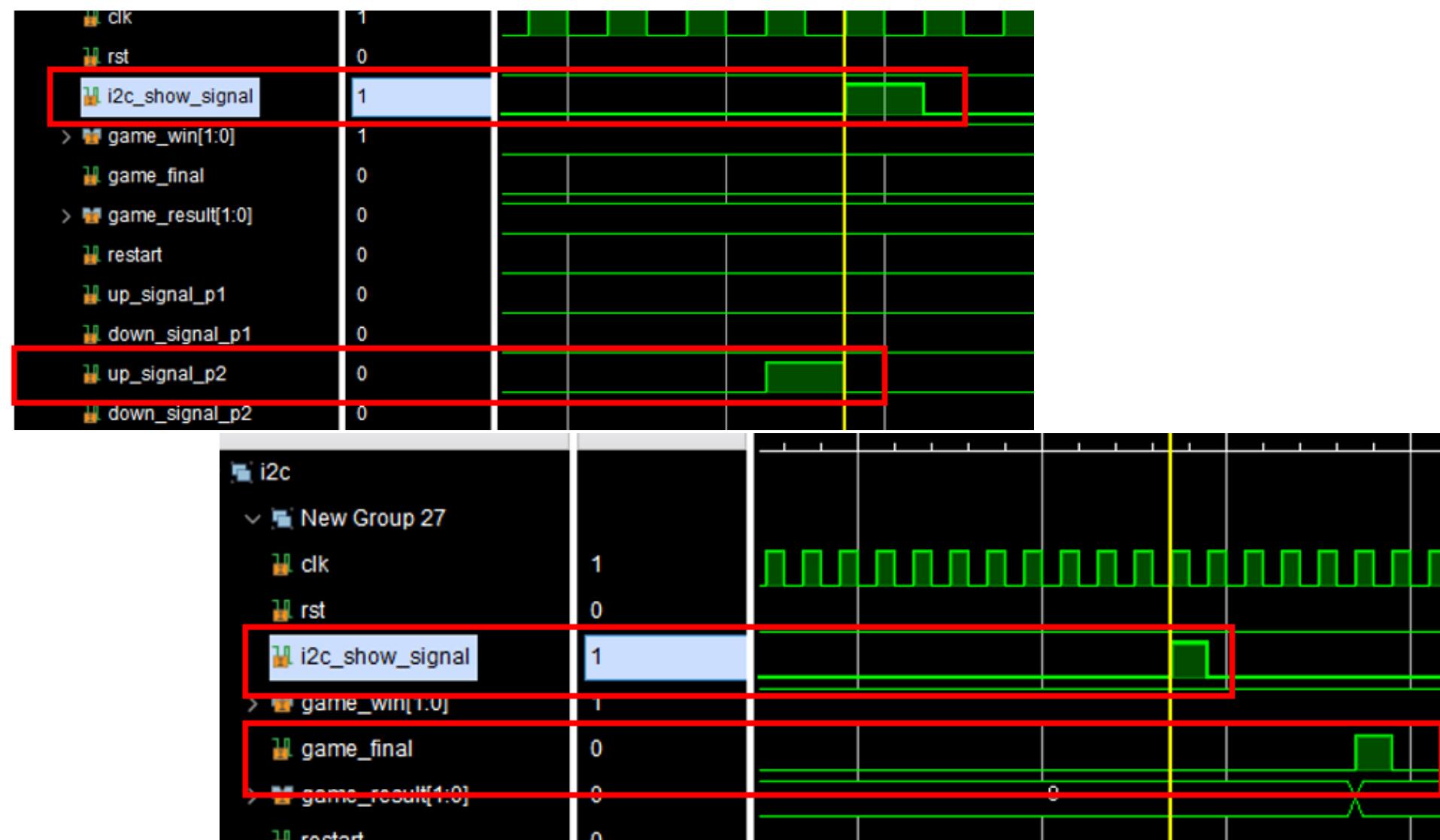
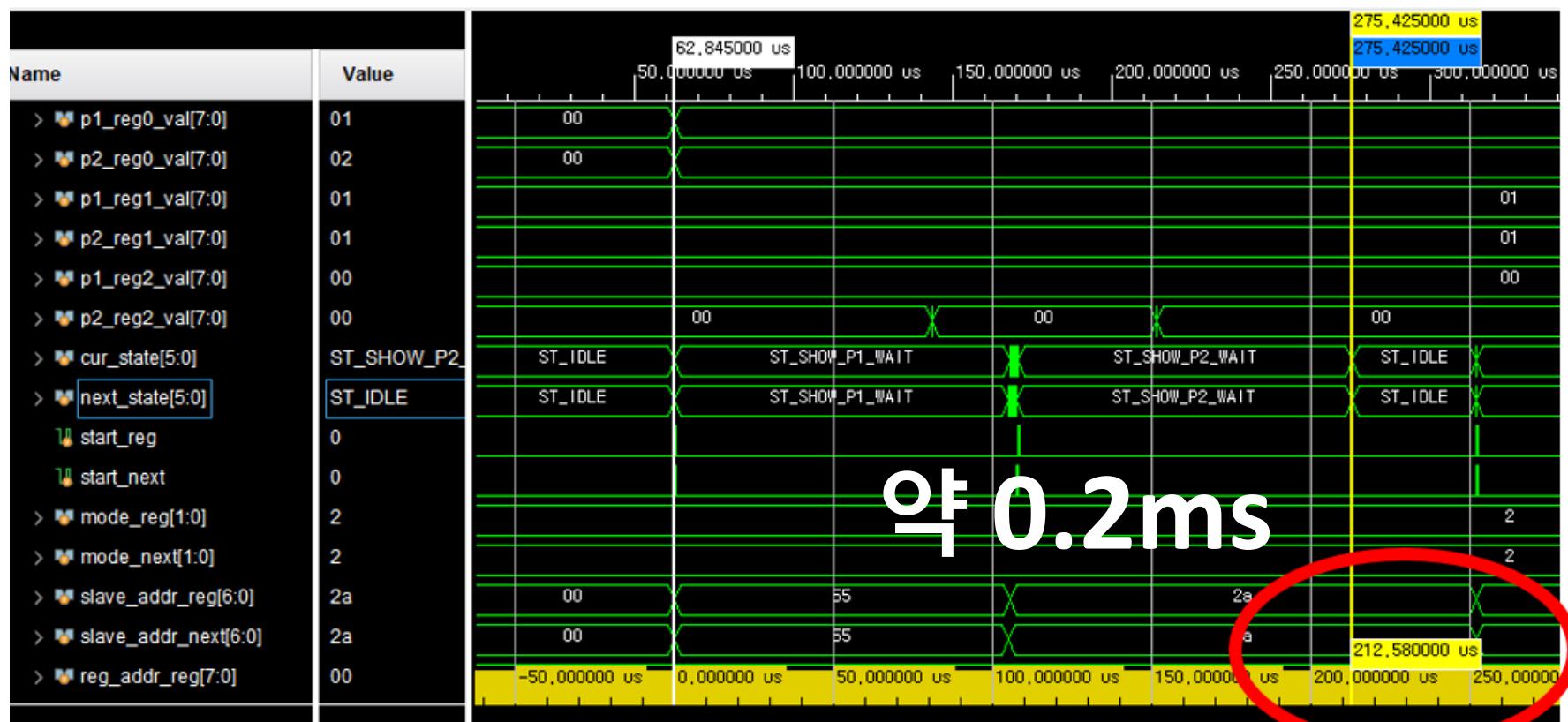
Game Logic + I2C Controller(Master) + I2C Slave interface 합쳐서 Test Bench 확인 시,
첫번째 발생 event 외에 다음 event가 무시 당하는 문제

문제원인

I2C event 가 시작 이후 끝이 나서 IDLE 상태로 오기까지
시간이 존재함 (약 0.2ms)
→ 해당 event 진행중, i2c master는 busy상태…



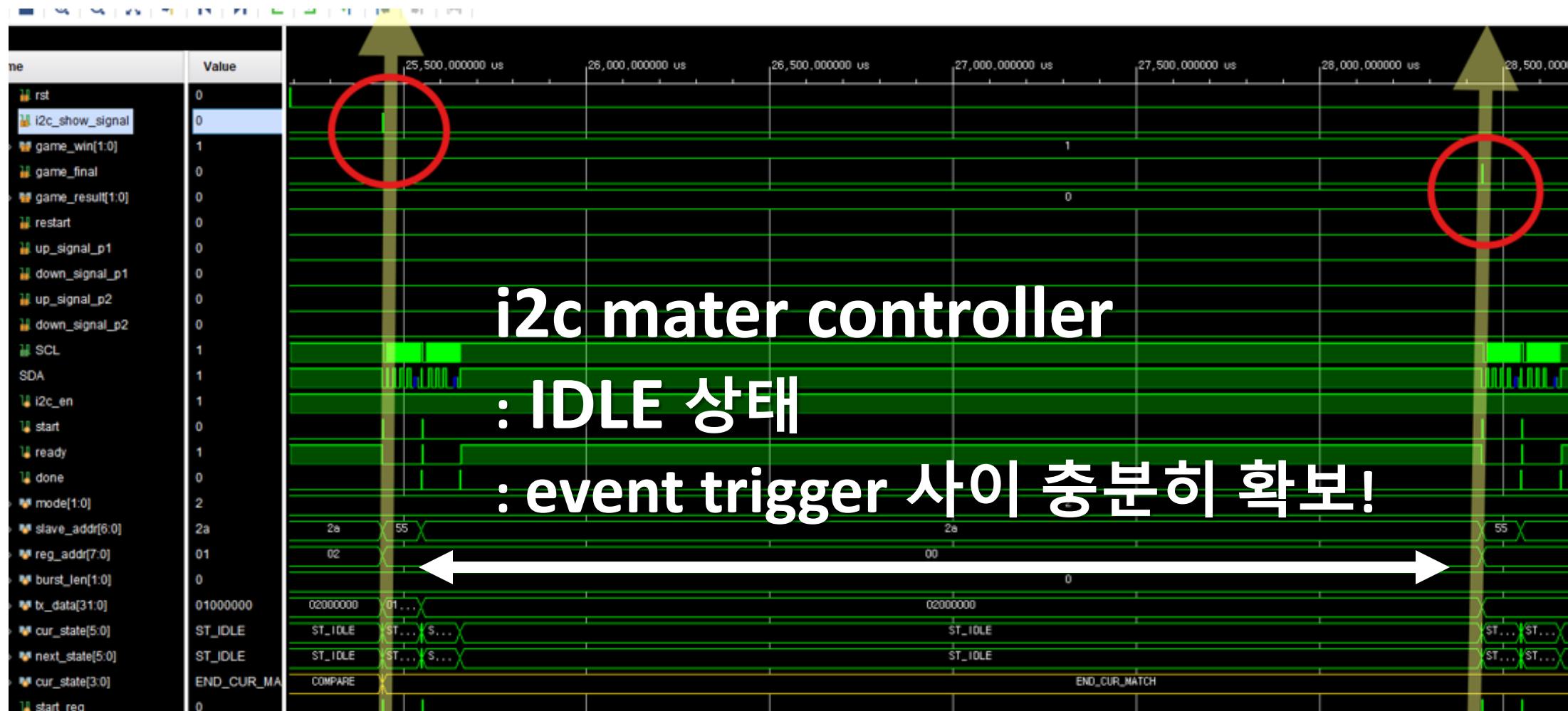
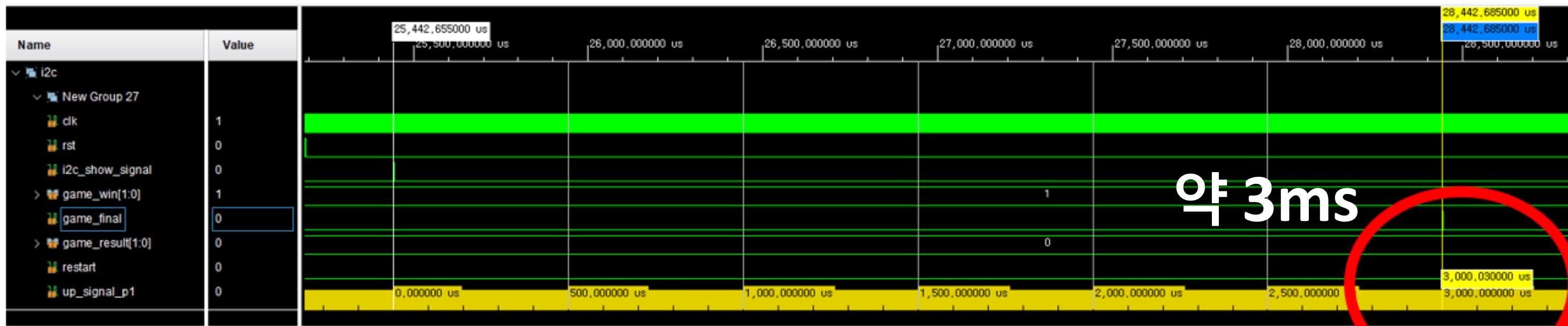
그 시간에 비해 I2C event Start 신호들 사이 시간이
너무 짧아서(10ns / 20ns) 첫번째 event 외에
다음 event 들이 무시 당하는 문제가 생김!



Trouble Shooting (2) : @ I2C Controller

문제해결

해당 event trigger를 만드는 Fsm logic에서 state를 넘어가기 전,
3ms counter를 통해 event 사이에 충분한 시간(3ms > 0.2ms)을 만들어 줌!!



테스트 벤치를 통해
미리 시나리오를 체크한 결과
실제 동작 시에도
무시되는 event 없이 i2c
통신이 정상적으로 동작함

Team



역할 : 팀장 / Game_Logic / I2C Master Controller / I2C 통신인터페이스 / SCCB 구현

소감 : 팀장으로서 병렬적인 프로젝트 진행을 위해 먼저 각 모듈 간의 입·출력 신호와 인터페이스를 명확히 정의하며 팀 작업 흐름을 조율하고 서로간의 약속을 명확히 했습니다. 또한 제가 담당한 게임의 코어 역할인 Game Logic은 다른 모듈이 완성되기 전까지 실제 하드웨어에서 바로 확인하기 어려웠기 때문에, 테스트벤치 기반 시뮬레이션으로 다양한 시나리오를 사전에 검증하며 설계의 정당성을 확보했습니다.

곽서유

그결과 최종 통합 단계에서 기능이 정상 동작했고, 불필요한 비트스트림 재생성과 디버깅 시간을 줄이며 효율적으로 통합할 수 있었습니다. 이 경험으로 협업에서 체계적인 서로간의 인터페이스 정의와 검증의 중요성을 체감했습니다



역할 : Dice Detector / VGA Display 구현 / 발표

소감 : 프로젝트에서 주사위 인식 알고리즘과 VGA 디스플레이 제어를 담당했습니다. 영상 인식에 앞서 이미지에서도 작동하는지 테스트벤치로 검증하였고, 알고리즘의 오류를 사전에 발견해 수정했습니다. 다운스케일링 기법으로 자원 효율성을 확보했으며, 데이터 처리 중 발생한 타이밍 경합 문제를 조건부 래치 구조로 해결했습니다.

황석현

이 과정에서 제한된 하드웨어 리소스를 최적화하고 시스템의 안정성을 설계하는 역량을 키웠습니다.

Team



한지윤

역할 : Player Movement / Visual Filter / I2C Slave & Top Integration

소감 : 이번 프로젝트에서 I2C Slave 인터페이스 설계와 전체 시스템 통합을 담당하며,
팀원들이 설계한 개별 모듈이 하나의 유기적인 시스템으로 완성되는 과정을 경험했습니다.

특히, 단순한 신호 전달을 넘어 'Auto-Clear' 기능이 포함된 레지스터 제어 로직을 직접 설계하여
필터 오동작 문제를 해결함으로써, 안정적이고 확장성 있는 하드웨어 제어 역량을 기를 수 있었습니다.



진우석

역할 : Back Ground Text 구현 / 영상편집

소감 : 이번 프로젝트에서 실시간 게임 정보를 VGA 화면에 확대·정렬·테두리 처리해 표시하는 기능을 구현했습니다.
과정 중 픽셀 매핑 오류로 글자가 깨지거나 제대로 출력되지 않는 문제가 반복적으로 발생했지만,
신호 흐름을 점검하고 좌표 연산을 수정하며 해결할 수 있었습니다.

이 과정에서 팀원들과 함께 원인을 분석하고 해결책을 논의하면서 협업의 중요성과 문제 해결 과정에서의
시야 확장의 가치를 실감할 수 있었습니다. 이번 경험은 VGA 제어 기술뿐 아니라 실제 하드웨어 문제를
직접 해결해 나가는 과정 자체가 큰 배움이 되는 프로젝트였습니다.



THANK YOU
FOR WATCHING!

