

VGA-based

Filter & Game System

6조 김은성 김재우 유지훈 조현우

목차

1

Introduction

2

**System
Architecture**

3

**VGA
Controller**

4

**Filter
Controller**

5

**Game
Controller**

6

**Trouble
shooting**

7

Demo

8

Conclusion

Introduction

Introduction

Project Goal

① VGA & Memory Control

- Camera Capture: OV7670 데이터 캡처 및 Frame Buffer(Block RAM) 동기화
- VGA Controller: 640x480 @ 60Hz 안정적 영상 출력 타이밍 생성
- Data Optimization: RGB565 -> RGB444 변환으로 대역폭 최적화

② Real-time Image Filtering

- Processing Core: Line Buffer 기반
실시간 3x3 윈도우 연산 구현
- Filter Set: Sobel(Edge), Gaussian(Blur),
Morphology, Cartoon, Retro 필터 적용
- Quad View: 4분할 화면으로 여러가지 필터 효과 동시 확인

③ Mini Game

- Color Detection: 화면 4개 영역(LT, RT, LB, RB)의 색상(Red/Green/Blue/Black) 인식
- Stability: 오인식 방지를 위한 Hold Threshold(45 Frame) 알고리즘 적용

④ Display Overlay

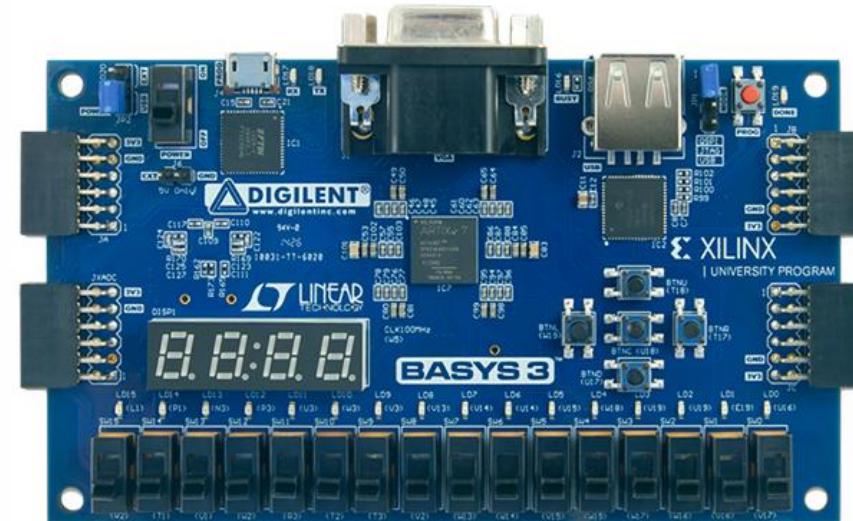
- Bitmap Font를 활용한 실시간 점수 및 상태 오버레이
- 정답/오답 시각 효과(Neon Box)로 직관적 인터페이스 제공

Introduction

개발 환경



- Language : SystemVerilog



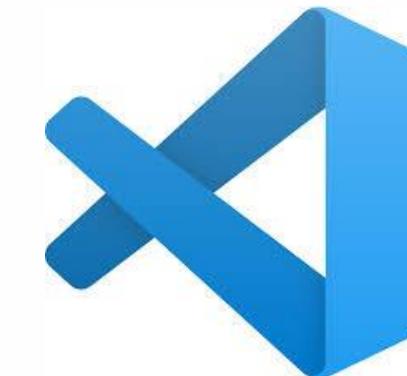
- Hardware : Basys3



- Design Tool : Xilinx Vivado



- Hardware : OV7670



- Editor : Visual Studio Code

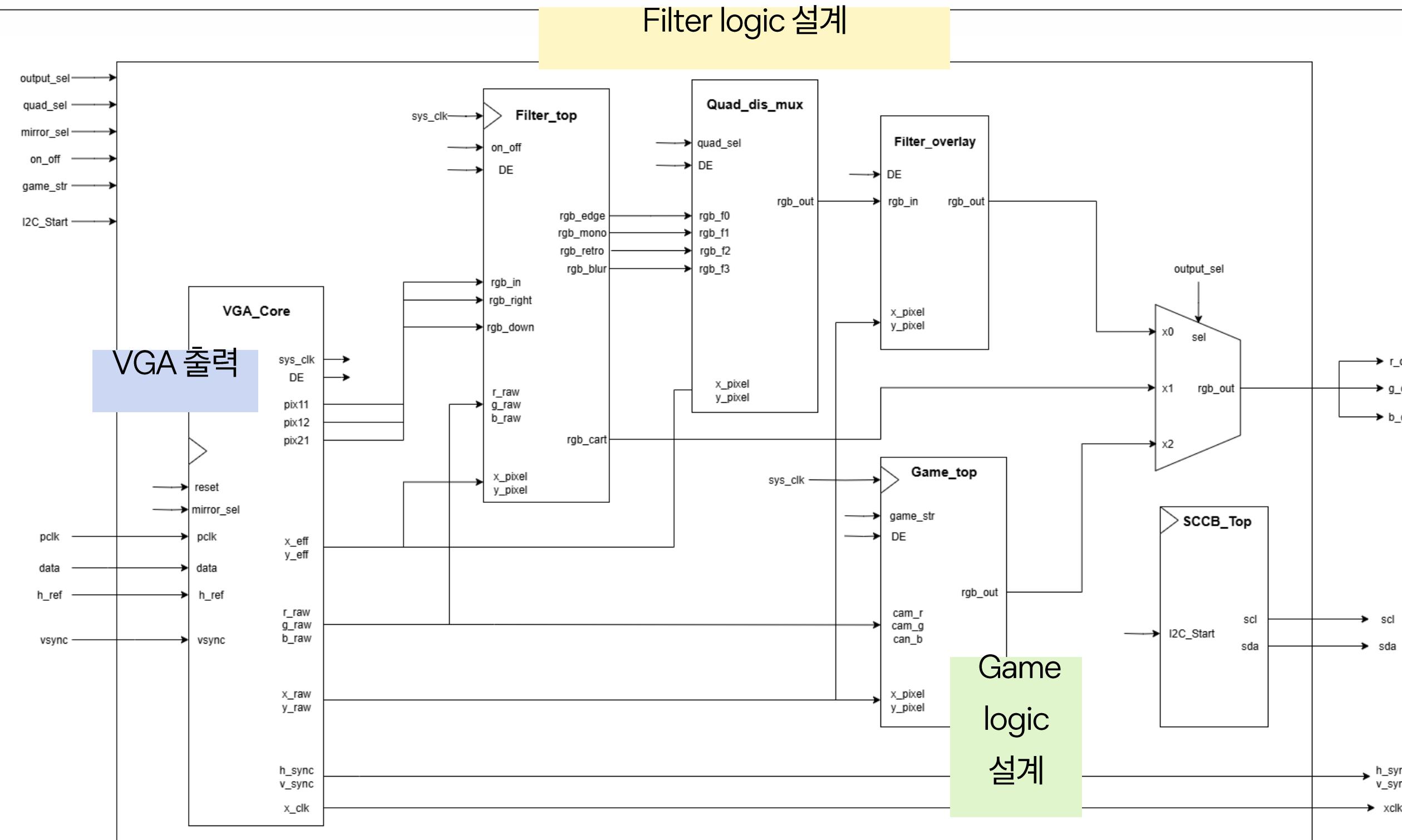


- Hardware : VGA port

System Architecture

System Architecture

TOP Block Diagram



System Architecture

✓ PART 01

Real-time Image Filtering

- 카메라 입력의 실시간 처리
- 다양한 영상 효과 (Edge, Blur, Retro 등) 적용
- 4분할 화면(Quad View)으로 필터 효과 동시 확인

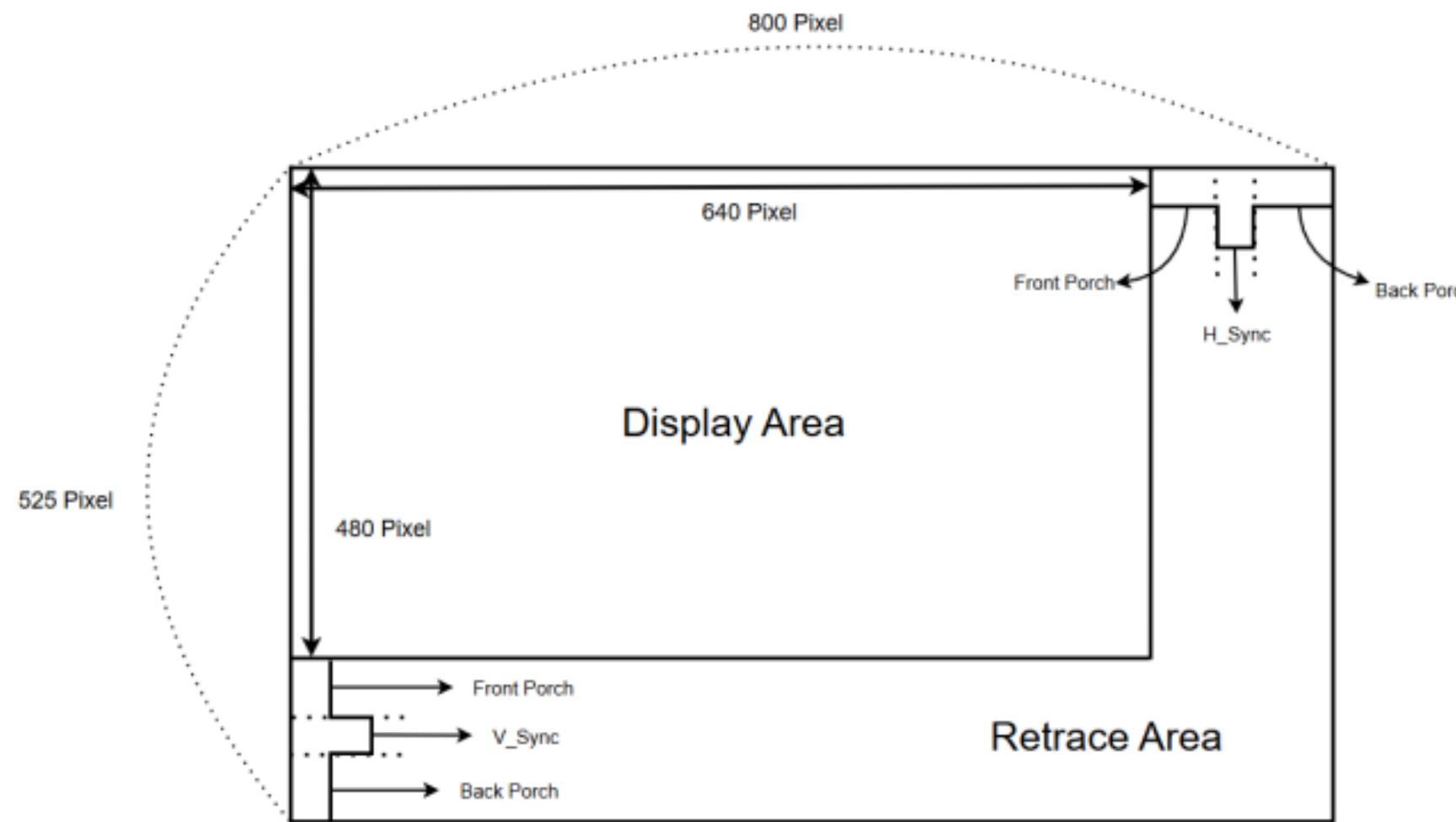
✓ PART 02

Interactive Game

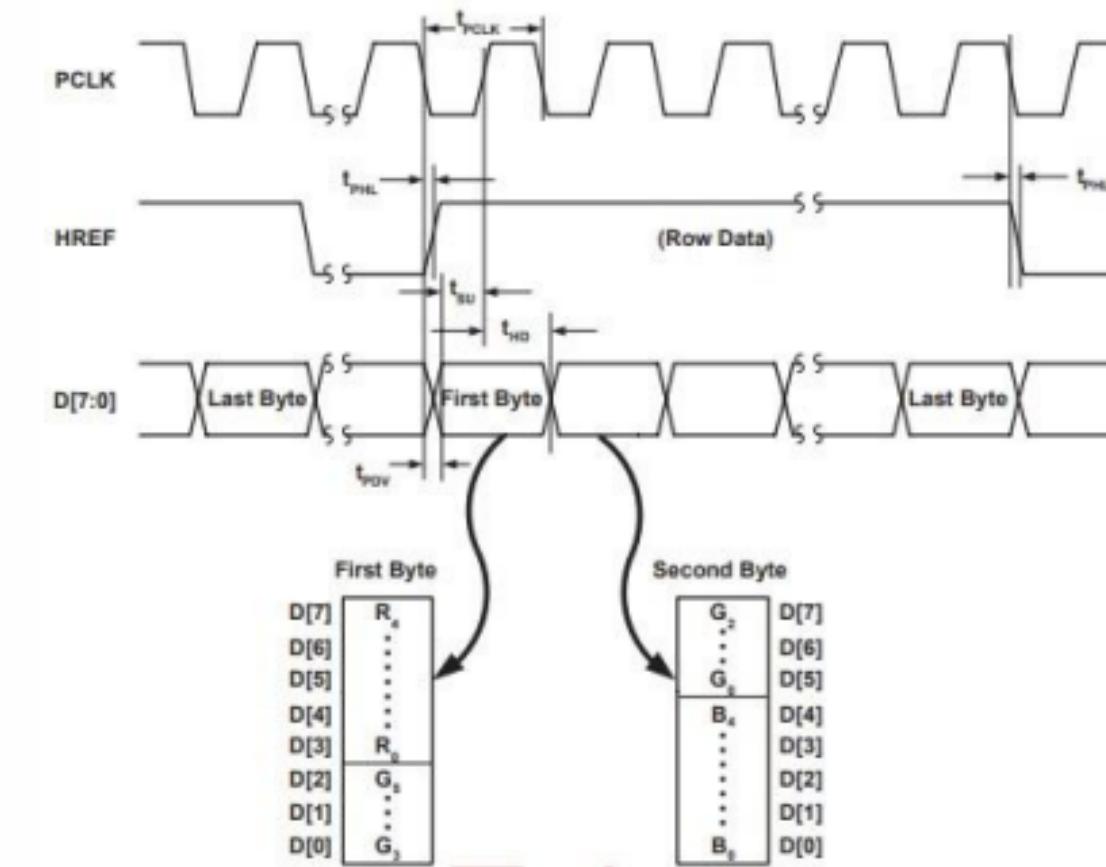
- 실시간 반응형 게임 플레이
- 화면 지시에 따라 4분할 된 영역에 알맞은 색상을 제시하는 순발력 게임 (청기백기 방식)
- 색상 검출(Color Detection) 알고리즘을 활용하여, 사용자가 제시한 물체의 색상과 위치를 실시간으로 판정

VGA Controller

VGA Controller



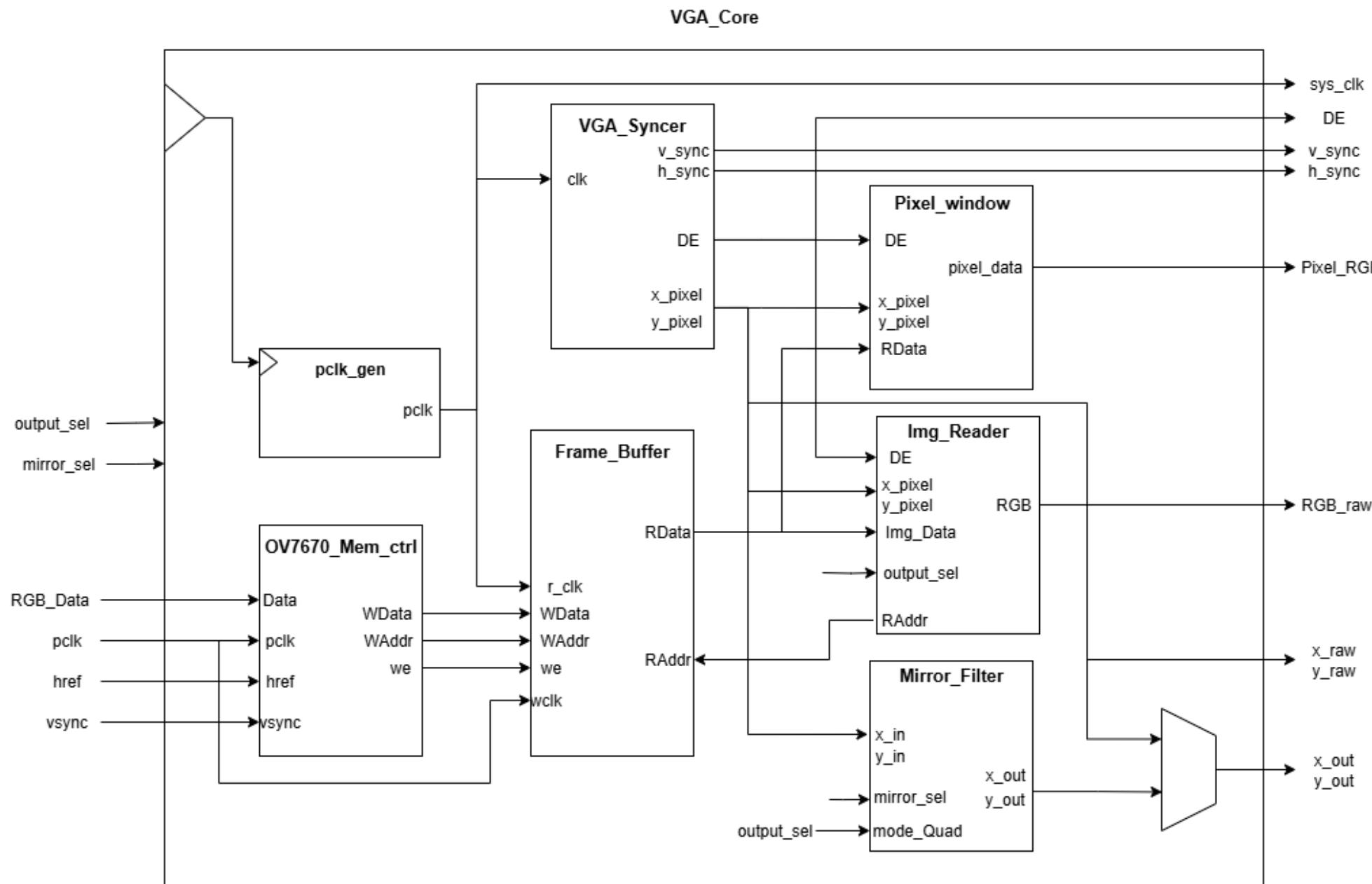
640x480 해상도 출력을 위해 전체 800x525 클럭 사이클을 카운팅하며, 화면에 표시되는 유효 영역(Display Area) 외의 구간에서는 RGB 값을 차단하여 모니터 동기화를 유지



OV7670 카메라에서 2번에 걸쳐 들어오는 8비트 데이터를 16비트(RGB565)로 조합한 후, 상위 비트(MSB)를 추출하여 FPGA 보드에 맞는 12비트(RGB444) 포맷으로 변환

VGA Controller

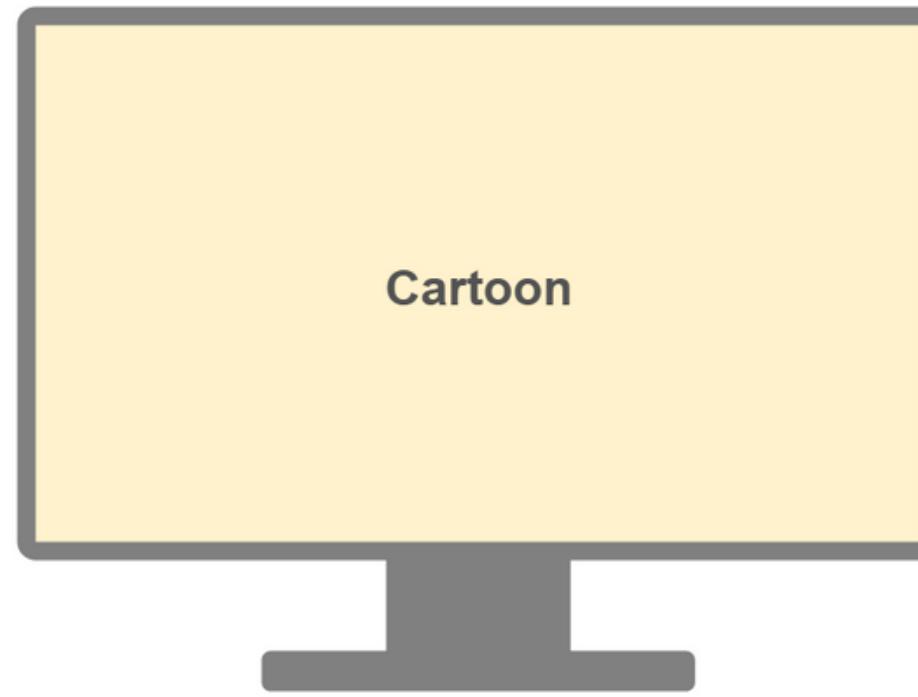
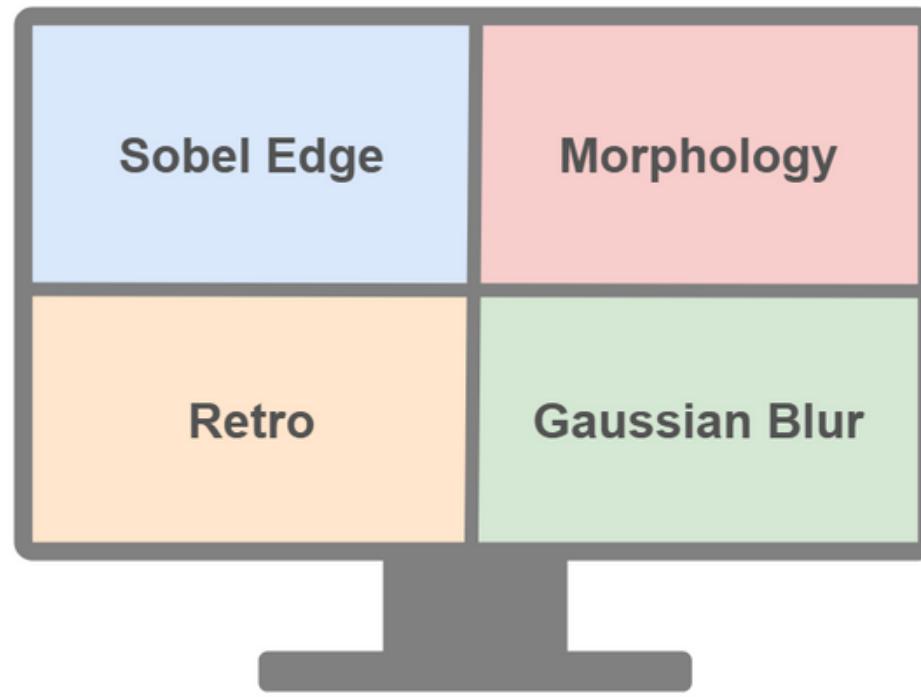
VGA Controller Block Diagram



카메라(OV7670)에서 들어오는 데이터를 메모리(Frame Buffer)에 저장하고,
이를 다시 꺼내어 VGA 모니터 타이밍에 맞춰 출력하는 중앙 제어 장치

Filter Controller

Image Processing



- 스위치를 통해서 4분할 필터와 카툰 필터로 전환 가능

Image Processing

✓ Retro

- 역할: 고전 8비트 게임과 같은 픽셀 아트(Pixel Art) 느낌을 연출하는 시각 효과 필터
 - 원리: 이미지의 해상도를 강제로 낮추는 모자이크(Pixelation)와 색상 단계를 단순화하는 양자화(Quantization)를 동시에 적용
- 픽셀이 큼직한 모자이크 형태로 변하고 색상이 단순화되어, 고전 8비트 게임 같은 독특한 느낌을 연출함

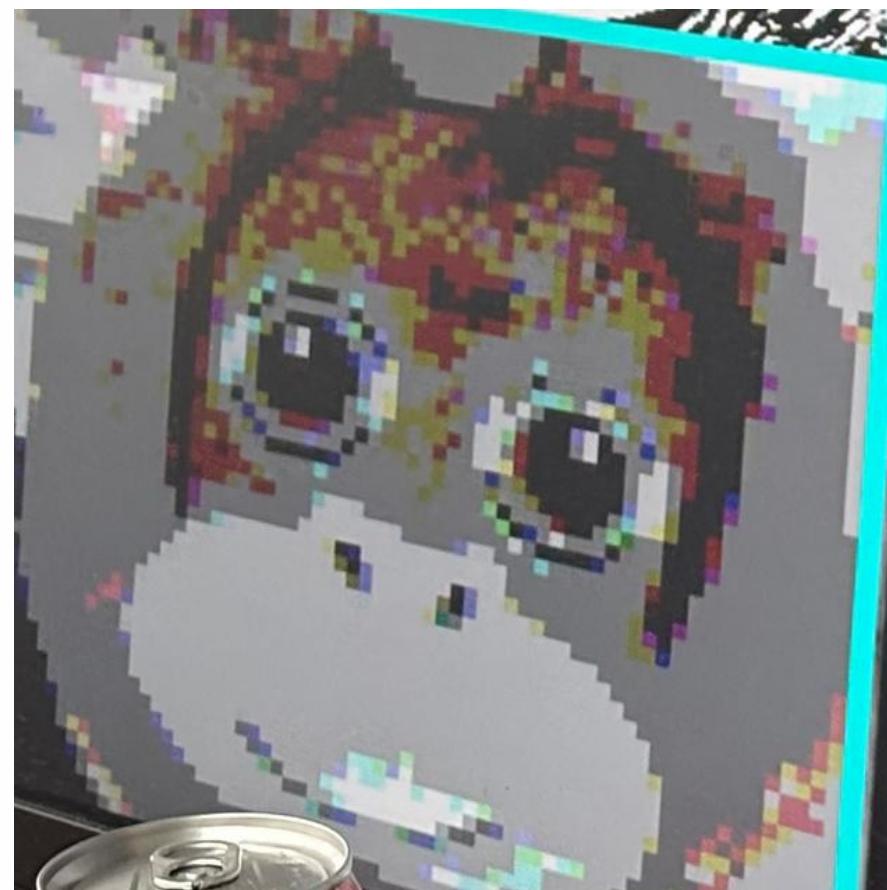
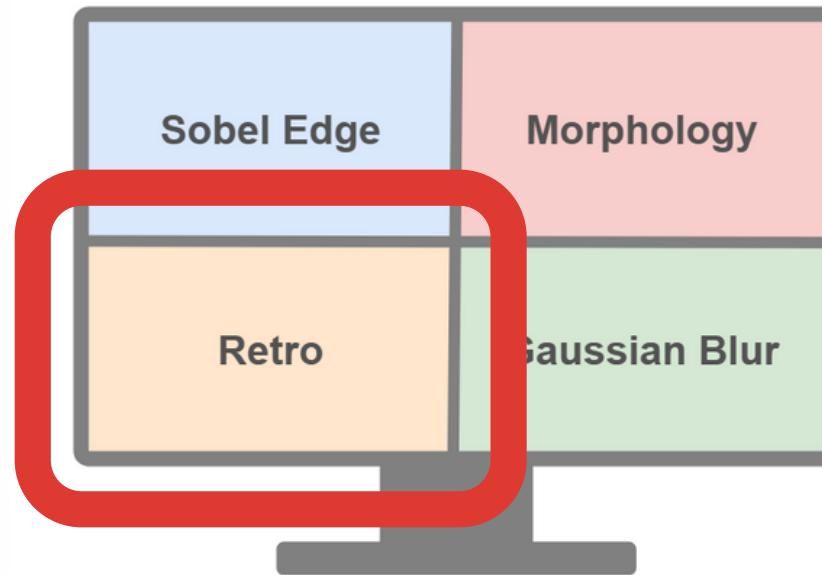
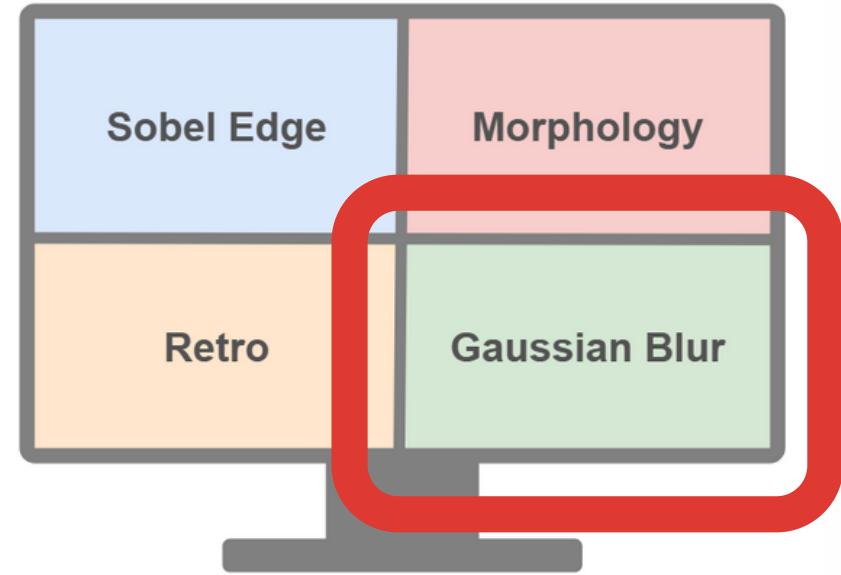
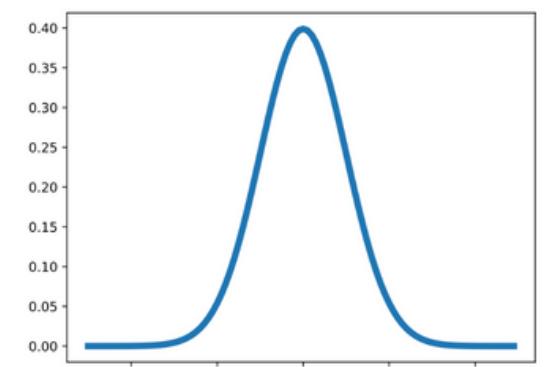


Image Processing



✓ Gaussian Blur

- 역할: 이미지의 노이즈를 제거하고 부드럽게 만드는 블러(Blur) 효과 필터
 - 원리: 중심 픽셀에 가장 큰 가중치를 주고, 주변으로 갈수록 낮은 가중치를 주는 '가중 평균' 방식
- 전체적으로 뿌옇게 흐려지며 고주파 노이즈가 제거되어, 부드러운 느낌을 줌
- Linebuffer 딜레이로 인한 좌측 화면 보라색 노이즈 발생
- Gausian 필터의 적용 범위를 픽셀단위로 조절하여 불완전한 필터 영역 문제 해결

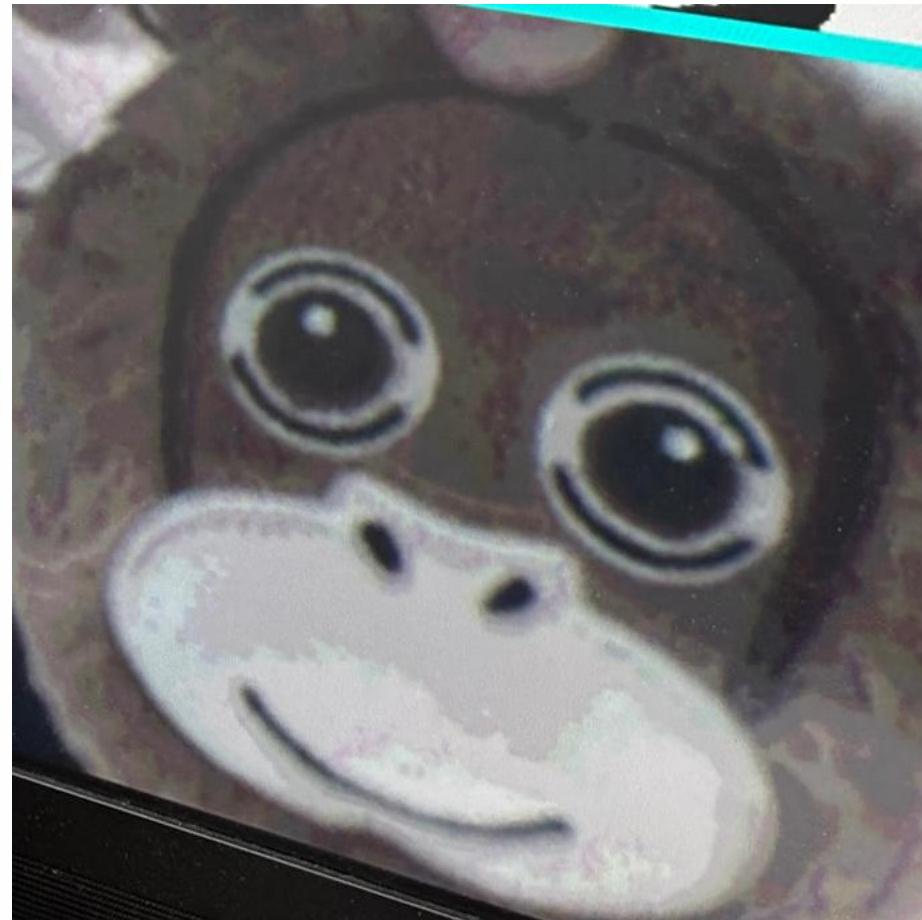


Image Processing

✓ Morphology

- 역할: 이진화(Binary)된 이미지에서 작은 노이즈(점)를 제거하고
 객체의 형태를 다듬는 필터
 - 원리: 이미지를 깎아내는 침식(Erosion) 연산을 수행한 후, 다시 부풀리는
 팽창(Dilation) 연산을 수행하는 오픈닝(Opening) 기법 사용
- 영상의 자잘한 잡티나 점(Noise)들은 깎여서 사라지고,
 주요 물체의 형태와 덩어리감은 깔끔하게 유지됨

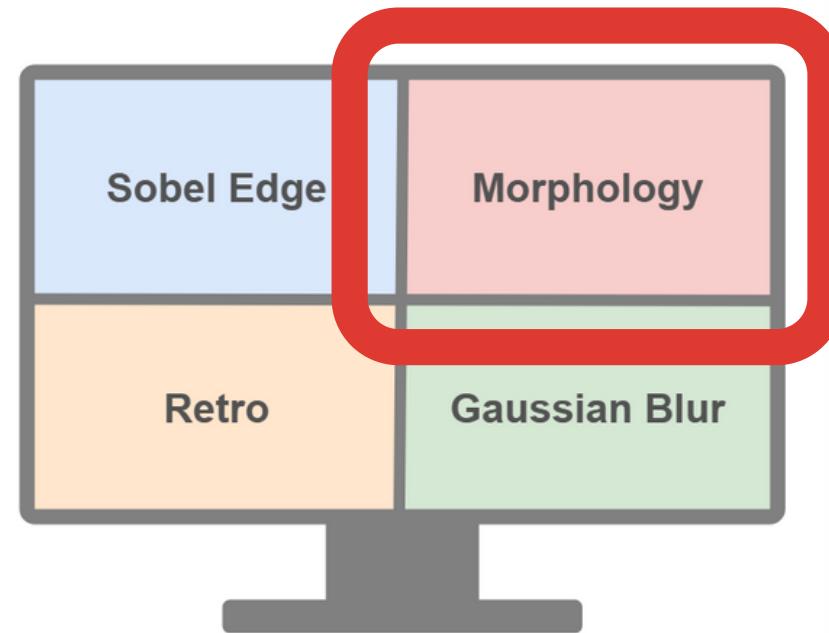


Image Processing

✓ Sobel Edge

- 역할: 이미지에서 밝기 변화가 급격한 부분, 즉 테두리(Edge)를 검출하는 필터
 - 원리: 가로 방향(G_x)과 세로 방향(G_y)의 밝기 변화량(미분값)을 각각 계산하여 합치는 방식
 - 소벨 엣지 전처리로 가우시안을 쓰는 이유:
소벨 엣지의 밝기 차이 비교 로직이 잡음에 민감하게 반응하기 때문에
- 배경과 색상 정보는 사라지고 물체의 윤곽선만 하얗게 강조되어,
사물의 형태가 뚜렷하게 드러남

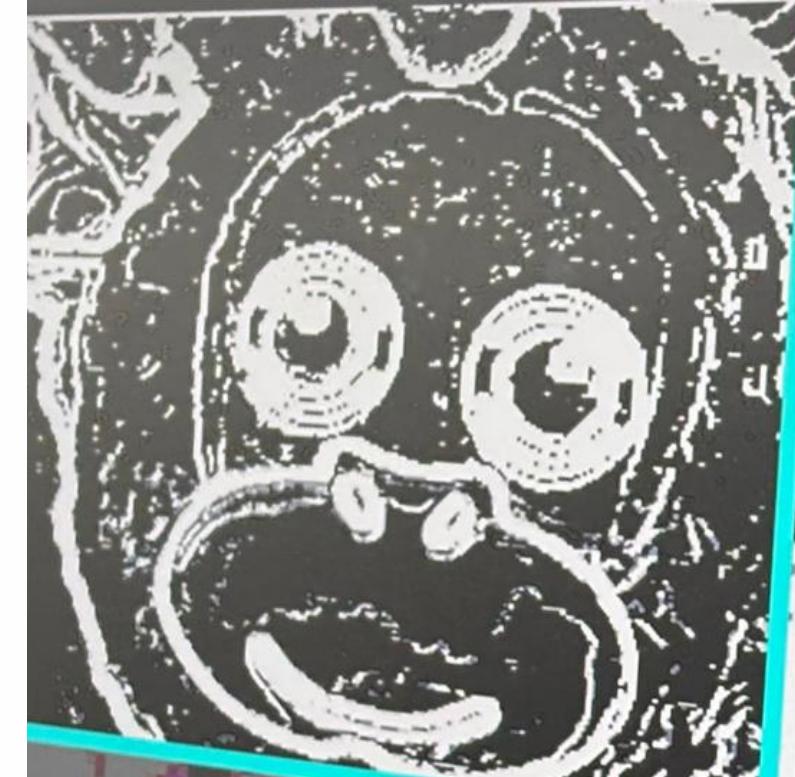
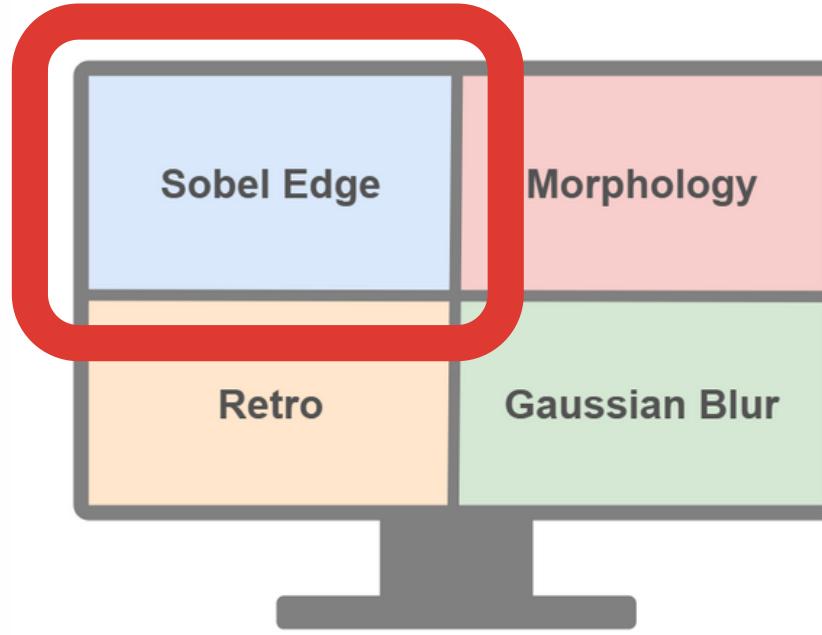


Image Processing

✓ Mirror

- 역할 : 화면의 중심축을 기준으로 영상을 반전시켜 대칭(Symmetry) 이미지 생성
- 원리 : 메모리에서 픽셀을 읽어올 때, 좌표 변환 기술을 사용
 - : 원본 좌표 (x, y) 대신, 화면 폭에서 현재 좌표를 뺀 대칭 좌표 ($width-x, y$) or ($x, width - y$) 의 데이터를 읽어와 출력하는 방식
- 화면의 좌우 또는 상하 좌표가 대칭 반전되어,
거울을 보거나 데칼코마니를 한 듯한 시각적 효과를 연출

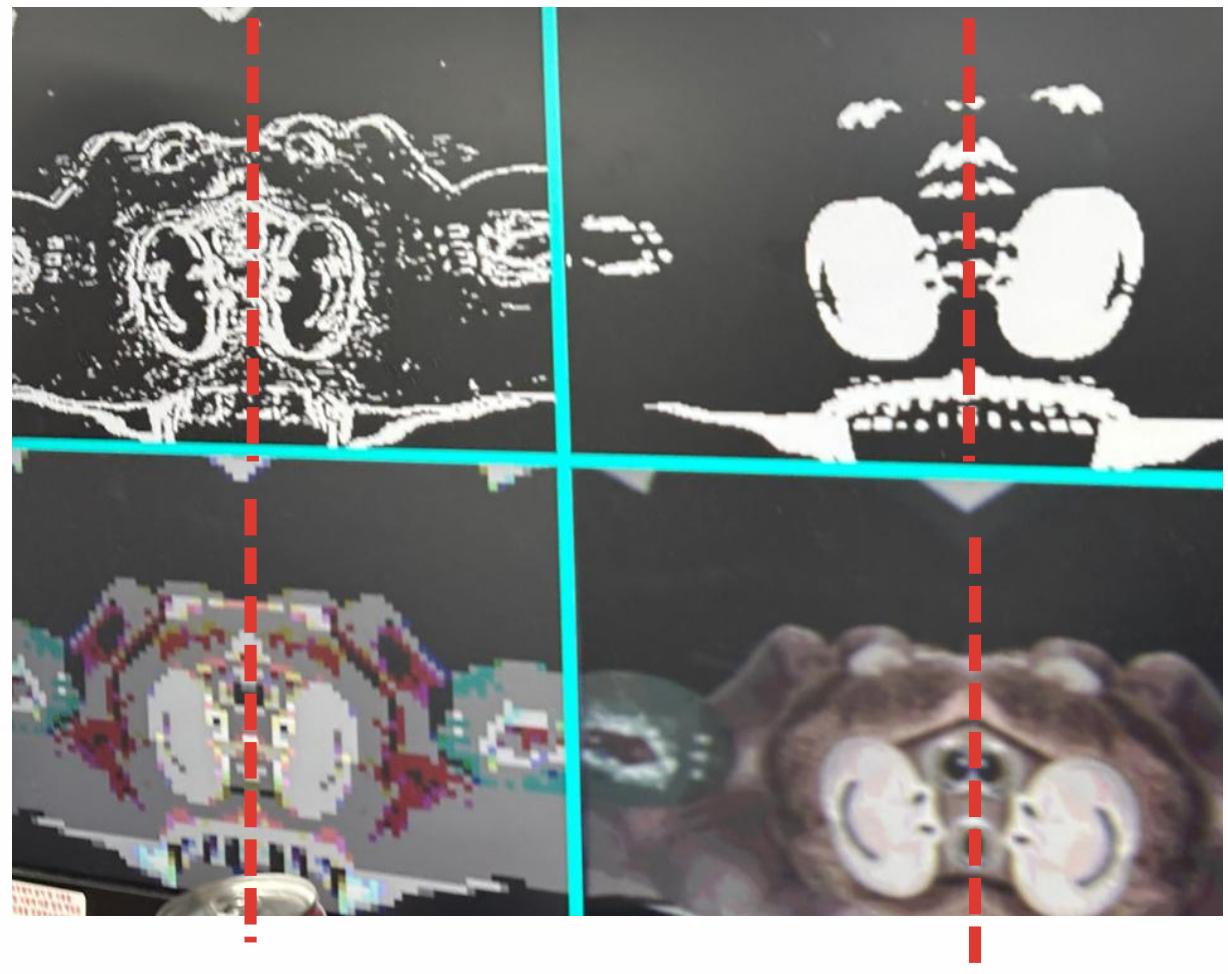
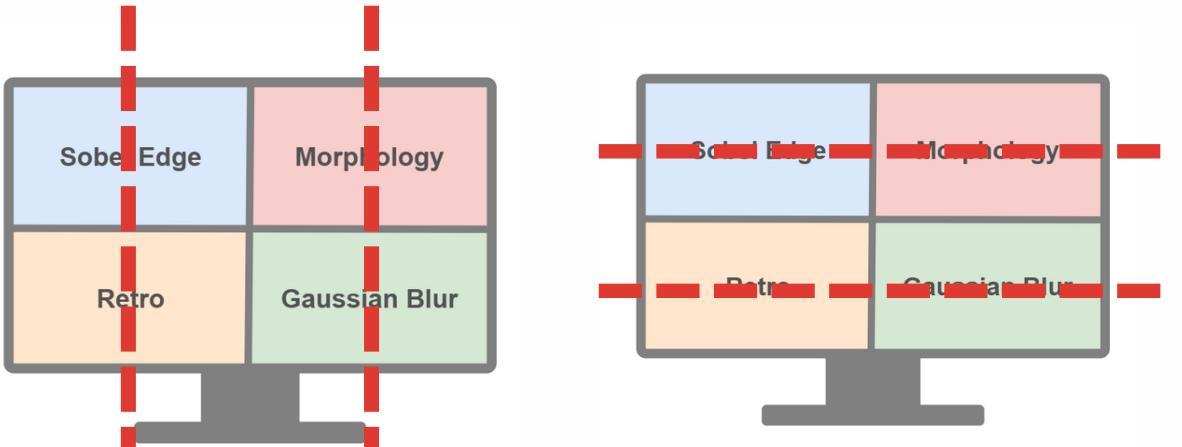
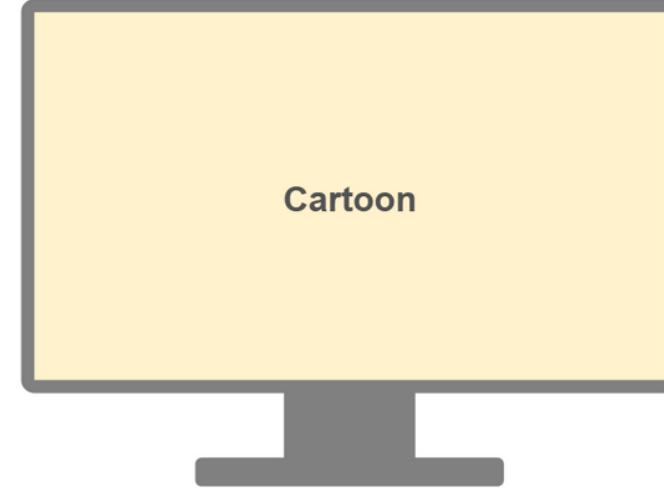


Image Processing

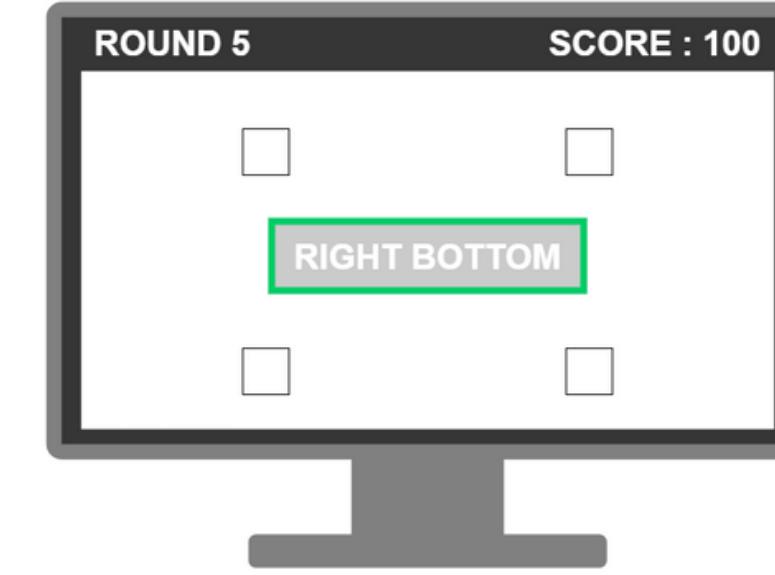
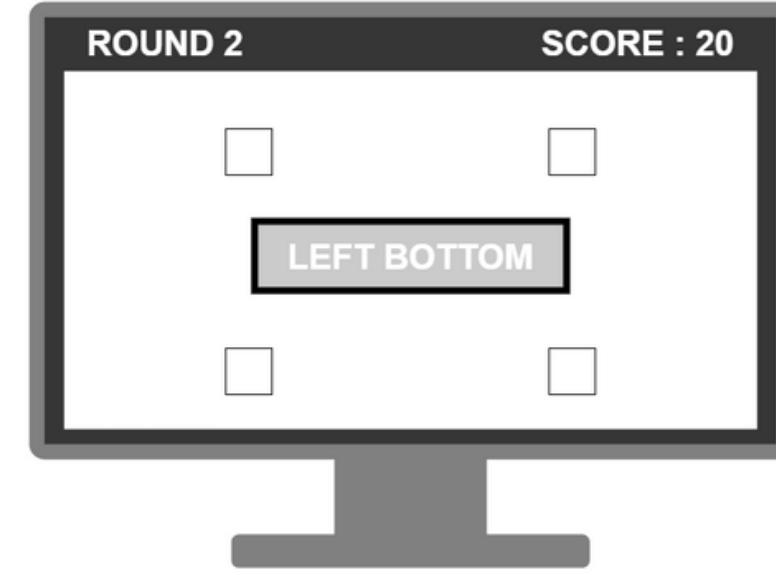
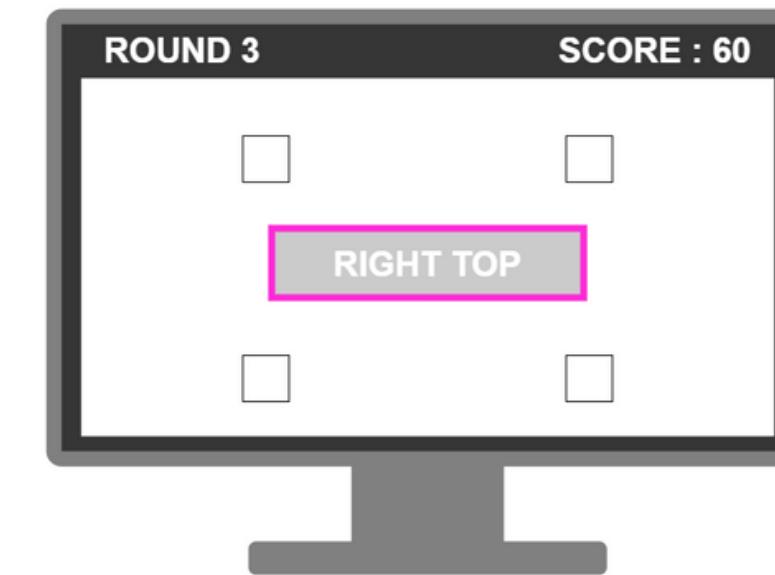
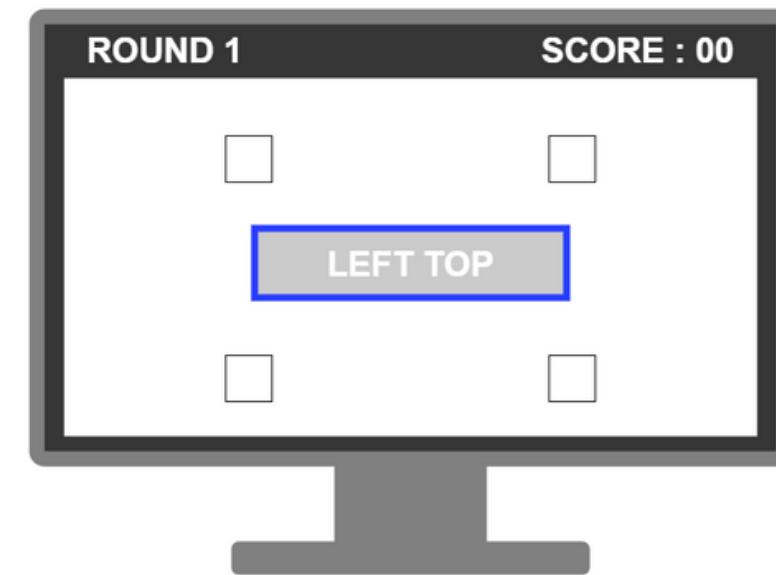
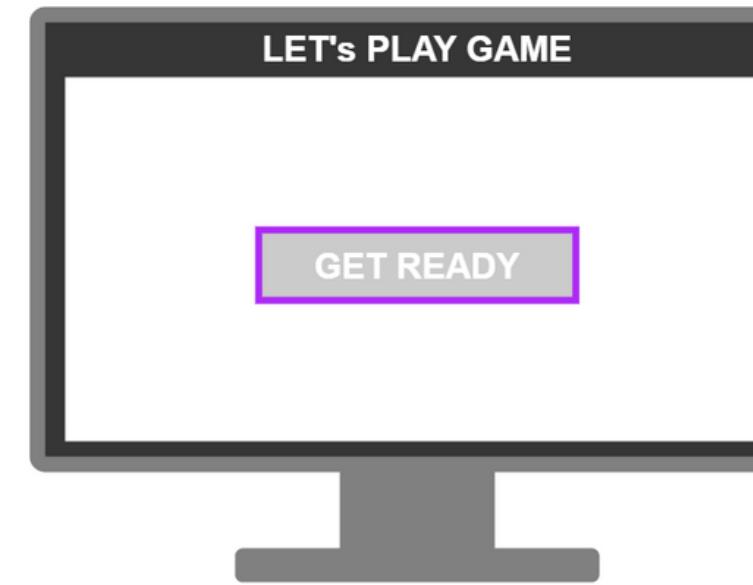
✓ Cartoon

- 역할 : 영상의 색상을 단순화하고 테두리를 진하게 그려서,
만화나 애니메이션 같은 느낌을 주는 필터
- 원리 : 두 가지 알고리즘을 결합
 - 색상 양자화 (Color Quantization): 픽셀값을 기준치(Threshold)에 따라
밝고/어둡게 이분화하여 단색(Flat Color) 처리
 - 윤곽선 강조(Edge Highlighting): 인접 픽셀(우측, 하단)과의 색차를 계산,
차이가 크면 검은색 테두리 생성
 - PixelWindow3x3 : SRAM에서 주변 픽셀 데이터를 실시간 로드
: Cartoon_Filter가 바로 인접 픽셀(right, down)로 엣지 검출할 때 사용
- 색상 단계를 단순화하고 테두리를 진하게 강조하여, 만화를 보는 듯한 느낌으로 변환

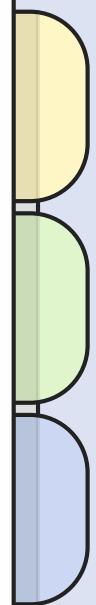
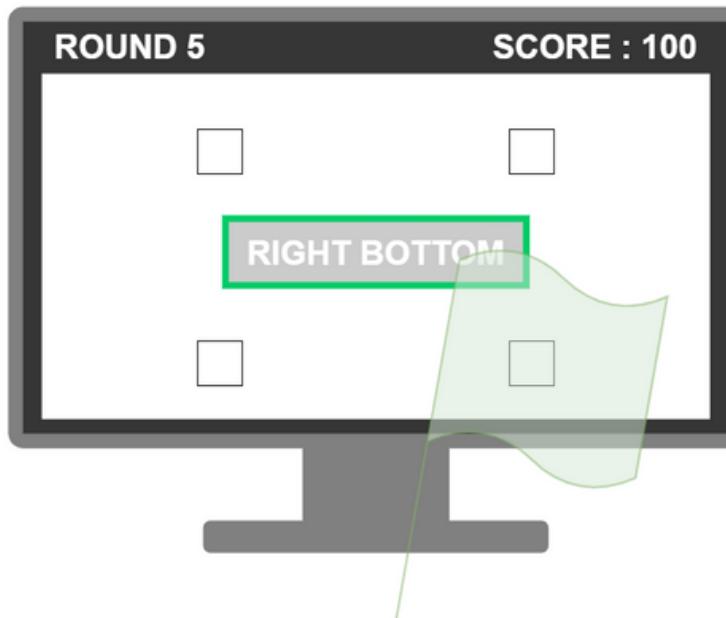
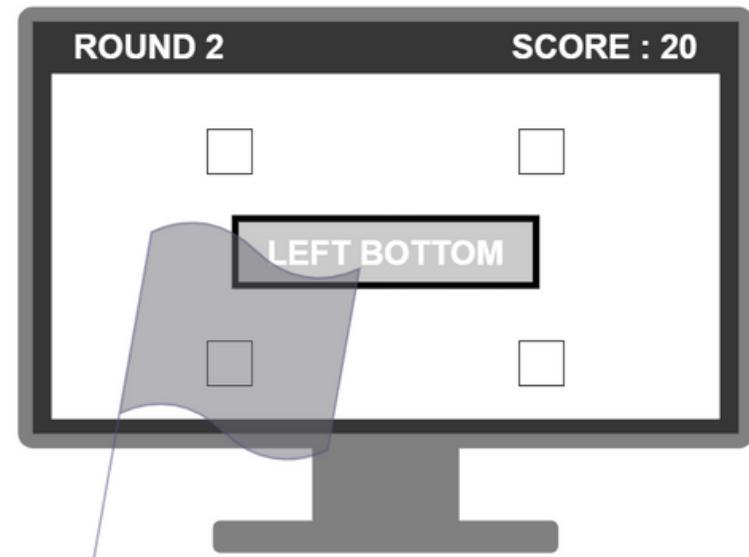
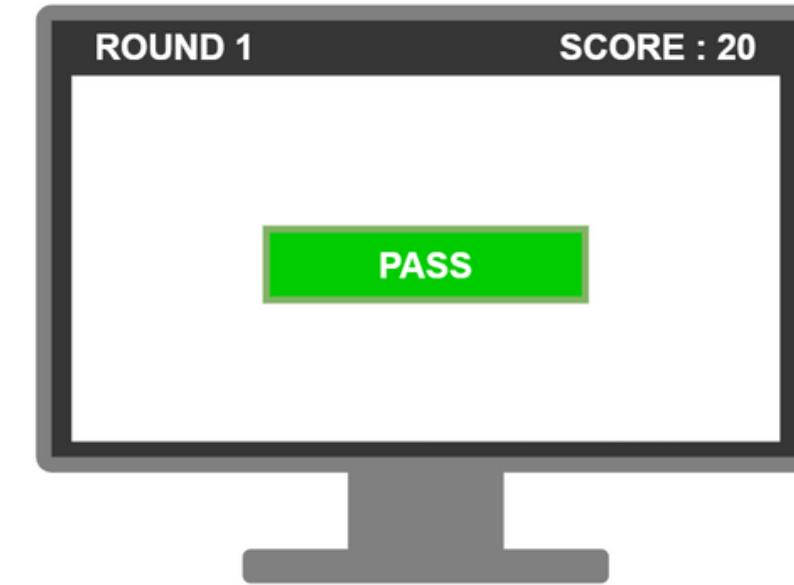
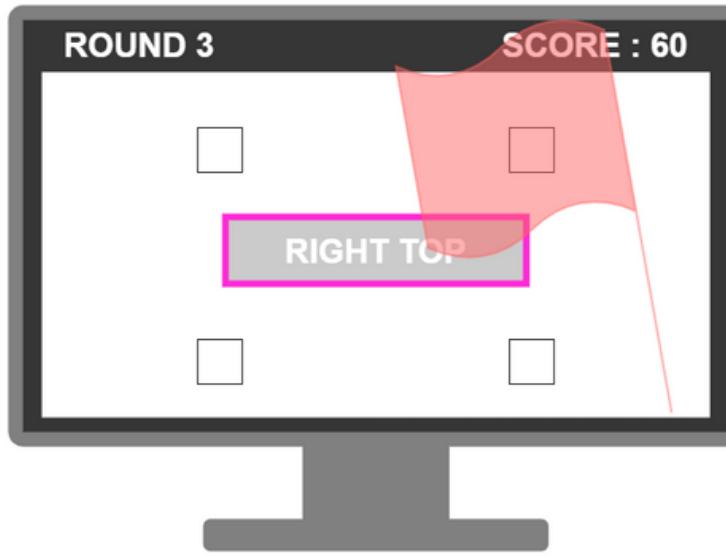
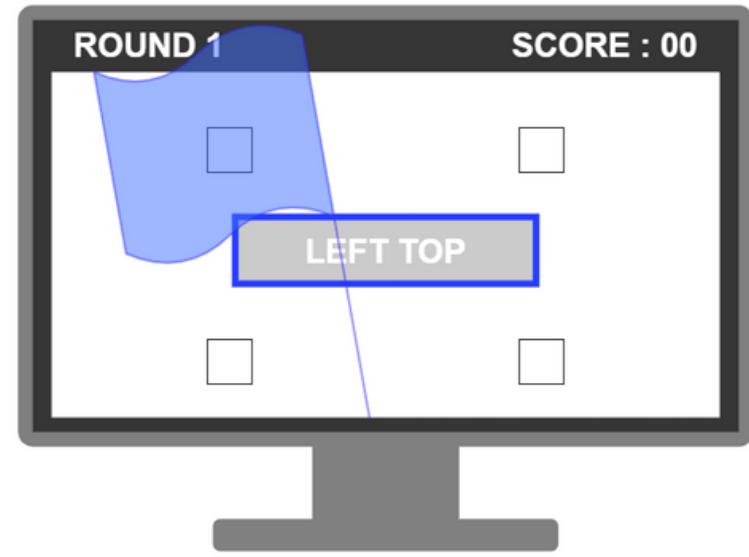


Game Controller

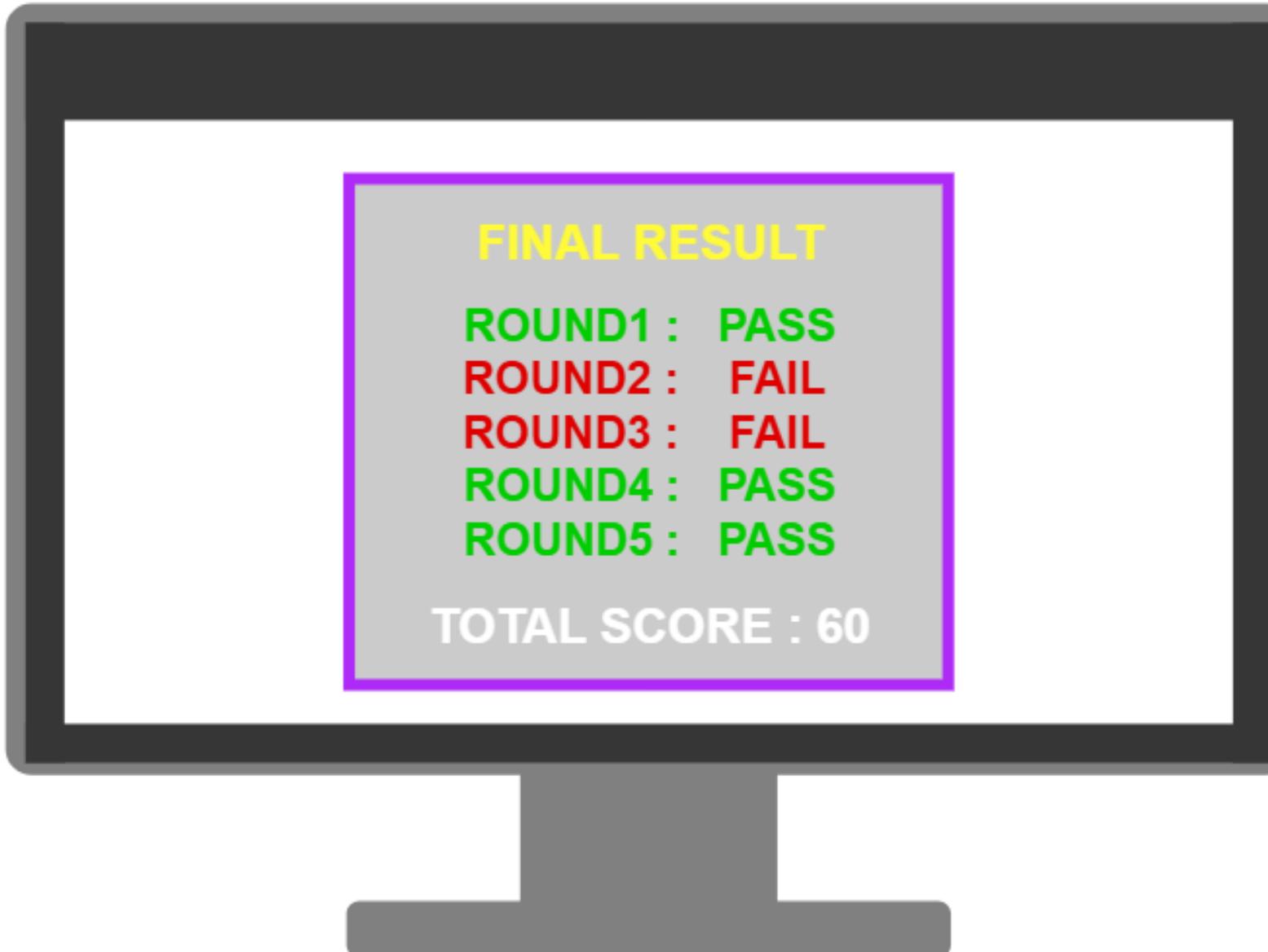
Mini Game



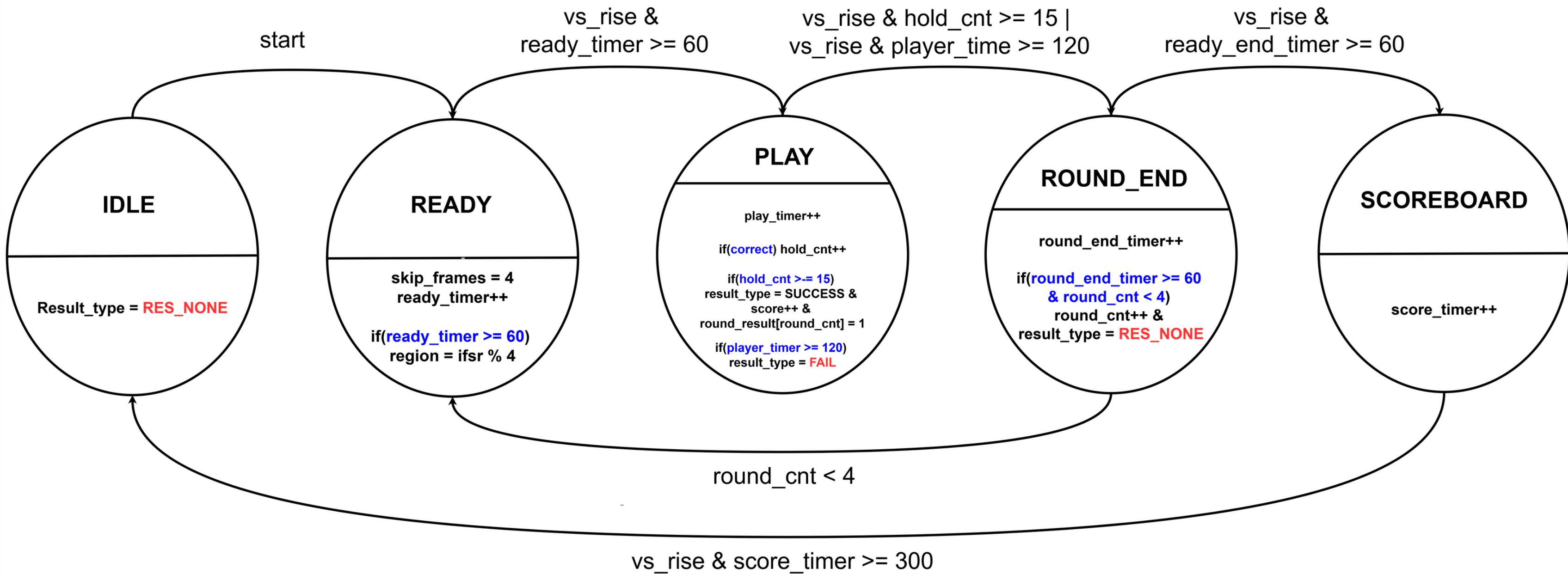
Mini Game



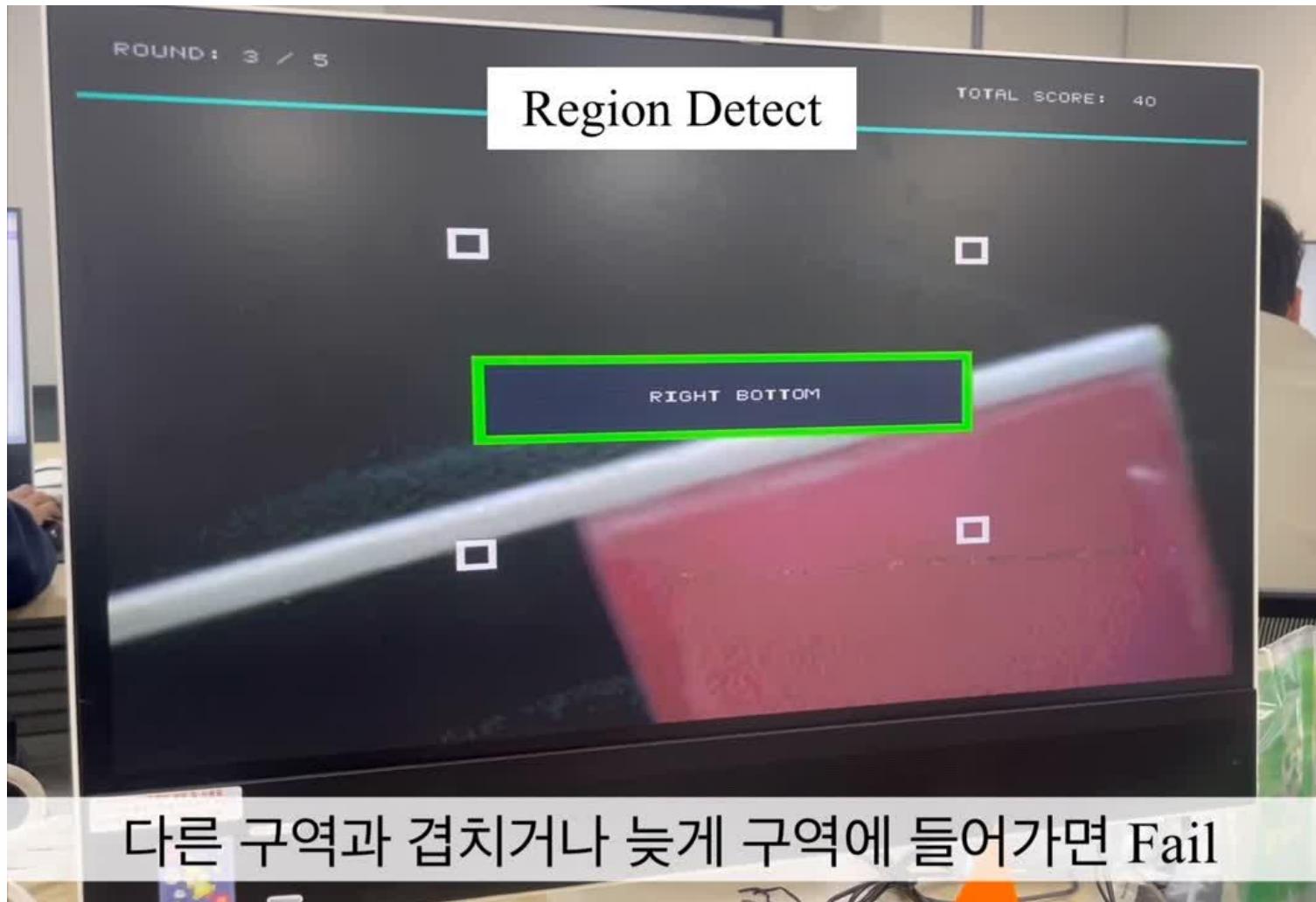
Mini Game



Game Controller FSM



Region Detect

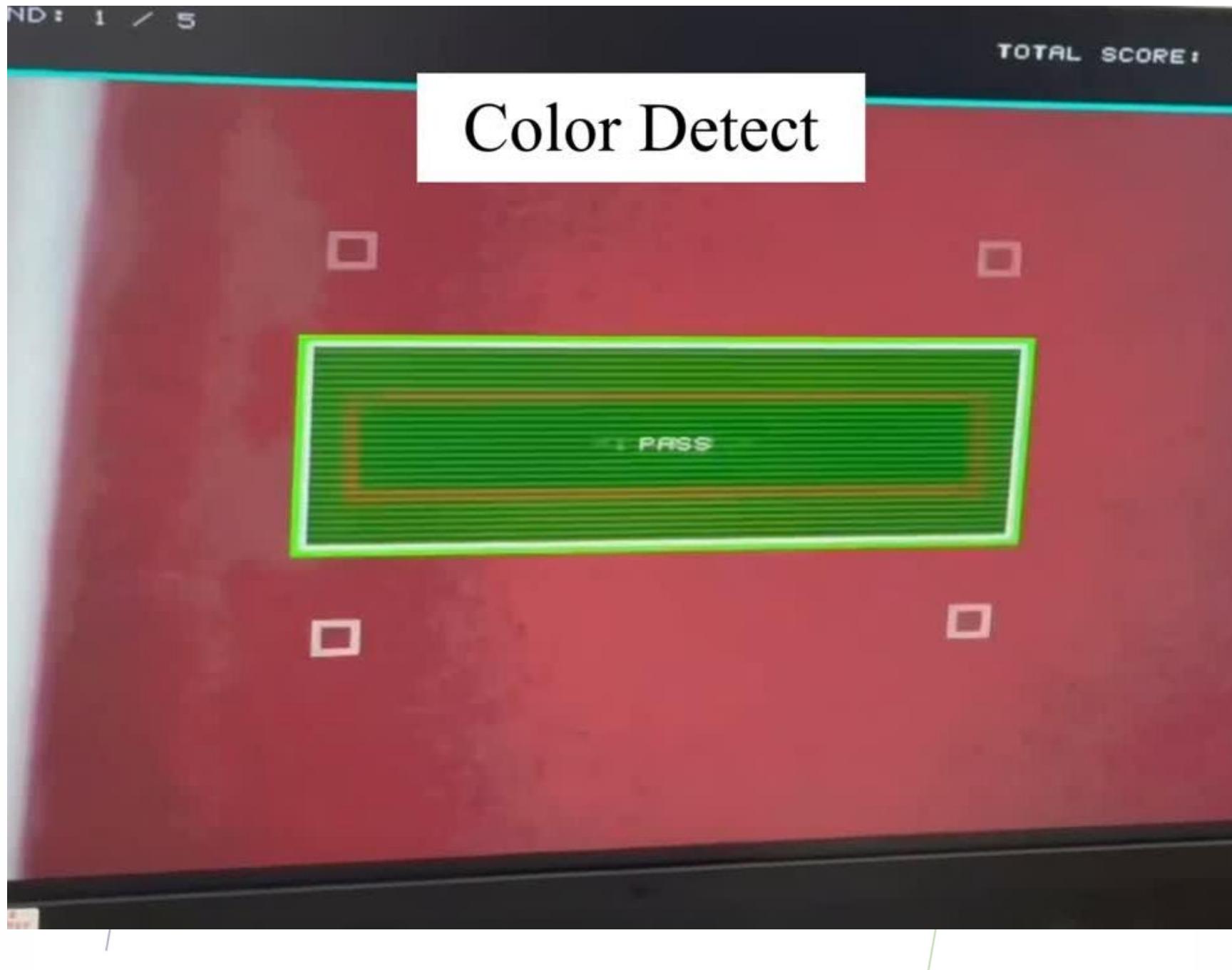


화면의 네 모서리 구역(Left Top, Right Top, Left Bottom, Right Bottom)을 정의하고, 물체가 해당 구역에 진입했는지를 좌표 비교를 통해 실시간으로 감지

Overlay 색이 r_in/g_in/b_in 입력을 왜곡하여 PASS가 랜덤하게 발생
→ 감지 영역과 Overlay 출력 좌표를 물리적으로 완전히 분리

```
parameter LT_X_START = 120,  
parameter LT_X_END   = 160,  
parameter LT_Y_START = 50,  
parameter LT_Y_END   = 90,  
  
parameter RT_X_START = 405,  
parameter RT_X_END   = 445,  
parameter RT_Y_START = 50,  
parameter RT_Y_END   = 90,  
  
parameter LB_X_START = 120,  
parameter LB_X_END   = 160,  
parameter LB_Y_START = 390,  
parameter LB_Y_END   = 430,  
  
parameter RB_X_START = 405,  
parameter RB_X_END   = 445,  
parameter RB_Y_START = 390,  
parameter RB_Y_END   = 430  
  
// Left-Top (LT)  
assign in_LT = (x_pixel >= LT_X_START && x_pixel < LT_X_END &&  
| | | | | y_pixel >= LT_Y_START && y_pixel < LT_Y_END);  
  
// Left-Bottom (LB)  
assign in_LB = (x_pixel >= LB_X_START && x_pixel < LB_X_END &&  
| | | | | y_pixel >= LB_Y_START && y_pixel < LB_Y_END);  
  
// Right-Top (RT)  
assign in_RT = (x_pixel >= RT_X_START && x_pixel < RT_X_END &&  
| | | | | y_pixel >= RT_Y_START && y_pixel < RT_Y_END);  
  
// Right-Bottom (RB)  
assign in_RB = (x_pixel >= RB_X_START && x_pixel < RB_X_END &&  
| | | | | y_pixel >= RB_Y_START && y_pixel < RB_Y_END);
```

Color Detect



OV7670은 노이즈가 많아서
RGB 값이 비슷해도
색으로 감지되는 현상 발생.

→ threshold : 각 색깔별로 조정
(카메라 화면에 나오는 색 바탕으로)

```
// R, G, B 중 가장 밝은 값 (빛의 세기, Value)
assign maxC = (r_in >= g_in && r_in >= b_in) ? r_in :
| | | | (g_in >= b_in) ? g_in : b_in;

// R, G, B 중 가장 어두운 값
assign minC = (r_in <= g_in && r_in <= b_in) ? r_in :
| | | | (g_in <= b_in) ? g_in : b_in;

assign diff = maxC - minC;

assign is_gray = (diff < 2);
assign is_black = (maxC <= 2);
assign is_red = (!is_gray && maxC == r_in);
assign is_green = (!is_gray && maxC == g_in);
assign is_blue = (!is_gray && maxC == b_in);
```

Troubleshooting

Troubleshooting

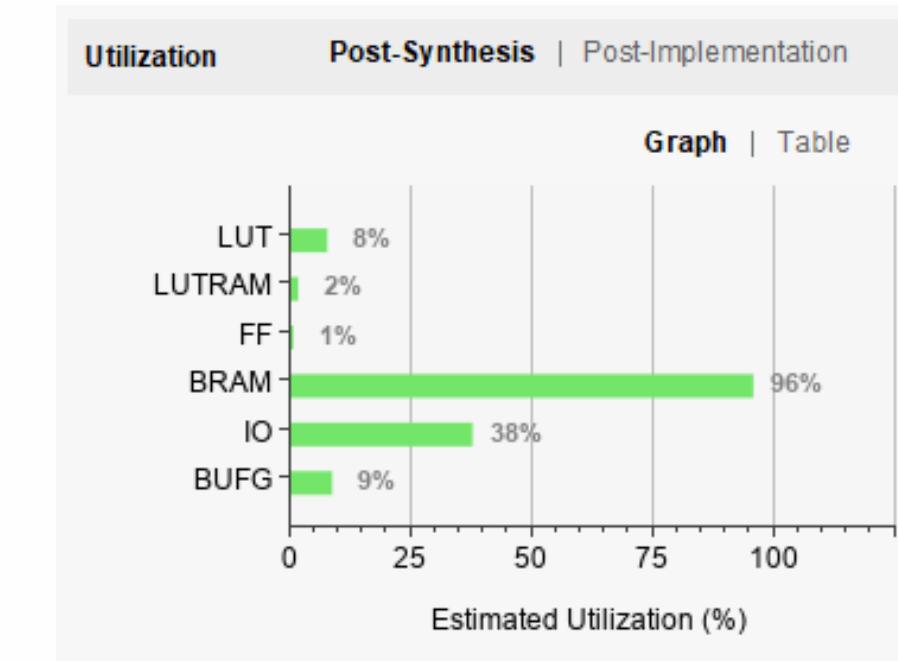
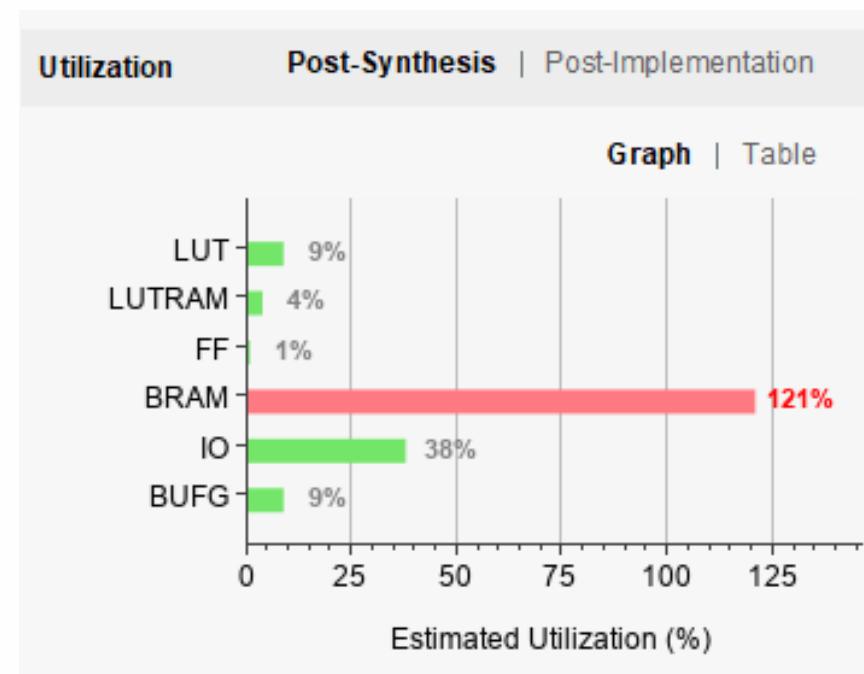
✓ Mini Game : Timing Mismatch & Synchronization Issue

- Issue: Multi-Clock Domain & Sampling Fail
 - pclk(Cam), sys_clk(VGA), clk 혼용으로 인한 **CDC(Clock Domain Crossing)** 문제
 - Vsync 기준의 FSM 상태 전이 시점과 Object Detect 신호(Pass)의 타이밍 불일치로 정답 인식 실패 (짧은 Pulse로 인한 Sampling Miss)
- Solution: Synchronization & Signal Extension
 - Clock Unification: 불안정한 FSM 동작 **클럭을 sys_clk로 단일화**하여 VGA 타이밍과 동기화
 - Signal Holding: Pass 신호 감지 시 hold_cnt를 적용, **신호 유지 시간(Pulse Width)**을 늘려 FSM 샘플링 마진 확보

→ 비동기 신호 간 Timing Margin 확보 및 게임 동작 안정성 향상

Troubleshooting

✓ PixelWindow3x3 & Cartoon Filter



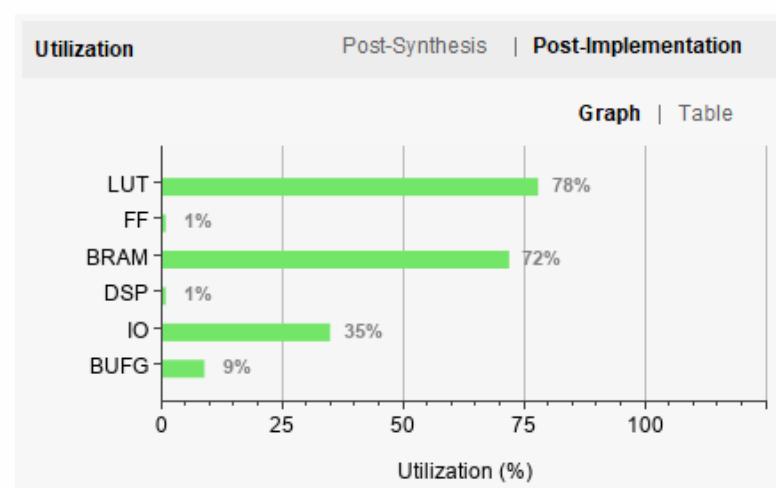
라인 버퍼(Line Buffer)를 활용하여 최소한의 행(Row) 데이터만
BRAM에 저장

PixelWindow3x3 모듈 설계: SRAM(BRAM)에서 데이터를 불
러와 윈도우 레지스터(pix00 ~ pix22)를 순차적으로 업데이트하
는 FSM(Finite State Machine) 구현 & 클럭 사이클에 맞춰 윈도
우가 한 칸씩 이동(Shift)하며 엣지 검출 연산 수행

- Frame Buffer 한계: Cartoon Filter의 엣지 검출을 위해
프레임 버퍼(Double Buffering) 사용 시도
- 자원 부족: 고해상도 이미지 저장 시 FPGA 내부 BRAM 용
량 초과 발생

Troubleshooting

✓ Chroma Key



단독 모듈
테스트에서는
크로마키 합성이
정상적으로 동작

해당 모듈만으로도
LUT 사용량이 78%,
BRAM 사용량이 72%에
육박하는 높은 점유율

[원인 분석]

1. LUT(Look-Up Table) : **픽셀 단위의 실시간 색상 비교 연산과 조건문, 그리고 비디오 데이터 처리를 위한 로직이 복잡해지면서** 로직 셀(Logic Cell) 점유율이 급격히 상승

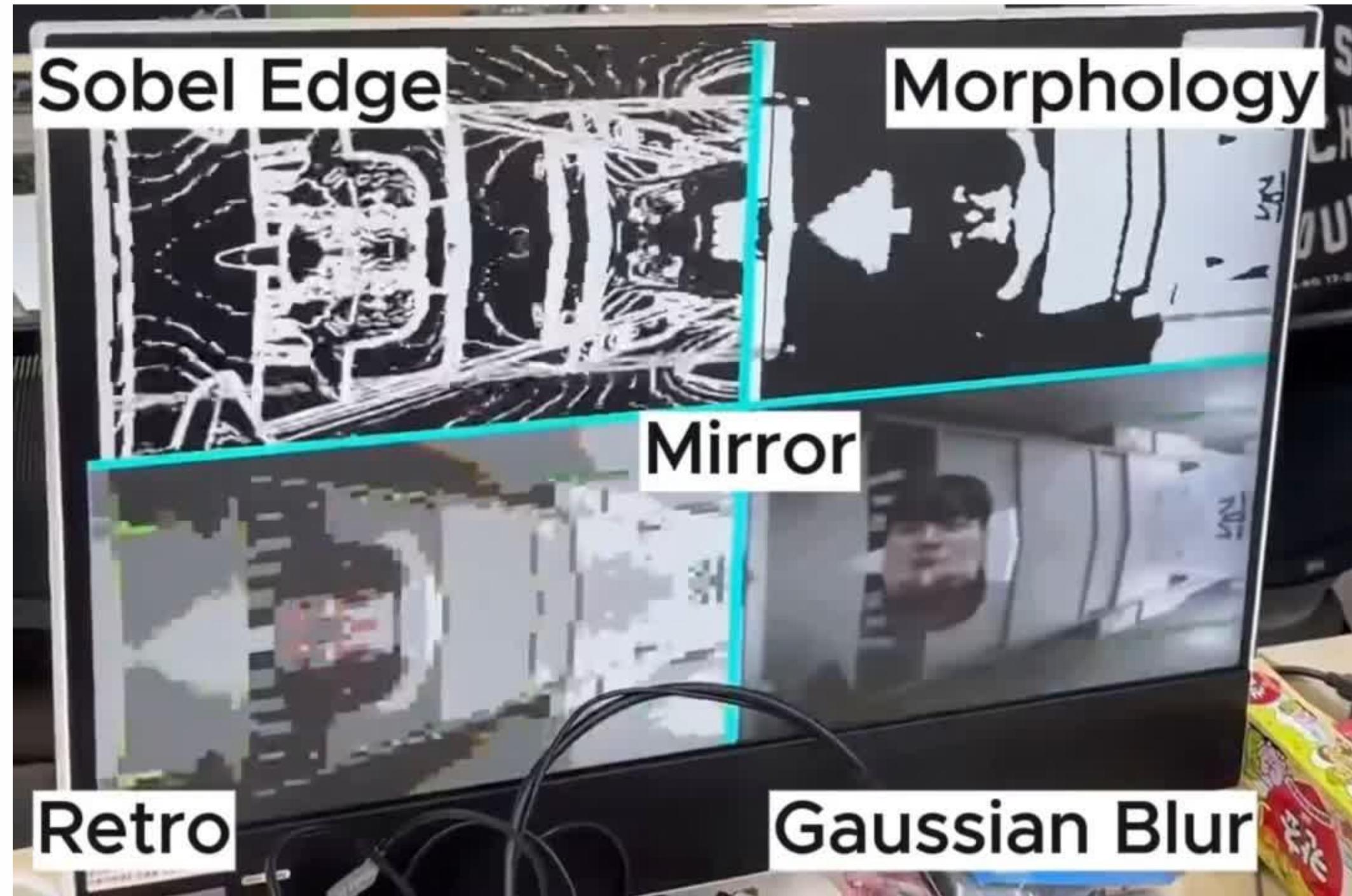
2. BRAM(Block RAM) : 외부 메모리(DDR)를 사용하지 않고
FPGA 내부의 BRAM만을 사용하여 고해상도 이미지와 비디오 프레임 버퍼를 처리하려다 보니,
가용 메모리 용량의 한계에 도달

→ 시스템의 전체적인 안정성과 필수 기능의 원활한 동작을
보장하기 위해, 크로마키 기능은 최종 통합본에서 제외

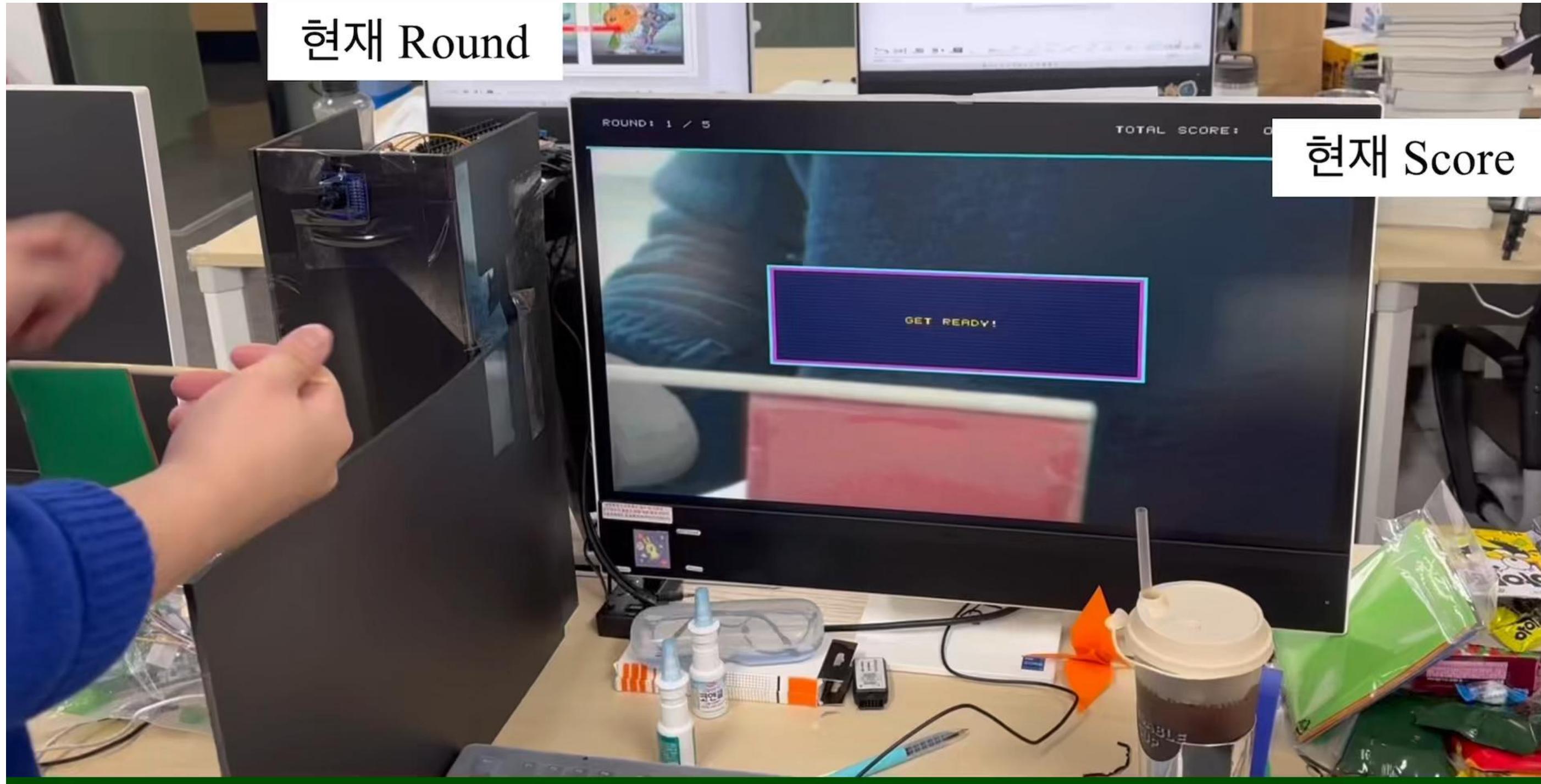
시스템 통합 과정에서 **FPGA 리소스 부족 현상(Resource Overflow)**이 발생

Demo

Demo



Demo



Conclusion

Conclusion

✓ 김은성

조장, Retro/Mirror Filter, Display Overlay, Chroma Key

LUT와 BRAM 부족이라는 자원 한계에 직면하며, 효율적인 하드웨어 리소스 관리의 중요성을 깨달았습니다. 불안정한 색상 인식 문제를 해결하기 위해 수없이 반복했던 과정은 무척 힘들고 아쉬움도 남았지만, 이를 통해 엔지니어에게 필요한 끈기를 배웠습니다. 또한 조장으로서 프로젝트를 이끌며, 기술적 난관을 함께 풀기 위한 솔직한 소통과 체계적인 코드 관리가 성공적인 협업의 필수 조건임을 깊이 깨닫게 되었습니다.

✓ 김재우

Top, Gaussian/Sobel Edge Filter, Game Core Debug, SCCB

Game Core를 디버깅하고 Top을 설계하면서 VGA신호들 중 동기화 신호인 Vertical Sync, Horizontal Sync와 같은 신호들의 Timing 제어, DE, X,Y Pixel에 대한 처리와 같은 중요 신호들의 처리 방식을 익혔으며, 또한 SCCB제어를 완성하며 이전 프로젝트에서 진행하면서 부족했던 I2C 제어에 대한 개념을 조금 더 확립할 수 있는 기회가 되어서 너무 좋았습니다.

Conclusion

✓ 유지훈

Cartoon Filter, 게임 구현

다양한 필터(Cartoon, Pixel Window, Sobel 등)를 FPGA에서 직접 구현하며 실시간 영상 처리에서는 프레임 저장 방식보다 스트리밍 기반 Pixel Window 구조가 훨씬 효율적이라는 것을 확인하였다.

전체 프로젝트를 통해 FPGA의 제한된 자원(BRAM, LUT 등) 안에서도 구조를 단순화하고 병렬 처리 특성을 활용하면 실시간 영상·게임 시스템을 충분히 구현할 수 있다는 것을 체감하였다.

✓ 조현우

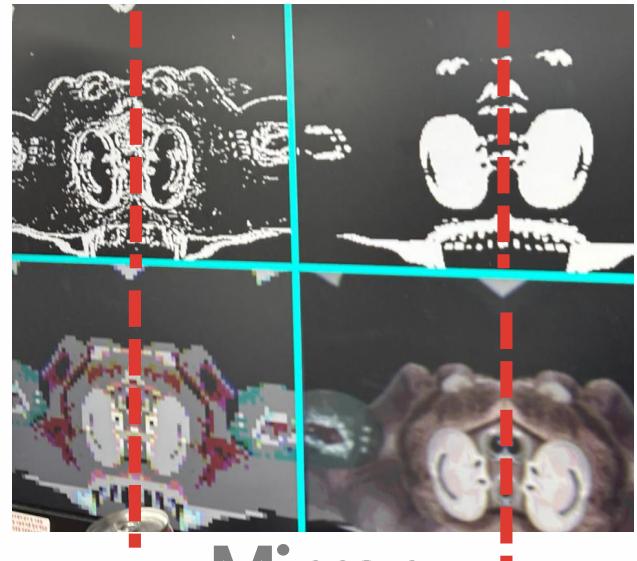
Morphology Filter, Color Detect

VGA Architecture System을 구현하면서 영상 처리에서는 허용된 Resource를 생각하면서 System을 Design 해야한다는 것을 다시 한 번 깨달았습니다. 또한 Image Processing 과정을 직접 적용해보면서, 단순히 동작하는 시스템을 만드는 것을 넘어 화질을 개선하고 원하는 효과를 구현하기 위해 어떤 알고리즘과 구조를 선택하고 최적화해야 하는지도 이해할 수 있었습니다.

특히 실시간 연산 처리 환경에서는 연산 효과 파이프라인 구조 설계가 매우 중요하다는 것을 체감하였고 이번 경험은 더 복잡한 영상 처리 시스템 구현시 중요한 기반이 될 것이라고 생각합니다.

감사합니다.

Filter



Mirror



Sobel Edge

Morphology



Retro

Gaussian Blur



Cartoon



Game

