# FPGA를 활용한 실시간 이미지 프로세싱

OV7670 FPGA Image Processing System

김동혁

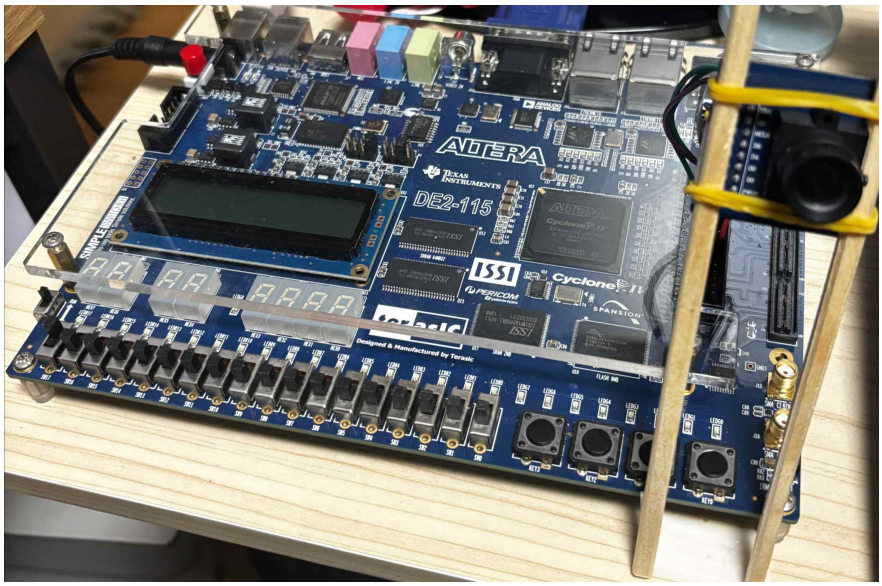
## 프로젝트의 목적

이 프로젝트의 가장 핵심적인 목적은, FPGA를 사용하여 카메라로부터 실시간으로 영상을 입력받아, 다양한 알고리즘을 하드웨어로 직접 구현 및 처리하고, 그 결과를 VGA 모니터로 출력하는 디지털 비디오 처리 시스템을 구축하는 것입니다.

## 프로젝트의 핵심 목표

1. 실시간 이미지 처리 능력의 하드웨어 구현
2. 다양한 비전 알고리즘의 적용 및 비교
3. FPGA 시스템 설계 능력의 종합적 증명

## 프로젝트 핵심 부품



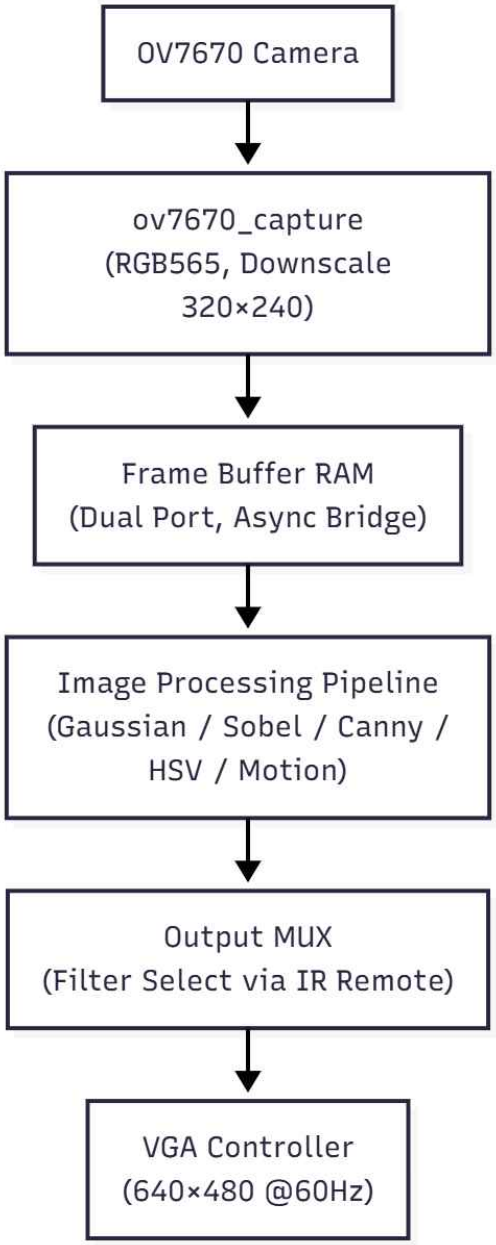

**테스트 환경**

**FPGA 보드 : DE2-115 Cyclone IV E**

**합성 프로그램 ; Quartus Prime 24.1**

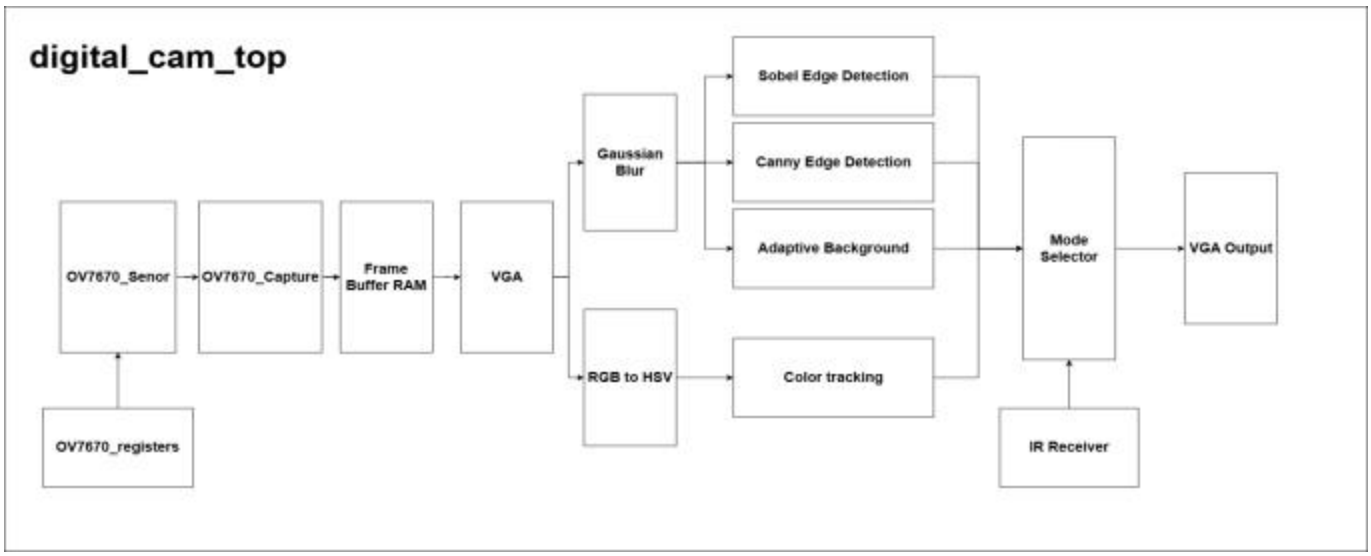
**시뮬레이션 프로그램 : Questa FSE**

시스템 구성도 (System Architecture)




| 구분 | 설명 |
|---|---|
| **카메라 인터페이스** | OV7670 모듈을 I2C로 초기화, RGB565 모드로 설정 |
| **이미지 캡처 및 축소** | 8bit×2 → 16bit RGB565 조합, 2x2 평균 필터로 320×240 축소 |
| **프레임 버퍼** | 듀얼 포트 RAM으로 구성, 클럭 도메인 분리 |
| **영상 처리 모듈** | Grayscale / Gaussian / Sobel / Canny / HSV / Adaptive Background |
| **VGA 출력** | 640×480 @60Hz로 업스케일 및 디스플레이 |
| **IR 리모컨 제어** | 실시간 필터 모드 변경 및 파라미터 조정 |

전체 로직 다이어그램




**각 필터 모듈은 독립 파이프라인 스테이지로 동작하며, 모든 단계가 동시에 실행됨.**

| 시스템 동작 요약 |
|---|
| 1. OV7670 카메라에서 입력된 RGB565 영상 데이터를 Dual-Port RAM을 통해 VGA 클럭 도메인으로 전달합니다. |
| 2. 각 영상 프레임은 Gaussian, Sobel, Canny, HSV 변환, Adaptive BG 등 여러 필터 모듈을 순차적으로 거치며 처리됩니다. |
| 3. 사용자는 Mode Selector를 통해 원하는 필터 (엣지, 컬러 추적, 적응형 배경 등)를 실시간 선택할 수 있습니다. |
| 4. 최종 처리 결과는 VGA로 출력되어 즉시 화면에 표시됩니다. |

| 입출력 구조 요약 |
|---|
| 입력: OV7670 카메라 (RGB565, 30fps, PCLK=24MHz) |
| 버퍼링: Dual-Port Frame Buffer RAM (클럭 도메인 분리, VGA 25MHz) |
| 처리: GrayScale → Gaussian → Sobel/Canny/Adaptive BG, HSV → Color Tracking |
| 출력: VGA 640×480 @ 60fps |
| 데이터 흐름: 1 px/clk 스트리밍 처리, 전체 지연 약 9clk (≈0.18μs) |

각 모듈의 구현 원리 및 동작

`digital_cam_top` (최상위 모듈)
모든 하위 모듈(카메라, 메모리, 필터, VGA)을 총괄하고, IR 리모컨 신호에 따라 실시간으로 영상 처리 파이프라인을 제어하는 시스템의 두뇌 역할을 수행합니다.

`ov7670_controller` (카메라 제어)
I2C 통신 프로토콜을 구현하여 OV7670 카메라 모듈의 내부 레지스터를 설정하고, 원하는 해상도(640x480)와 컬러 포맷(RGB565)으로 동작하도록 초기화합니다.

`ov7670_capture` (입력 처리)
카메라의 Raw 데이터 스트림을 받아 16비트 RGB565 픽셀로 재조립하고, 2x2 Averaging Filter를 통해 실시간으로 320x240 해상도로 다운 스케일링하여 메모리 사용량을 최적화합니다. (내부 BRAM 용량 문제로 인해)

`frame_buffer_ram` , 'frame_buffer_ram_11k'(프레임 버퍼)
듀얼 포트 RAM으로 구현되어, 서로 다른 클럭(카메라 pclk, VGA clk)으로 동작하는 쓰기(입력)단과 읽기(출력)단을 분리하여 데이터 충돌 없이 안정적으로 연결하는 비동기 브릿지 역할을 수행합니다.

`vga_640` (VGA 출력)
VESA 표준에 맞는 640x480@60Hz의 수평/수직 동기 신호를 생성하며, 프레임 버퍼의 주소를 조작하여 읽기 과정에서 실시간으로 영상을 2배 업스케일링합니다.

`gaussian_3x3_gray8` (가우시안 블러)
3x3 가우시안 커널을 적용하여 이미지의 노이즈를 부드럽게 제거하며, 특히 Canny/Sobel 같은 엣지 검출 필터의 전처리 단계로 사용되어 성능을 향상시킵니다.

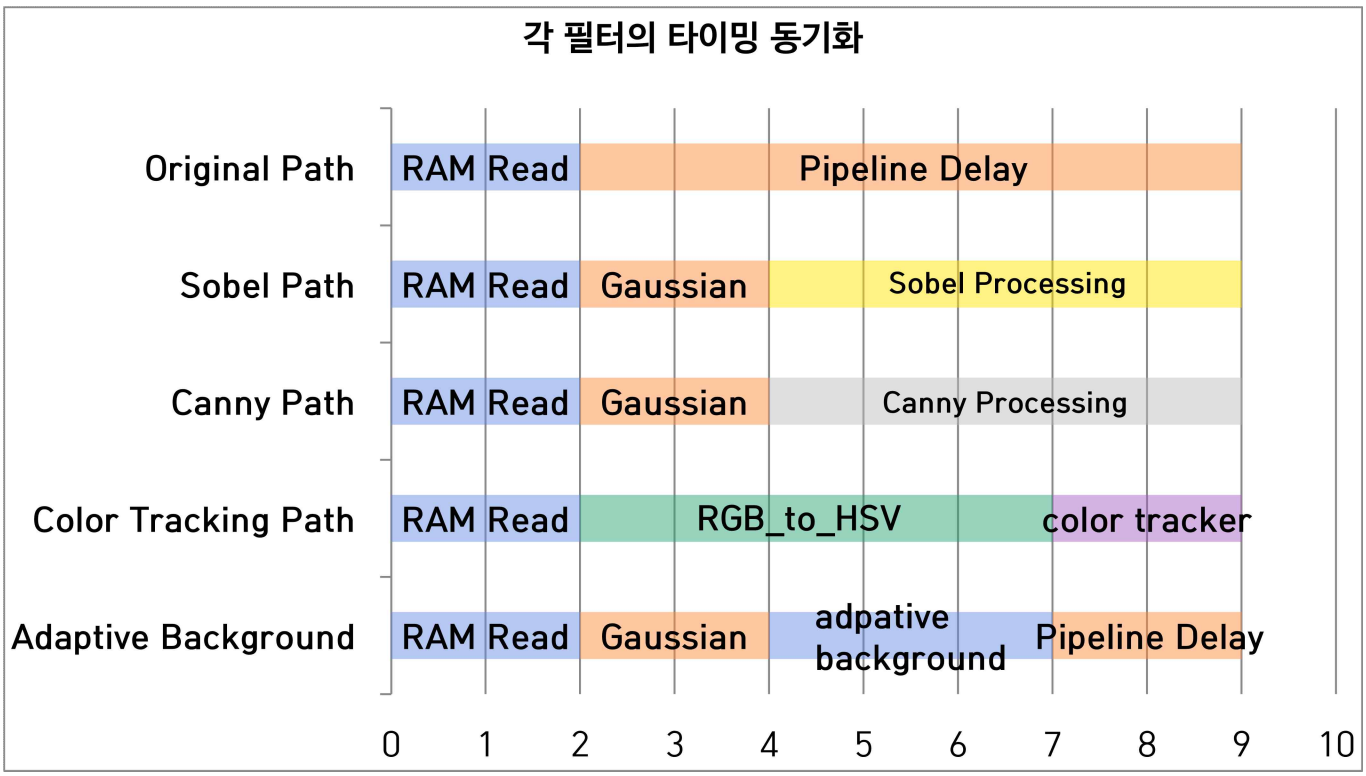
`sobel_3x3_gray8` (소벨 엣지)
3x3 Sobel 연산자를 이용하여 이미지의 X, Y축 그래디언트를 계산하고, 그 크기를 통해 픽셀 밝기가 급격하게 변하는 엣지(윤곽선)를 검출합니다.

`canny_3x3_gray8` (캐니 엣지)
비최대 억제(NMS)와 이중 임계값(Double Threshold) 등 복합적인 알고리즘을 통해, Sobel보다 더 가늘고 정확하며 노이즈가 적은 고품질의 엣지를 검출합니다.

`rgb_to_hsv` & `color_tracker` (컬러 트래킹)
조명 변화에 민감한 RGB 색상 공간을 HSV 공간으로 변환한 뒤, 미리 정의된 H/S/V 임계값 범위와 비교하여 특정 색상(빨강, 초록, 파랑)에 해당하는 픽셀만을 정확히 식별해냅니다.

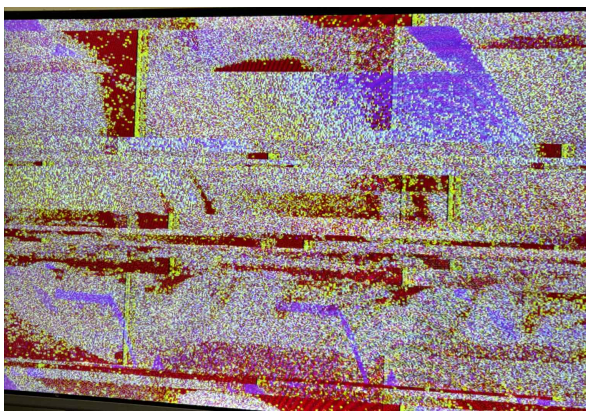
`adaptive_bac kground` (움직임 감지)
현재 영상과 배경 모델을 비교하여 움직이는 전경을 분리하며, 이중 학습률과 동적 임계값을 적용하여 조명 변화에 스스로 적응하고 멈춘 물체는 서서히 배경으로 흡수합니다.

'IR_RECEVER' (IR 리모컨 수신)
IR 수신 다이오드에서 들어오는, 시간 기반의 펄스 신호를 NEC 프로토콜에 맞춰 디코딩하여, 시스템이 인식할 수 있는 8비트 버튼 커맨드 코드로 변환하는 역할을 합니다.




**각 필터 모듈은 내부 파이프라인 깊이에 따라 고유한 처리 지연(latency)을 가지며,**
**시스템은 이를 고려해 지연 보상 파이프라인을 삽입하여 모든 출력이**
**동일한 클럭 타이밍(+9clk)에서 유효하도록 정렬하였다.**
**결과적으로 각 경로의 출력 픽셀이 VGA 프레임 타이밍에 정확히 매칭된다.**

**프로젝트 진행 중 각종 문제 및 해결**



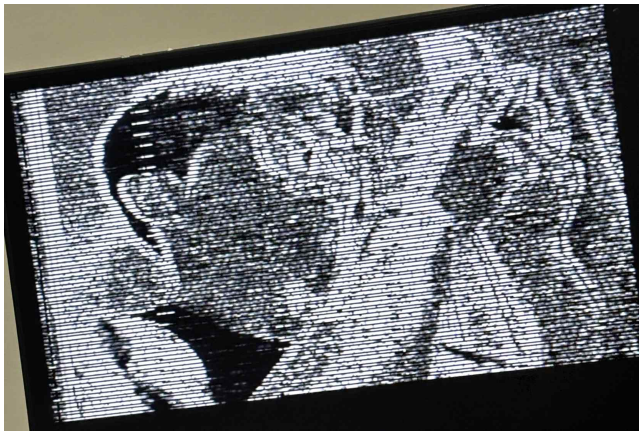
1) 카메라-메모리-VGA 간 클럭 비동기로 인한 프레임 깨짐

문제점

RAM은 읽기 지연(Read Latency)이 있는데, 주소 변경 직후 바로 데이터를 읽으려 함
카메라 쓰기 완료 전에 VGA 읽기 시작

해결 방안

메모리 지연 보정 추가 (FF 추가)
첫 프레임 완료 후 VGA 시작 (ov7670_vsync 신호를 활용)



2) 소벨필터 노이즈 증폭 문제와 필터 로직과 출력 로직간의 타이밍 오류

문제점

카메라 모듈의 노이즈 또한 소벨필터가 엣지로 인식하는 문제 발생.

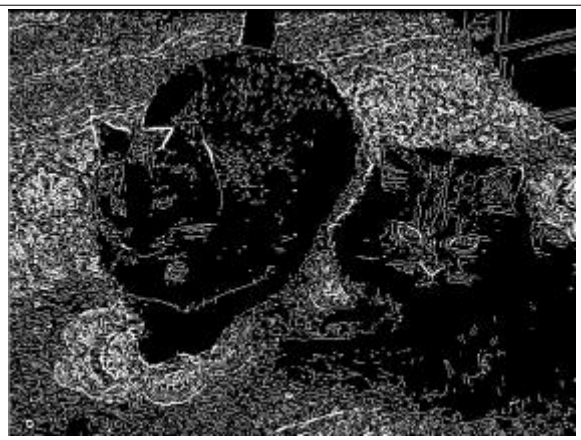
소벨필터 처리로 인한 지연을 고려하지 않고 VGA로 출력 타이밍 동기화 문제

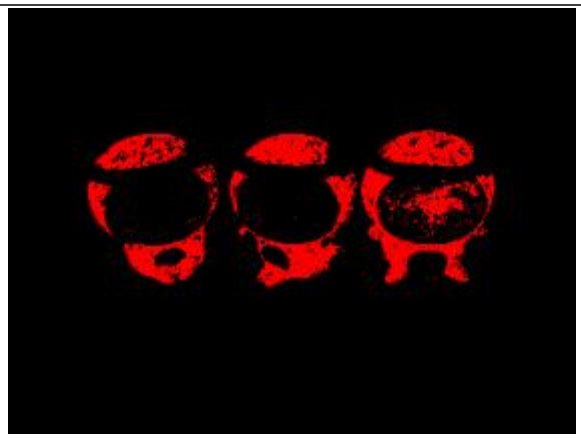
해결 방안

Grayscale, gaussian_blur 처리를 한 데이터를 소벨필터 입력으로 받음.

VGA신호를 가우시안 지연, 소벨필터 지연, 메모리 읽기 지연 등을 고려하여 파이프라이닝 지연설계

## 시뮬레이션 결과

<table>
<tr>
<td>  <br>원본 이미지</td>
<td>  <br>Gaussian Blur</td>
</tr>
<tr>
<td>  <br>Sobel</td>
<td>  <br>Canny</td>
</tr>
</table>

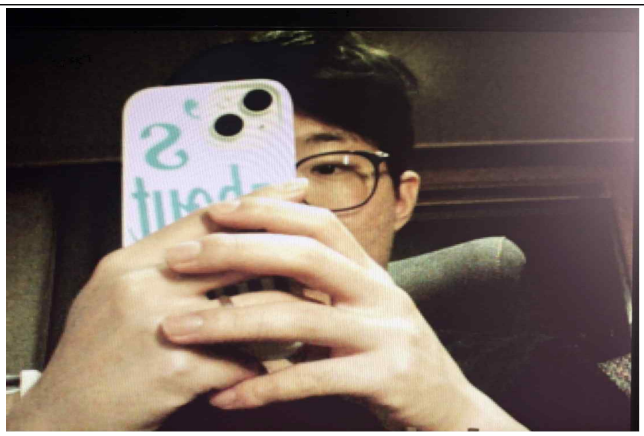
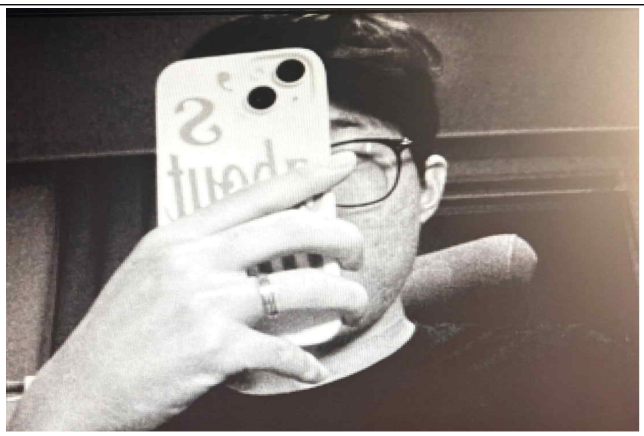
| | |
|---|---|
|  |  |
| 원본 이미지 | Color tracking |
|  |  |
| 배경 이미지 | Adaptive Backgruond |

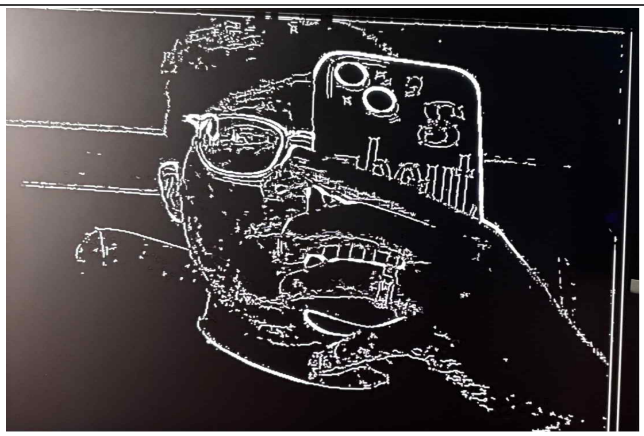
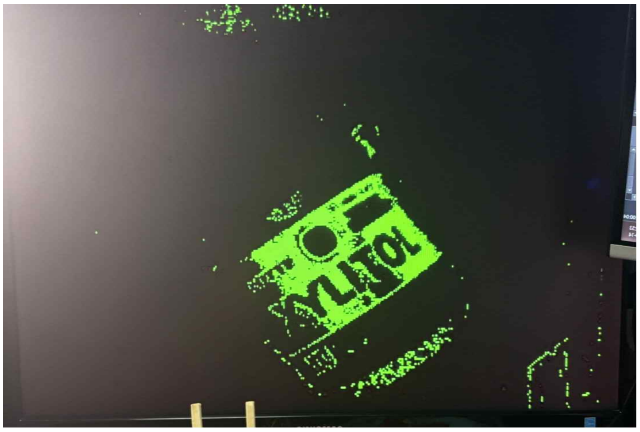
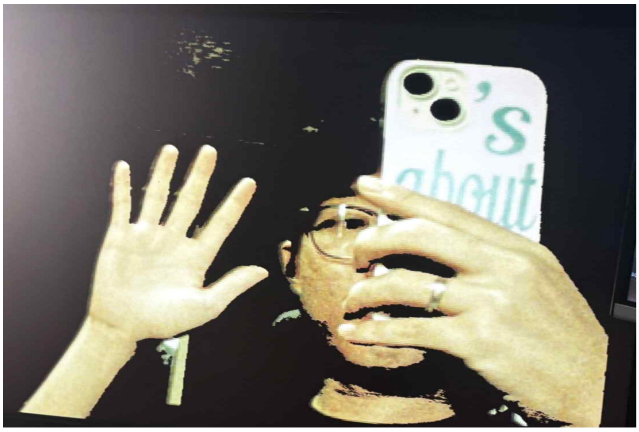
**프로젝트 결과 (Project Outcome)**

OV7670 영상 입력을 FPGA에서 실시간으로 처리,

VGA 640×480 해상도로 안정 출력 성공.

Gaussian, Sobel, Canny, HSV, Motion Detection 등 모든 필터 실시간 IR 리모컨 조작 가능.

| | |
|---|---|
|  |  |
| 원본 영상 | Gray Scale |

<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>Sobel</td><td>Canny</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>Color tracking</td><td>Adaptive Background</td></tr>
</table>

| | |
|---|---|
| Total combinational functions | 39,008 / 114,480 ( 34 % ) |
| Dedicated logic registers | 49,819 / 114,480 ( 44 % ) |
| Total memory bits | 1,929,252 / 3,981,312 ( 48 % ) |

**아쉬웠던 점과 수정해야 할 점**

당초 계획은 640×480 해상도의 원본 영상을 프레임 버퍼에 저장한 뒤, 필터 처리를 거쳐 출력하는 구조였습니다.

그러나 내부 BRAM 용량의 한계로 인해 전체 프레임을 저장하기 어려워, 2×2 디시메이션(Downsampling) 로직을 적용하여 320×240 해상도로 축소 후 처리하는 방식을 택했습니다.

이 과정에서 필연적으로 영상 세부 품질 저하(특히 엣지·텍스처 영역 손실) 가 발생하는 한계가 있었습니다.

이를 해결하기 위해 SDRAM을 활용한 외부 메모리 버퍼링 구조를 시도했으나,

SDRAM Controller의 타이밍·초기화·클럭 동기화 문제 등 여러 기술적 이슈가 발생하여

프로젝트 일정 내에 완전한 통합이 어려웠습니다.

결과적으로 실시간 동작 안정성을 우선시하여, BRAM 기반의 다운샘플링 방식을 최종 선택하였습니다.

**이 프로젝트를 진행하며 배운 점**

본 프로젝트를 통해 FPGA 기반의 영상처리 시스템을 처음부터 끝까지 직접 설계하며,
하드웨어에서의 데이터 흐름 제어와 멀티클럭 동기화의 중요성을 체득했습니다.
또한 소프트웨어 알고리즘을 하드웨어로 최적화하는 과정을 통해 연산 구조와 자원 효율의 균형을 이해하였고,
다양한 타이밍 문제를 논리적으로 추적·해결하면서 실제 시스템 수준의 문제 해결 능력을 크게 향상시켰습니다.