FPGA를 활용한 실시간 이미지 프로세싱

OV7670 FPGA Image Processing System

20211461 김동혁

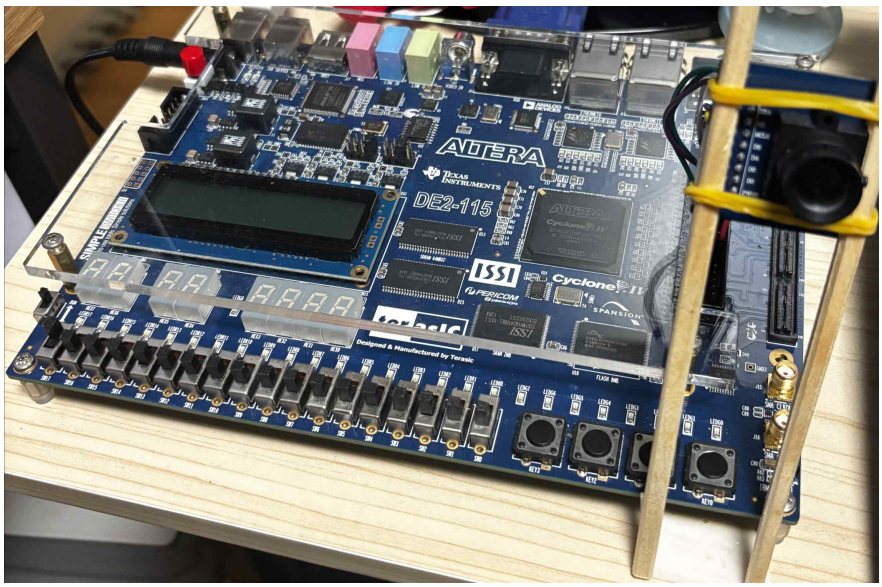
프로젝트의 목적

본 프로젝트는 FPGA를 이용해 카메라로부터 입력되는 실시간 영상을 하드웨어에서 직접 처리하고,
이를 VGA 디스플레이로 출력하는 디지털 영상처리 시스템을 구현하는 것을 목표로 했다.
소프트웨어 기반 영상처리에 비해 하드웨어 병렬 구조의 성능과 효율을 검증하고,
FPGA 상에서의 영상 데이터 흐름 제어 및 파이프라인 동작을 심층적으로 이해하는 데 중점을 두었다..

프로젝트의 핵심 목표

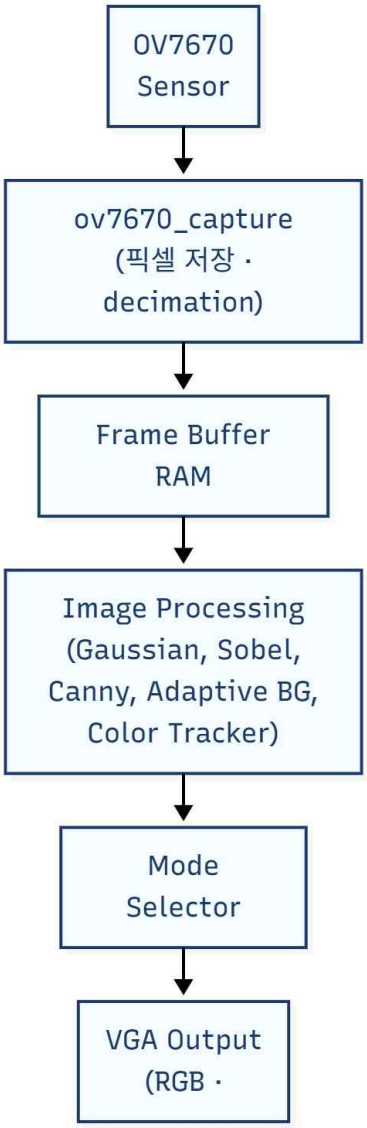
1. OV7670 카메라 영상의 실시간 하드웨어 처리 파이프라인 구현
2. FPGA 기반 영상처리 알고리즘(Gray, Gaussian, Sobel, Canny, HSV, Adaptive Background) 비교 및 최적화
3. 클럭 도메인 분리 및 파이프라인 지연 제어를 통한 안정적인 실시간 처리 구조 설계

프로젝트 핵심 부품



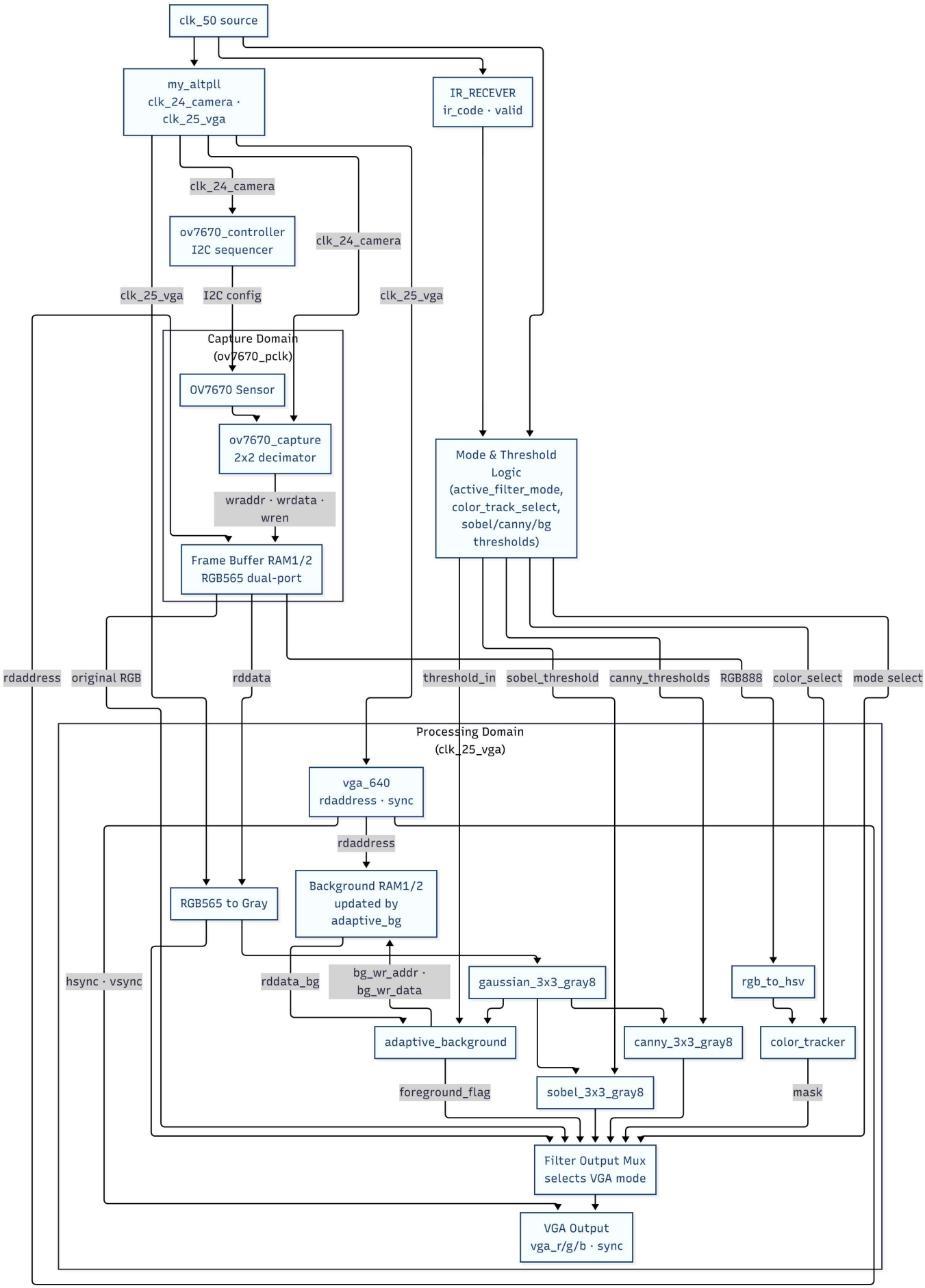
FPGA : DE2-115
카메라 모듈 : OV7670
tool : Quartus Prime Lite Edition
언어 : Verilog

시스템 구성도 (System Architecture)



구분	주요 기능
Camera Interface	OV7670 모듈을 I2C로 초기화하고 RGB565 모드로 설정
Capture & Downscale	8bit×2 입력을 16bit RGB565로 재조합 후, 2×2 Averaging Filter로 320×240 축소
Frame Buffer	Dual-Port RAM을 이용한 비동기 버퍼. PCLK과 VGA CLK 도메인 분리
Processing Module	Grayscale, Gaussian Blur, Sobel, Canny, HSV 변환, Adaptive Background Subtraction 등
VGA Output	640×480 @60Hz 해상도로 업스케일 및 디스플레이
IR Control	NEC 프로토콜 기반 IR 리모컨을 통해 필터 모드 및 파라미터 실시간 제어

전체 로직 다이어그램


clk_50 source
my_altpll
clk_24_camera · clk_25_vga
IR_RECEVER
ir_code · valid
clk_24_camera
ov7670_controller
I2C sequencer
clk_24_camera
clk_25_vga
I2C config
clk_25_vga
Capture Domain
(ov7670_pclk)
OV7670 Sensor
ov7670_capture
2x2 decimator
wraddr · wrdata · wren
Frame Buffer RAM1/2
RGB565 dual-port
Mode & Threshold Logic
(active_filter_mode, color_track_select, sobel/canny/bg thresholds)
rdaddress
original RGB
rddata
threshold_in
sobel_threshold
canny_thresholds
RGB888
color_select
mode select
Processing Domain
(clk_25_vga)
vga_640
rdaddress · sync
rdaddress
RGB565 to Gray
Background RAM1/2
updated by adaptive_bg
hsync · vsync
rddata_bg
bg_wr_addr · bg_wr_data
gaussian_3x3_gray8
rgb_to_hsv
adaptive_background
canny_3x3_gray8
color_tracker
foreground_flag
sobel_3x3_gray8
mask
Filter Output Mux
selects VGA mode
VGA Output
vga_r/g/b · sync

각 모듈의 구현 원리 및 동작

🔻 digital_cam_top (Top-Level)
시스템 전체의 제어 및 모듈 간 데이터 흐름을 관리한다.
카메라 입력, 메모리 버퍼, 영상처리 필터, VGA 출력, IR 리모컨 제어를 통합해
실시간 파이프라인의 핵심 제어 역할을 수행한다.

🔻 ov7670_controller
I2C 통신을 통해 OV7670 카메라의 내부 레지스터를 초기화하고
해상도(640×480) 및 색상 포맷(RGB565)을 설정한다.

🔻 ov7670_capture
카메라의 Raw 데이터 스트림을 RGB565 포맷으로 조립한 후,
2×2 평균 연산을 적용해 320×240 해상도로 축소한다.
이는 BRAM 용량 제약을 극복하기 위한 실시간 다운샘플링 구조이며,
픽셀 클럭(PCLK) 도메인에서 동작한다.

🔻 frame_buffer_ram
듀얼 포트 구조로, 카메라의 쓰기 클럭과 VGA의 읽기 클럭을 분리한다.
이 모듈은 두 도메인 간 데이터를 안전하게 전달하기 위한 비동기 브릿지 역할을 수행한다.

🔻 vga_640
640×480@60Hz 신호를 생성하고,
프레임 버퍼 데이터를 읽어 실시간 업스케일링 후 디스플레이한다.

🔻 gaussian_3x3_gray8
3×3 Gaussian 커널을 적용해 노이즈를 억제하며,
Sobel 및 Canny 필터의 전처리 단계로 사용된다.

🔻 sobel_3x3_gray8
X, Y 방향 그래디언트를 계산하여 밝기 변화가 급격한 엣지 영역을 검출한다.

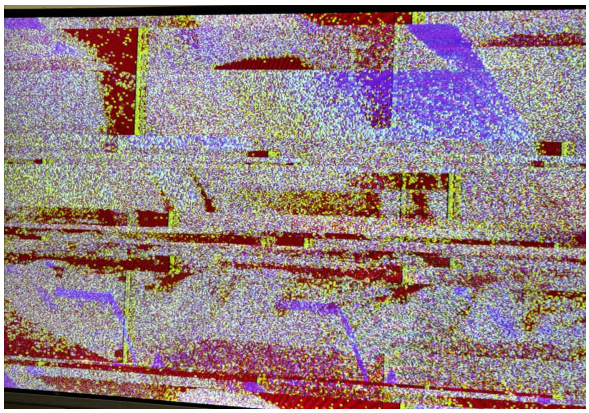
🔻 canny_3x3_gray8
비최대 억제(NMS)와 이중 임계값(Double Threshold)을 적용해
더 세밀하고 노이즈에 강한 엣지를 추출한다.

🔻 rgb_to_hsv & color_tracker
RGB 영상을 HSV로 변환하여 색상(H), 채도(S), 명도(V) 범위를 비교하고,
특정 색상의 픽셀만을 선택적으로 검출한다.

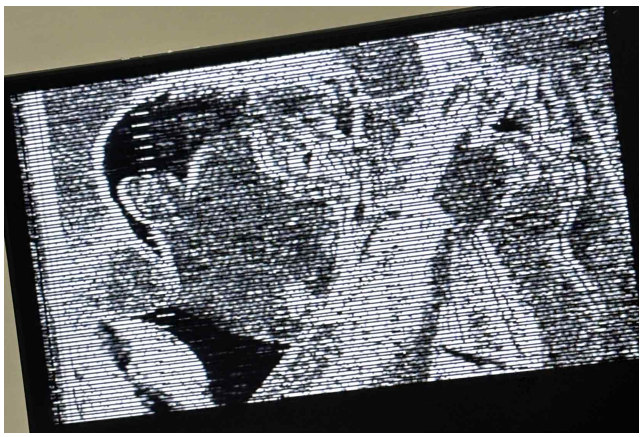
🔻 adaptive_background
현재 프레임과 배경 모델을 비교해 움직임을 감지하며,
조명 변화에 적응하는 이중 학습률 기반 구조로 설계되었다.

🔻 ir_receiver
NEC 프로토콜에 따라 IR 신호를 디코딩하여
시스템 제어용 8비트 명령 코드로 변환한다.

프로젝트 진행 중 각종 문제 및 해결



1. 프레임 깨짐 현상
원인: VGA가 카메라 쓰기 완료 전 데이터를 읽기 시작함
해결: 프레임 완료 신호(frame_done) 이후 VGA 동작 시작, RAM 지연 보정 FF 추가



문제 2. Sobel 필터 노이즈 증폭
원인: Gaussian 전처리 미적용, 타이밍 불일치
해결: Gaussian Blur 추가, 파이프라인 지연 보정으로 타이밍 일치화

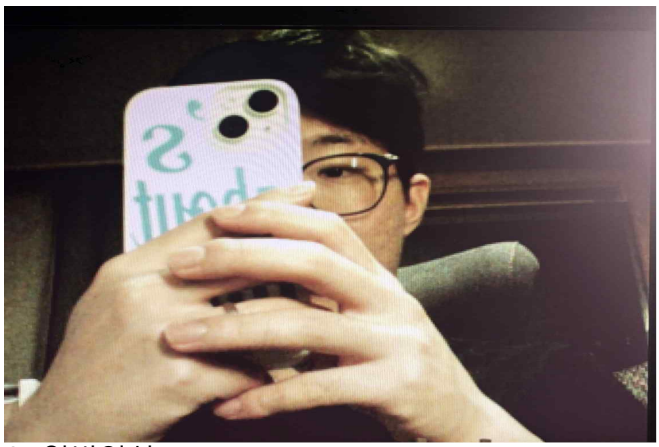
문제 3 VGA 해상도 불일치(320→640)
원인 :도메인 불일치 및 유효(valid) 신호 오정렬
해결 : 모든 처리 해상도를 VGA 기준(640×480)으로 통일, 신호 파이프라인 재정렬

프로젝트 결과

OV7670 영상 입력을 FPGA에서 실시간 처리 및 VGA 640×480 해상도로 안정적 출력

IR 리모컨으로 실시간 필터 전환 가능

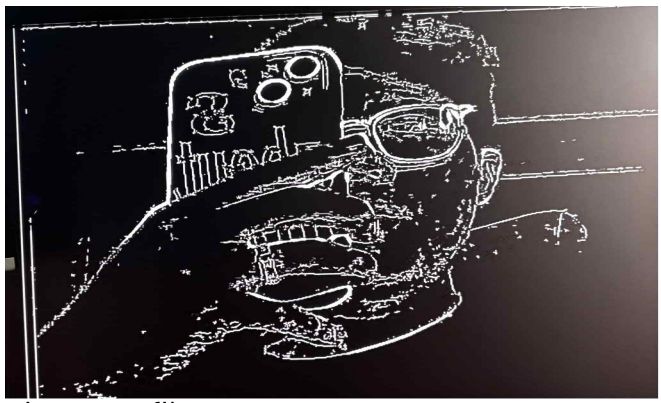
Gray, Sobel, Canny, Color Tracker, Adaptive Background 등 모든 필터 정상 동작



1. 일반영상

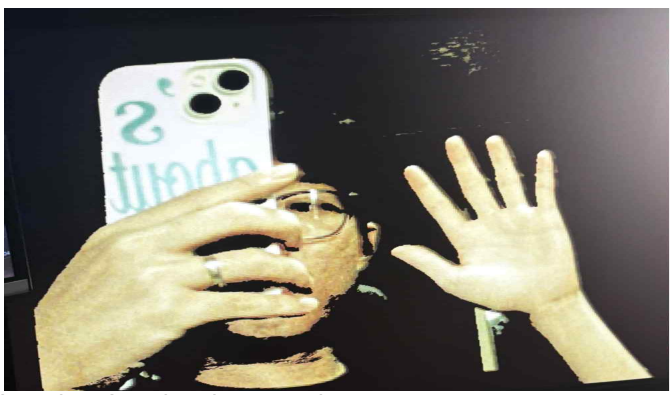
2. gray_scale

3. sobel_filter



4. canny_filter

5. color_tracker



6. adaptive_background

아쉬웠던 점과 수정해야 할 점

초기 설계 목표는 640×480 해상도의 원본 프레임을 전체 저장 후 처리하는 구조였다.
그러나 내부 BRAM 용량 제약으로 전체 프레임 저장이 어려워,
2×2 디시메이션 방식으로 320×240으로 축소하여 처리하였다.
이로 인해 일부 세부 질감 손실이 발생하였다.
이를 개선하기 위해 SDRAM 기반 외부 메모리 버퍼링을 시도했으나,
타이밍·초기화·동기화 문제로 인해 프로젝트 일정 내 완전 통합은 어려웠다.
결과적으로 실시간 안정성을 우선하여 BRAM 기반 구조를 최종 선택하였다.
또한 밝은 배경 환경에서는 카메라 노이즈가 전경으로 인식되는 문제가 발생하였다.
Adaptive Threshold 및 Gaussian 기반 보정을 적용했으나,
완벽한 구분은 이루어지지 않아 향후 조명 보정 및 동적 임계값 조정 알고리즘의
하드웨어 구현이 필요함을 확인했다.

이 프로젝트를 진행하며 배운 점

본 프로젝트를 통해 파이프라인 지연과 신호 정렬의 중요성을 깊이 이해하게 되었다.
모듈 간 데이터·주소·유효 신호의 타이밍 불일치로 인한
프레임 깨짐과 색상 왜곡 문제를 직접 분석·해결하면서,
'한 클럭의 차이'가 전체 시스템 안정성에 결정적 영향을 미친다는 것을 체감했다.
이를 해결하기 위해 각 모듈의 latency를 정량적으로 분석하고,
데이터·좌표·valid 신호를 동일한 파이프라인 깊이로 정렬하였다.
또한 Dual-Port RAM의 read latency와 필터 처리 지연을 세밀하게 보정해
안정적인 프레임 출력을 달성했다.
이 과정을 통해 단순히 동작하는 회로를 만드는 수준을 넘어,
시간 축 기반의 하드웨어 설계 사고를 확립할 수 있었다.