

<Final Project : Multi-function BlackBox>

2024-Fall

디지털시스템설계 [Digital System Design] 01분반

이강 교수님

Team1	
	
22200034 곽도현	22000249 문희재
전산전자공학부 전자공학심화	전산전자공학부 컴퓨터공학과_전자공학과

2024.12.17 (수)

목차

0. Introduction	3
1. Problem Definition	3
a. Objective	
b. Function	
c. I/O Interfaces	
2. Design.....	6
a. Clock	
b. Data	
c. Top-module	
d. Datapath	
e. Controller	
3. Code	13
a. Hierarchy	
b. Modules	
4. Simulation	78
a. Testbench Code & Result	
5. Implementation	85
a. Synthesis Utilization	
b. Implementation Utilization	
c. Area	
d. Timing Analysis	
6. Conclusion	90
a. Conclusion	
b. Future Plan & Possibility	
7. Appendix	92
a. Usage Description & Demo	
b. Individual Role	
c. Schedule	
d. References	
e. Individual Impressions	

0. Introduction

해당 보고서는 상기 보고서이며, 이번 프로젝트 ‘Multi-Function BlackBox(MFBB)’에 관하여 문제진술문, 설계, 코드, 시뮬레이션, 구현, 결론을 차례대로 진술하면서 현재까지의 상황을 설명하고 있습니다. 이 보고서를 통해 ‘Multi-Function BlackBox’ 프로젝트의 의의와 과정을 살펴보시고 저희의 프로젝트에 대해 더욱 깊은 관심을 가져주시면 감사하겠습니다. 본 프로젝트의 주제는 경찰 블랙박스로, 카메라 화면을 저장하고 스피커로 출력하는 것과 자율주행 기술을 위한 전방 카메라 전처리 기능을 모두 포함하는 것이 목표였습니다. 하지만, 실제로 코드로 구현하고 프로젝트를 진행하는 과정에서 카메라 영상을 메모리에 담기에는 메모리 용량 등의 부족으로 인해 sobel 필터 적용하는 것만 구현하게 되었습니다. 프로젝트 진행 중에 주제가 조금씩 바뀌어 혼란을 드릴 수 있다는 점 양해 부탁드립니다.

1. Problem Definition (문제 정의)

문제 기술문: 현대 자동차에는 다양한 안전 시스템이 더해지고 있으며, 그중 대표적인 장치가 블랙박스와 ADAS (Advanced Driver Assistance System)와 같은 차량용 카메라 시스템이다.

- 블랙박스는 사고 발생 시 영상 기록을 통해 증거를 확보하고 사고 원인을 분석하는 데 사용된다.
- 차량 카메라 시스템은 주행 보조 시스템(예: 전방 차량 감지, 차선 이탈 경보)을 지원하기 위해 실시간 영상을 처리하고 데이터를 분석한다.

그러나 현재의 차량 시스템은 블랙박스와 차량 카메라 시스템을 별개의 장치로 운영하는 경우가 많다. 이는 다음과 같은 문제점을 초래한다:

- 중복 설치로 인한 비용 증가
- 두 시스템의 데이터 처리 및 저장을 위한 자원 낭비
- 시스템 설치 및 유지 보수의 복잡성 증가

기존 시스템의 문제점

- 중복된 하드웨어 비용: 블랙박스와 차량 카메라 시스템은 동일한 영상 데이터를 처리하지만, 별개의 카메라 및 저장 장치를 사용하고 있다. 예를 들어, 블랙박스는 사고 기록만을 목적으로 하고, 차량 카메라는 ADAS에 필요한 실시간 영상 데이터를 처리한다. 이로 인해 차량에는 불필요하게 2개의 카메라가 장착되며, 이에 따른 하드웨어 비용이 크게 증가한다.

일반적으로 전방 카메라만 장착된 제품의 설치 비용은 5~10만원 정도이며, 전후방 카메라가 장착된 제품의 설치 비용은 10~15만원 정도이다. (“블랙박스 설치방법 설치비용 영상보는법 총정리 2024”, 하비부자)

아직 블랙박스를 구입하지 않은 운전자들은 비싼 가격에 대한 부담감을 느껴 구입하지 않은 경우가 가장 많았다. (“블랙박스, 84.2%, 필요성 느끼고 있지만 장착은 10명 중 4명”, 테크월드뉴스)

- 데이터 리소스 낭비: 블랙박스와 차량 카메라는 서로 다른 목적으로 데이터를 수집하지만, 기본적으로 동일한 영상을 수집한다.
 - 블랙박스는 영상을 녹화 및 저장
 - 차량 카메라는 영상을 분석 및 실시간 처리

이러한 기능은 하나의 시스템에서도 충분히 구현할 수 있음에도 불구하고 별도의 장치에서 처리된다.

- 공간 및 설치 문제: 두 장치를 동시에 설치할 경우, 차량 내부 공간을 더 많이 차지하게 되어 사용자 불편이 가중된다. 특히, 블랙박스와 차량 카메라의 설치 위치가 중첩될 경우 시야를 가리는 문제가 발생한다.

- 소프트웨어 및 데이터 처리의 이중화: 각 장치가 서로 다른 소프트웨어를 사용하므로 시스템 관리 및 유지 보수가 어려워진다. 두 시스템에서 개별적으로 데이터를 처리하기 때문에 데이터 연동이 불가능하며 통합 분석이 어렵다.

1-a. Objective

본 프로젝트의 목적은 블랙박스와 차량용 카메라 시스템을 단일 시스템으로 통합하는 것이다. 이를 통해 다음과 같은 기대 효과를 도출한다.

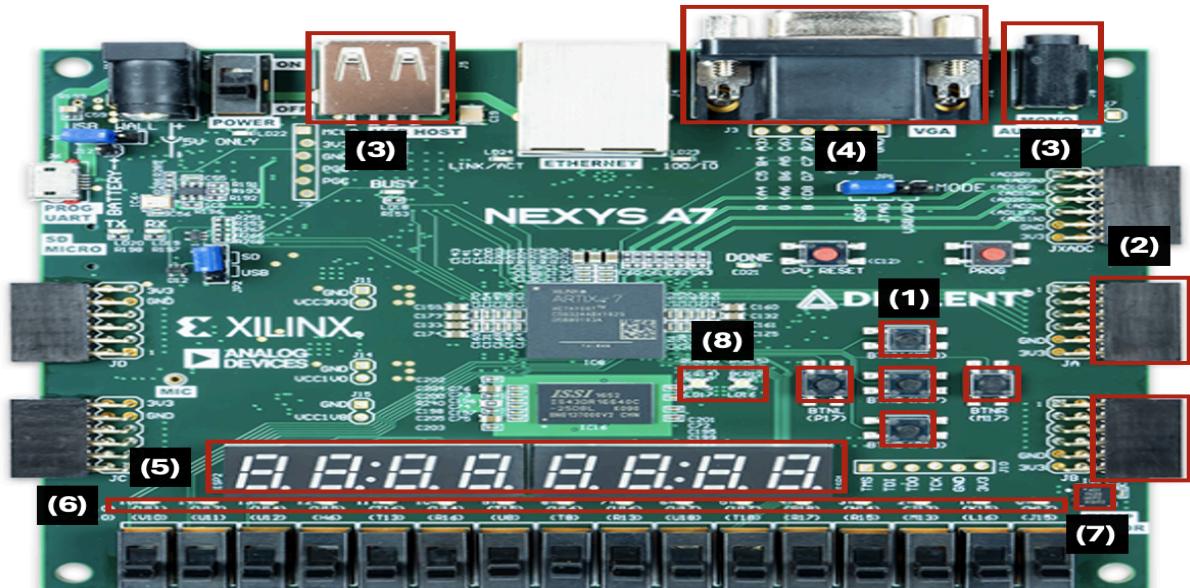
1. 장치 비용 절감
2. 데이터 처리 및 저장 리소스 통합으로 효율적인 운영
3. 설치 및 유지보수 비용 절감
4. 차량 내 시스템 단순화

1-b. Function

단일 시스템은 다음과 같은 기능을 제공해야 한다.

1. 하나의 카메라로 영상 데이터 수집:
 - a. 블랙박스와 차량 카메라의 기능을 통합하여 단일 카메라로 영상을 촬영한다.
2. 이중 기능 구현:
 - a. 블랙박스 기능(영상 저장 및 사고 발생 시 데이터 기록) + 차량용 ADAS 기능(차선 이탈, 장애물 감지 등)
3. 하드웨어 자원 공유: 동일한 하드웨어 플랫폼을 사용하여 데이터 저장, 처리, 송출을 동시에 지원한다.
 - a. CPU, 메모리, 저장 장치를 효율적으로 활용한다.
4. 비용 절감 및 효율성 향상
 - a. 카메라, 저장 장치, 설치 비용을 줄여 전체 비용을 절감한다.
 - b. 시스템 유지 보수의 단순화를 통해 관리 효율성을 향상시킨다.

1-c. I/O Interface



[Figure 1.1] Nexys-A7-100T Board Device_I/O ports

1. Buttons :

- a. **BTNC** (Central Button)
 - i. 역할: 프로그램을 전체적으로 초기화하거나 리셋하고 전원버튼의 역할을 수행한다.
 - ii. 기능: 프로그램 전체의 **reset** 기능을 담당한다.
- b. **BTNL** (Left Button)
 - i. 역할: Threshold 감소 기능을 수행한다. (임계값을 낮추는 역할)
 - ii. 기능: 이미지 처리에서 설정된 임계값을 낮추어 흰색 선의 검출 조건을 높게 조정할 수 있다.
- c. **BTNR** (Right Button)
 - i. 역할: Threshold 증가 기능을 수행한다. (임계값을 높이는 역할)
 - ii. 기능: 이미지 처리에서 설정된 임계값을 높여서 흰색 선의 검출 조건을 덜 민감하게 설정한다.
- d. **BTNU** (Up Button)
 - i. 역할: 카메라 입력으로부터 처리된 **line detecting** 화면을 출력하는 기능이다.
 - ii. 기능: Sobel 필터로 처리된 카메라 데이터를 선택적으로 모니터에 출력한다.

2. PMOD HEADER :

- a. **JA, JB PMOD**
 - i. 역할: OV7670 카메라를 연결하여 데이터를 수신하고 FPGA 시스템에 입력한다.
 - ii. 기능:
 1. PMOD 케이블과 브레드보드를 통해 연결되며, 이 과정에서 노이즈 방지를 위해 저항을 함께 연결하여 노이즈를 해결한다.
 2. 연결된 카메라 데이터를 FPGA가 처리하고 VGA에 결과를 출력한다.

3. PWM Audio Amplifier, USB HOST :

- a. **PWM Audio & USB HOST**
 - i. 역할:
 1. BTNL (임계값 높을 때): 스피커를 통해 휘선 검출 시 효과음을 출력한다.
 2. BTNR (임계값 낮을 때): 스피커를 통해 휘선 검출 시 효과음을 출력한다.
 3. 과열 상황 감지 시 즉각적으로 알림음을 발생시켜 시스템의 안전성을 높인다.
 - ii. 기능:
 1. 임계값을 변화시킬 때 효과음을 출력한다.
 2. 온도 센서에서 40도 이상의 온도를 감지할 경우 경고음을 출력한다.

4. VGA Connector :

- a. **12-bit VGA**
 - i. 역할:
 1. 초기에 설정된 블랙 화면을 출력한 후 카메라 데이터를 통해 처리된 이미지를 실시간으로 모니터에 출력한다.
 2. 초기 설정해둔 블랙박스 시작 화면을 모니터 화면에 출력한다.
 - ii. 기능:
 1. 카메라를 통해 입력받은 영상 데이터를 Sobel 필터링 등의 영상 처리를 거친 후 화면에 출력하고 초기화면을 출력한다.
 2. 실시간으로 라인 검출 결과와 초기화면을 VGA 디스플레이에 표시한다.

5. 7 segment display :

a. Two 4-digit 7-segment display

- i. 역할: 내부 온도 센서(Temperature Sensor)로부터 수집된 데이터를 실시간으로 디스플레이한다.
- ii. 기능: 각 세그먼트를 통해 화씨 및 섭씨 온도를 동시에 보여준다.

6. LEDs :

a. 16 LEDs

- i. 역할: 내부 온도 센서의 데이터를 분석하고 온도를 LED에 실시간으로 표시한다.
- ii. 기능: 내부 온도를 Temperature Sensor로 통해 입력받은 섭씨, 화씨 온도를 2진수화하여 LED에 실시간으로 출력한다.

7. Temperature Sensor :

a. ADT7420 temperature sensor

- i. 역할: TEMP 센서가 40도 이상의 온도를 감지하면 시스템에 경고를 보낸다.
- ii. 기능:
 1. 스피커를 통해 즉시 경고음 출력한다.
 2. 7 Segment Display 및 LED를 통해 온도 상태를 표시한다. FPGA 시스템이 과열되거나 센서로 감지되는 온도가 높을 때, 즉각적으로 경고를 발생시켜 시스템의 안전성을 유지한다.

8. RGB LEDs :

a. one Tri-LED

- i. 역할: BTNC (전원 버튼)와 BTNU (카메라 송출 버튼)의 상태를 나타낸다.
- ii. 기능:
 1. 카메라 데이터 송출 준비 상태로 RED 빛을 낸다.
 2. 카메라 데이터가 정상적으로 송출 상태로 Green 빛을 낸다.
 3. 시스템이 준비 상태에서 송출 중으로 전환되면 LED 색상이 변경되어 상태를 시각적으로 확인할 수 있다.

9. Others

a. Block Memory

- i. 역할:
 1. coe 파일을 통해 프로그램이 초기화된 이미지(초기화면) 데이터를 저장한다. 이 coe 파일은 Verilog 코드 내에서 초기 Block RAM 데이터로 로드되며, VGA 컨트롤러를 통해 출력된다.
 2. Block RAM은 동기식 메모리로, 클럭 신호에 따라 데이터를 읽고 쓴다.
 3. FPGA 보드의 VGA 출력 시스템은 브람 데이터를 읽어와 픽셀 단위로 모니터에 송출한다.
 4. 초기화된 이미지는 고정된 메모리 주소에 저장되며, 화면 출력 시 해당 주소를 참조하여 데이터를 읽는다.
- ii. 기능: 카메라 송출이 없을 때 초기화면 이미지를 출력하여 시스템 상태를 확인하거나 기본 화면을 제공합니다.

b. Clk_gen

- i. 역할: clock divider의 역할을 수행한다. 24MHz, 25MHz, 50MHz, 100Hz를 생성한다.
- ii. 기능: OV7670 카메라와 Block RAM이 필요로 하는 주파수의 클럭을 생성한다.

2. Design (설계)

2-a. Clock

클럭 생성 모듈(**clk_wiz_0**, **clk_wiz_1**, **clkgen_200KHz**)은 시스템의 다양한 부분에서 요구되는 클럭 주파수를 제공한다. 각 모듈이 동작할 수 있도록 필요한 주파수를 생성하였다.

1. 구성 요소:

- a. **clk_wiz_0**: 100MHz 입력 클럭으로부터 25MHz, 50MHz, 100MHz 클럭을 생성. VGA 동작을 위해 사용된다.
- b. **clk_wiz_1**: 카메라 OV7670에서 필요한 XCLK(25MHz)를 생성하며, VGA 및 메모리 클럭에 동기화된 25MHz도 출력한다.
- c. **clkgen_200KHz**: 온도 센서를 위한 200kHz SCL 클럭 생성한다.

2. 특징:

- a. 각 모듈 간 클럭이 정확히 동기화될 수 있도록 설계되었다.
- b. 다양한 주파수 요구 사항을 충족하였다.

2-b. Data

외부 데이터를 처리하기 위해 BRAM을 활용하였다.

1. 구성 요소:

- a. **mem_bram**: BRAM 기반의 듀얼 포트 메모리이며, 비디오 데이터(픽셀 정보)를 저장하고 읽는다.
- b. **sync_fifo**: 동일한 클럭 도메인에서 동작하며, 내부 데이터 처리의 속도 차이를 완화하는 역할을 수행한다.
- c. **async_fifo**: 서로 다른 클럭 도메인 간 데이터 전송을 조율하며, 카메라 데이터와 내부 데이터 처리 간의 상호작용을 관리한다.

2. 특징:

- a. BRAM 기반의 듀얼 포트 메모리(**mem_bram**)를 사용하여 비디오 데이터를 병렬로 저장하고 읽을 수 있어, 고속 데이터 처리가 가능하다.
- b. **sync_fifo**는 동일한 클럭 도메인 내에서 데이터를 순차적으로 처리하여 내부 처리 모듈 간의 속도 차이를 완화한다.
- c. **async_fifo**는 서로 다른 클럭 도메인 간 데이터를 안전하게 전송하고 동기화하여, 카메라 모듈과 내부 처리 모듈 사이의 안정적인 데이터 흐름을 보장한다.
- d. FIFO와 BRAM을 조합하여 다양한 데이터 처리 요구 사항에 맞게 설계 가능하며, 데이터 저장 및 전송을 효율적으로 관리한다.
- e. 메모리와 FIFO를 활용함으로써 데이터 손실을 방지하고 시스템의 안정성을 높이며, 모듈 간 확장성을 제공한다.

2-c. Top-module

top 모듈은 전체 시스템의 최상위 모듈로, BGM 생성, 온도 센서 관리, 카메라 데이터 처리, VGA 디스플레이 제어 등 다양한 서브시스템을 통합한다. 설계 관점에서 이 모듈의 구조와 동작 원리는 다음과 같이 설명할 수 있다.

1. 목표

- a. 다양한 하위 모듈을 하나의 시스템으로 통합하여 효과적으로 데이터 흐름을 관리
- b. 사용자 입력 및 외부 장치(카메라, 온도 센서, 디스플레이 등)와 상호작용
- c. 복잡한 서브 모듈 간 데이터 동기화 및 상태 제어

2. 역할

- a. **BGM** 제어: 게임 상태와 사용자 입력에 따라 적절한 사운드 출력을 생성한다.
- b. 온도 디스플레이: 온도 센서로부터 읽은 데이터를 디스플레이 및 LED로 출력하며, 특정 조건에서 알람을 생성한다.
- c. 카메라 및 영상 처리: 카메라에서 수집한 픽셀 데이터를 처리하여 Sobel 필터를 적용하고 VGA 디스플레이로 출력한다.
- d. 클럭 및 제어 신호 관리: 각 서브 모듈이 필요로 하는 클럭과 제어 신호를 생성 및 분배한다.

3. 모듈 간 데이터 흐름 설계

a. 주요 모듈 구성

- i. **top_bgm**: 배경 음악 및 효과음을 생성하는 모듈이다.
 - ii. **top_temp**: 온도 센서 데이터를 읽어 디스플레이와 LED로 출력하며, 특정 온도 이상에서는 알람을 발생한다.
 - iii. **top_cam**: 카메라 데이터를 처리하고, Sobel 필터링을 통해 가장자리 검출을 수행한 후 VGA로 출력한다.
 - iv. **VGA_mem_top**: 메모리에서 픽셀 데이터를 읽어 VGA 신호를 생성한다.
- b. 데이터 흐름의 통합
 - i. 입력 신호:
 1. 사용자 입력 (예: Sobel 필터의 임계값 조정)
 2. 카메라 픽셀 데이터 및 온도 센서 데이터
 - ii. 처리 흐름:
 1. 각 서브 모듈은 독립적으로 데이터를 처리하지만, 상위 모듈에서 전체 데이터 흐름을 조율한다.
 2. 카메라 데이터는 Sobel 필터를 거친 후 VGA 메모리에 저장되며, 이후 디스플레이로 출력한다.
 3. 온도 데이터는 I2C 프로토콜로 읽어들인 후 7-Segment 디스플레이와 LED로 출력한다.
 - iii. 출력 신호:
 1. VGA 출력 (RGB 데이터 및 싱크 신호)
 2. 사운드 출력
 3. 온도 데이터 디스플레이

4. 클럭 및 동기화 설계

a. 클럭 생성 및 분배

- i. **i_top_clk** (시스템 메인 클럭, 100MHz)를 기반으로 여러 주파수의 클럭을 생성:
 1. clk_wiz_0 및 clk_wiz_1 모듈을 통해 25MHz, 50MHz, 100MHz 클럭을 생성한다.
 2. 카메라 모듈은 25MHz XCLK 신호를 사용한다.
 3. VGA 디스플레이는 25MHz 클럭으로 동작한다.

b. 동기화

- i. 카메라와 내부 처리 간 동기화:
 1. **async_fifo**를 사용하여 서로 다른 클럭 도메인 간 데이터를 안정적으로 전송한다.
 2. 카메라 데이터는 Sobel 필터링을 거친 후 동기 FIFO(sync_fifo)에 저장한다.
- ii. VGA 디스플레이와 메모리 간 동기화: VGA 메모리는 Sobel 필터 결과 데이터를 읽어 25MHz 클럭으로 송출한다.

5. 제어 신호 및 FSM 설계

a. 제어 신호

- i. **BGM** 제어: i_top_sobel 신호를 통해 Sobel 필터 조정 상태에 따라 BGM 모드를 변경한다.
- ii. 카메라 제어: cam_start_cond 신호를 통해 카메라 모듈 시작을 VGA 메모리

- iii. 준비 상태(vga_ready)와 연동한다.
- iii. **VGA** 제어: VGA 출력은 처리된 픽셀 데이터와 상태 신호(vga_ready)에 의해 제어한다.
- b. 상태 머신(**FSM**)
 - i. 카메라 데이터 처리: top_cam은 카메라 시작, 데이터 수신, 처리 완료의 세 단계로 구성된 FSM으로 동작한다.
 - ii. **Sobel** 필터 적용: 임계값 조정 신호(i_top_inc_sobel_thresh, i_top_dec_sobel_thresh)를 기반으로 Sobel 필터링의 민감도를 변경한다.

6. 출력 관리

- a. **VGA** 출력
 - i. top_cam과 VGA_mem_top에서 처리된 데이터를 VGA 디스플레이로 출력한다.
 - ii. VGA 출력 신호는 RGB 데이터(o_top_vga_red, o_top_vga_green, o_top_vga_blue)와 싱크 신호(o_top_vga_vsync, o_top_vga_hsync)로 구성한다.
- b. 오디오 출력
 - i. BGM 모듈에서 생성된 사운드 신호(voice_fre)와 알람 신호(speaker)를 audio_output으로 출력한다.
- c. 온도 데이터 출력
 - i. I2C로 읽어들인 온도 데이터를 7-Segment 디스플레이(SEG, AN)와 LED(LED)로 표시한다.
 - ii. 온도가 특정 임계값을 초과하면 알람 발생한다.

7. 설계의 확장성

- a. 모듈화: 각각의 기능이 독립적인 모듈로 구현되어 유지보수 및 확장이 용이하다. 예를 들어, 다른 센서를 추가하거나 새로운 필터를 적용할 경우, 관련 모듈만 수정하면 된다.
- b. 동기화 설계: FIFO와 클럭 동기화를 통해 다양한 클럭 도메인에서도 안정적인 동작 보장한다.
- c. 유연성: 사용자 입력을 기반으로 Sobel 필터 민감도 조정, 다양한 상태의 BGM 출력 등 동적 제어 가능하다.

topmodule은 다양한 외부 입력과 동작 조건을 효율적으로 통합하며, 모듈 간 데이터를 안정적으로 처리하여 VGA 디스플레이와 같은 실시간 출력에 적합한 구조를 갖추도록 한다. 각 기능이 모듈화되어 있어 유지보수와 확장성을 높이도록 한다.

2-d. Datapath

프로젝트의 데이터 경로(Datapath)는 카메라에서 수집한 데이터를 처리하고 디스플레이로 출력하는 주요 작업을 수행하도록 한다. 이 과정에서 데이터는 여러 모듈을 거치며, 각각의 모듈은 특정 연산을 담당하게 된다.

1. 데이터 경로의 목표와 역할

- a. 목표
 - i. 카메라로부터 수집한 원시 데이터를 처리하여 사용자에게 유의미한 정보 제공
 - ii. 실시간으로 데이터를 처리하여 VGA 디스플레이에 출력
 - iii. 데이터의 정확성을 유지하면서 처리 속도 최적화
- b. 역할
 - i. 데이터 수집: 카메라 모듈에서 수집한 픽셀 데이터를 캡처한다.
 - ii. 데이터 변환: 픽셀 데이터를 그레이스케일로 변환하여 연산을

- iii. 단순화한다.
- iv. 데이터 필터링: **Sobel** 필터를 통해 경계선을 감지하고 이미지 특징을 추출한다.
- iv. 데이터 저장 및 전송: 처리된 데이터를 메모리에 저장하고 VGA 출력 모듈로 전송한다.

2. 데이터 경로 구성 요소

a. 주요 모듈

- i. **cam_top:**
 1. 카메라 데이터를 수집하여 픽셀 단위로 전송한다.
 2. 데이터의 동기화를 위해 FIFO를 사용한다.
- ii. **vp_top:** 데이터 변환과 필터링을 담당하는 핵심 모듈
 1. grayscale 모듈을 통해 grayscale 변환한다.
 2. conv 모듈을 통해 3x3 Sobel 필터 적용한다.
 3. FIFO 방식으로 데이터 저장 및 동기화한다.
- iii. **mem_top:**
 1. 처리된 데이터를 BRAM에 저장한다.
 2. VGA 출력 모듈로 데이터를 제공한다.
- iv. **vga_top:**
 1. 메모리에서 데이터를 읽어와 VGA 디스플레이로 출력한다.
 2. RGB 데이터와 싱크 신호를 생성한다.

b. 데이터 처리 흐름

- i. 카메라 데이터 캡처:
 1. **cam_top**에서 카메라로부터 RGB444 데이터를 수집한다.
 2. 데이터는 픽셀 단위로 FIFO에 저장되어 처리 모듈로 전달한다.
- ii. 데이터 변환:
 1. **grayscale** 모듈에서 RGB 데이터를 8비트 grayscale로 변환한다.
 2. 계산 효율성을 높이기 위해 색 정보를 단순화한다.
- iii. 데이터 필터링:
 1. **conv** 모듈에서 Sobel 필터를 적용하여 경계선을 감지한다.
 2. 임계값(threshold)을 기반으로 유의미한 데이터만 추출한다.
- iv. 데이터 저장:
 1. 처리된 데이터는 **sync_fifo**를 통해 **mem_top**의 BRAM에 저장한다.
- v. VGA 출력:
 1. **vga_top**에서 메모리 데이터를 읽어 RGB 신호로 변환한다.
 2. VGA 디스플레이를 위한 싱크 신호와 함께 출력한다.

3. 데이터 경로의 단계별 설명

a. 데이터 입력 단계

- i. 카메라 데이터 수집:
 1. **cam_top**은 카메라의 픽셀 데이터를 수집하고, **async_fifo**를 사용하여 내부 처리 모듈로 전송한다.
 2. 픽셀 데이터는 **RGB444** 형식이며, 동기화를 위해 비동기 FIFO를 활용한다.

b. 데이터 변환 및 필터링 단계

- i. **grayscale** 변환:
 1. **vp_top** 모듈의 **grayscale** 서브 모듈에서 RGB 데이터를 8비트 그레이스케일로 변환한다.
 2. 변환 공식: $Gray = 0.3*R + 0.59*G + 0.11*B$
- ii. **Sobel** 필터 적용:
 1. **conv** 모듈에서 3x3 커널을 사용하여 경계선을 감지한다.
 2. 임계값(threshold)을 설정하여 불필요한 데이터 제거한다.
 3. 결과 데이터는 픽셀 단위로 FIFO에 저장한다.

c. 데이터 저장 단계

- i. FIFO와 메모리 사용:
 - 1. sync_fifo를 통해 처리된 데이터를 mem_top의 BRAM에 저장한다.
 - 2. BRAM은 VGA 출력 모듈이 읽을 수 있도록 데이터를 저장한다.
- d. 데이터 출력 단계
 - i. VGA 신호 생성:
 - 1. vga_top 모듈은 BRAM 데이터를 읽어와 RGB 데이터로 변환한다.
 - 2. VGA 디스플레이를 위한 싱크 신호와 함께 출력한다.

4. 데이터 경로 설계의 특징

- a. 효율적인 데이터 처리
 - i. 데이터 경로는 병렬 처리를 최대한 활용하여 처리 속도를 높인다.
 - ii. FIFO와 BRAM을 활용해 데이터 병목 현상을 최소화한다.
- b. 실시간 처리
 - i. 데이터 수집에서 출력까지의 경로가 최적화되어 실시간 처리가 가능하다.
 - ii. Sobel 필터와 같은 연산도 클럭 주파수에 맞춰 빠르게 수행한다.
- c. 모듈화된 구조
 - i. 각 기능이 독립적인 모듈로 설계되어 유지보수와 확장이 용이하다.
 - ii. 새로운 필터나 처리 단계를 추가할 경우 기존 모듈에 최소한의 영향을 미친다.
- d. 안정적인 데이터 흐름
 - i. 비동기 FIFO를 통해 서로 다른 클럭 도메인 간 데이터 전송이 안정적으로 이루어진다.
 - ii. FIFO와 BRAM을 조합하여 데이터 손실 없이 효율적으로 관리한다.

5. 데이터 경로 설계의 확장 가능성

- a. 효율적인 데이터 처리
 - i. 데이터 경로는 병렬 처리를 최대한 활용하여 처리 속도를 높인다.
 - ii. FIFO와 BRAM을 활용해 데이터 병목 현상을 최소화한다.
- b. 고해상도 처리
 - i. Sobel 필터 외에 다양한 필터(예: Gaussian, Laplacian)를 추가 가능하다.
 - ii. 멀티채널 데이터 처리로 확장하여 더 복잡한 연산 수행 가능하다.
 - iii. BRAM 용량과 FIFO 크기를 확장하여 고해상도 이미지 처리 가능하다.
 - iv. 카메라에서 수집하는 데이터의 해상도를 조정하여 다양한 응용 가능하다.
- c. 데이터 분석
 - i. 처리된 데이터를 외부로 출력하여 추가적인 데이터 분석 수행 가능하다.
 - ii. 머신러닝 기반의 패턴 인식 시스템과 통합 가능하다.

데이터 경로는 시스템의 성능과 안정성을 결정짓는 중요한 요소로, 본 설계에서는 이를 최적화하기 위한 다양한 기법을 적용할 것이다. 이와 같은 구조는 실시간 데이터 처리와 안정성을 모두 만족시키며, 다양한 응용 분야로 확장이 가능할 것이다.

2-e. Controller

이 프로젝트의 컨트롤러 설계는 각 모듈이 데이터를 효율적으로 처리하고 시스템 전체의 동작이 조화를 이루도록 하는 핵심 역할을 담당한다. 상태 머신(FSM) 기반의 설계가 주요 특징이며, 사용자 입력, 데이터 처리 흐름 제어, 그리고 모듈 간 동기화를 담당한다.

1. 목표

- a. 각 모듈의 상태를 제어하고 동작 순서를 조율
- b. 외부 입력 신호에 따라 시스템의 동작을 동적으로 변경
- c. 데이터 처리 과정에서 안정적인 데이터 흐름을 보장

2. 역할

- a. 상태 관리: 각 모듈의 동작 상태를 관리하고 상태 전환을 수행한다.
- b. 데이터 흐름 제어: 데이터가 각 모듈을 통과하는 순서와 속도를 조절한다.
- c. 사용자 입력 처리: 사용자 입력 신호를 디바운싱하고, 이를 기반으로 Sobel 필터 민감도 및 BGM 상태를 변경한다.
- d. 에러 방지: FIFO와 메모리 사용 시 과/부하를 방지하고 시스템 안정성을 유지한다.

3. 주요 컨트롤러 모듈

- a. **top_user_control:** 사용자 입력을 처리하는 모듈로, Sobel 필터의 임계값을 조정하는 역할을 한다.
 - i. 입력 신호:
 - 1. i_inc_sobel_thresh: Sobel 필터의 임계값 증가 신호
 - 2. i_dec_sobel_thresh: Sobel 필터의 임계값 감소 신호
 - ii. 주요 동작:
 - 1. 디바운싱 처리: 입력 신호의 노이즈 제거
 - 2. FSM 기반 동작: 상태 전환에 따라 Sobel 필터 임계값 증가 또는 감소
 - 3. 임계값의 최소/최대 제한 설정
 - iii. 출력 신호: o_sobel_thresh: 조정된 Sobel 필터 임계값
- b. **vp_control:** 비디오 처리 모듈의 컨트롤러로, 3x3 커널 데이터 준비와 데이터 유효성 검사를 담당한다.
 - i. 입력 신호:
 - 1. i_pixel_data_valid: 입력 픽셀 데이터의 유효성 신호
 - 2. i_pixel_data: 입력 픽셀 데이터
 - ii. 주요 동작:
 - 1. 라인 버퍼 관리:
 - a. 각 라인 버퍼에 데이터를 순차적으로 저장
 - b. 3개의 라인 버퍼 데이터를 조합하여 3x3 커널 생성
 - 2. FSM 기반 읽기/쓰기 제어:
 - a. 충분한 데이터가 라인 버퍼에 채워질 때까지 대기(IDLE 상태)
 - b. 커널 생성이 가능해지면 데이터를 읽어 conv 모듈로 전달
 - iii. 출력 신호:
 - 1. o_pixel_valid: 3x3 커널 데이터의 유효성 신호
 - 2. o_pixel_data: 3x3 커널 데이터
- c. **top_cam:** 카메라 데이터 처리 흐름을 제어하는 컨트롤러로, 데이터 수집, 처리, 저장을 담당한다.
 - i. 입력 신호:
 - 1. i_top_cam_start: 카메라 데이터 수집 시작 신호
 - 2. i_top_rst: 전체 시스템 초기화 신호
 - ii. 주요 동작:
 - 1. 카메라 초기화: cam_init을 통해 카메라 레지스터 설정
 - 2. 픽셀 데이터 캡처: 카메라로부터 픽셀 데이터를 수집하고, FIFO를 통해 내부 처리 모듈로 전달
 - 3. 처리 완료 신호: 데이터 처리가 완료되면 o_top_cam_done 신호를 출력
- d. **vga_top:** VGA 디스플레이의 컨트롤러로, 메모리에서 데이터를 읽어와 디스플레이로 출력하는 과정을 제어한다.
 - i. 입력 신호:
 - 1. i_pix_data: 메모리에서 읽어온 픽셀 데이터
 - 2. i_clk25m: 25MHz VGA 클럭
 - ii. 주요 동작:
 - 1. FSM 기반 동작:
 - a. 두 프레임을 건너뛰는 초기화 단계(WAIT 상태)

- b. 데이터가 준비되면 메모리 주소를 증가시키며 픽셀 데이터를 읽어 출력
- 2. 활성 비디오 영역 확인: 현재 픽셀이 활성 비디오 영역에 해당하는지 확인
- iii. 출력 신호:
 - 1. VGA RGB 데이터 (`o_VGA_r`, `o_VGA_g`, `o_VGA_b`)
 - 2. 싱크 신호 (`o_VGA_vsync`, `o_VGA_hsync`)

4. FSM 기반 동작

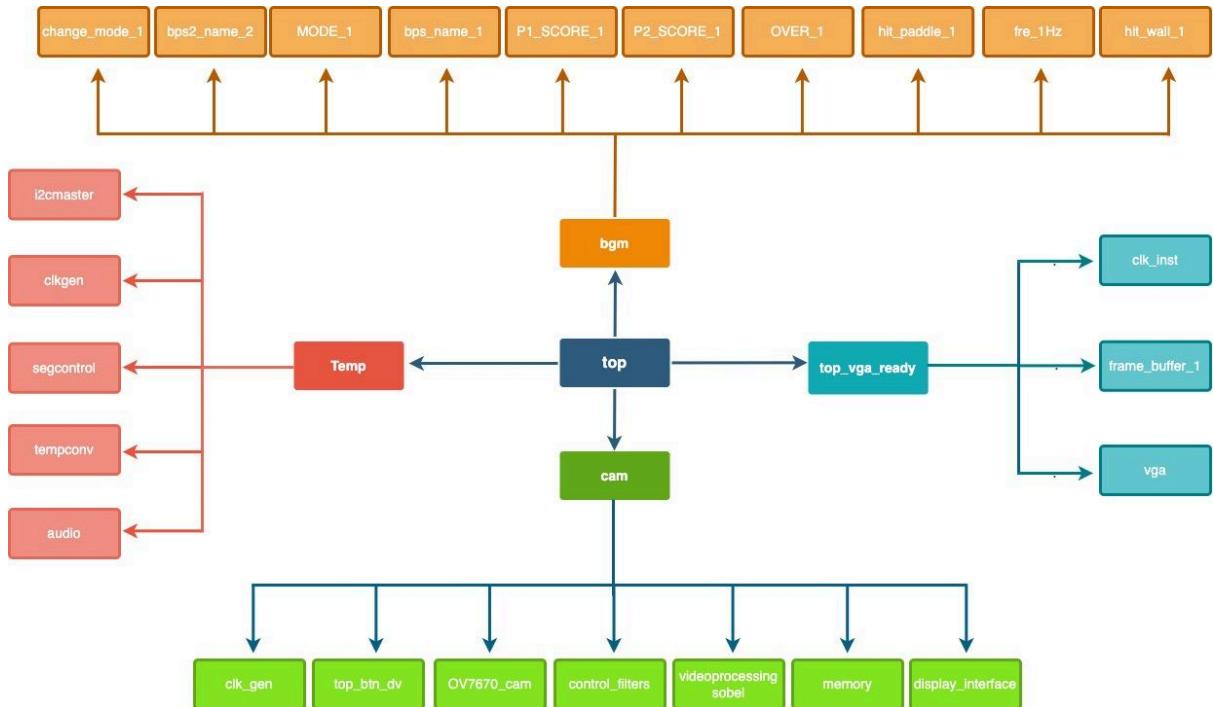
- a. **FSM** 설계 철학
 - i. 각 컨트롤러는 **FSM** 기반으로 설계되어 상태 전환이 명확하며, 각 상태에서 수행할 작업이 명시적이다.
 - ii. **FSM**은 주로 IDLE, READ, WRITE, WAIT 등의 상태로 구성된다.
 - iii. 외부 신호(예: 데이터 유효성 신호, 사용자 입력)에 따라 상태가 전환된다.
- b. **FSM** 예시
 - i. `vp_control`:
 - 1. IDLE
 - a. 라인 버퍼에 충분한 데이터가 채워지기를 대기
 - b. 데이터가 충분하면 READ 상태로 전환
 - 2. READ
 - a. 라인 버퍼에서 데이터를 읽어 3x3 커널을 생성
 - b. 읽기가 완료되면 다시 IDLE 상태로 전환
 - ii. `vga_top`
 - 1. WAIT:
 - a. 초기 두 프레임을 건너뛰며 메모리 초기화.
 - 2. READ:
 - a. 메모리 주소를 증가시키며 데이터를 읽어 VGA로 출력
 - b. 비활성 비디오 영역에 도달하면 주소를 초기화

5. Controller 설계 특징

- a. 모듈 간 동기화
 - i. 데이터 유효성 신호(valid)와 준비 신호(ready)를 활용하여 모듈 간 데이터를 안전하게 전송
 - ii. FIFO를 통해 클럭 도메인 간 데이터 흐름을 안정화
- b. 사용자 입력 반영
 - i. Sobel 필터 민감도 조정과 같은 동작은 사용자 입력에 따라 동적으로 변경
 - ii. 디바운싱 처리로 입력 신호의 노이즈를 제거
- c. 데이터 손실 방지
 - i. FIFO의 가득 참(full) 및 거의 가득 참(almost full) 상태를 지속적으로 확인하여 데이터 손실 방지
 - ii. 데이터 처리 속도를 조정하여 안정적인 동작 유지
- d. 설계의 장점과 확장성
 - i. 장점
 - 1. 모듈화: 각 기능이 독립적인 컨트롤러로 분리되어 유지보수가 용이
 - 2. 안정성: 데이터 흐름을 제어하는 **FSM** 기반 설계로 안정적인 동작 보장
 - 3. 효율성: 불필요한 상태 전환을 최소화하여 처리 속도를 최적화
 - ii. 확장성
 - 1. 사용자 입력, 새로운 데이터 처리 기법, 추가 센서 등을 쉽게 통합 가능
 - 2. **FSM** 구조는 상태 추가 및 변경이 용이하여 새로운 동작 요구사항에 유연하게 대응 가능

3. Code (코드)

3-a. Hierarchy



[Figure 5.1] topmodule의 Hierarchy

1. 최상위 모듈 (Top)

- a. top 모듈은 시스템의 중심 역할을 하며, 주요 하위 모듈인 BGM, 온도(Temp), 카메라(Cam), VGA 출력(top_vga_ready)와 데이터를 주고받는다.
- b. 시스템의 데이터 흐름을 관리하며, 입력 데이터를 처리하고 최종 출력을 담당한다.

2. BGM 모듈: BGM은 배경 음악과 사운드 효과를 생성하는 역할을 한다.

- a. 상위 입력 신호로는 change_mode_1, bps_name_1, MODE_1, P1_SCORE_1, P2_SCORE_1, OVER_1, hit_paddle_1, fre_1Hz, hit_wall_1 등이 포함된다. 이러한 신호는 점수 변화, 게임 상태 변화, 충돌 이벤트 등을 BGM 모듈에 전달하여 적절한 사운드 출력을 제어한다.
- b. 이 모듈은 점수나 상태에 따라 BGM 및 효과음을 동적으로 변경한다.

3. 온도 제어 모듈 (Temp): 온도 데이터를 수집 및 변환하여 다른 모듈에 전달하는 역할을 한다.

- a. 입력 모듈:
 - i. i2cmaster: 온도 데이터 통신
 - ii. clkgen: 클럭 신호 생성
 - iii. segcontrol: 세그먼트 디스플레이 제어
 - iv. tempconv: 온도 변환
 - v. audio: 온도 기반 오디오 출력
- b. 온도 데이터는 segcontrol로 출력되어 디스플레이 되며, BGM이나 다른 모듈에도 영향을 줄 수 있다.

4. 카메라 모듈 (**Cam**): Cam은 OV7670 카메라 모듈을 제어하여 영상을 처리한다.
- 하위 모듈:
 - `clk_gen`: 카메라 동작을 위한 클럭 생성
 - `top_btn_dv`: 버튼 입력 처리
 - `OV7670_cam`: 카메라 제어
 - `control_filters`: 영상 필터 제어
 - `videoprocessing_sobel`: 엣지 검출 등 영상 처리
 - `memory`: 프레임 버퍼 저장
 - `display_interface`: 디스플레이 인터페이스
 - 카메라로부터 입력된 영상은 Sobel 필터 등으로 처리된 후 메모리에 저장되며, VGA 디스플레이에 출력된다.
5. **VGA 출력 모듈 (Top VGA Ready)**: `top_vga_ready` 모듈은 카메라 및 처리된 데이터를 VGA 디스플레이로 출력하는 데 사용된다.
- 연결된 모듈:
 - `clk_inst`: 클럭 설정
 - `frame_buffer_1`: 프레임 데이터를 저장
 - `vga`: VGA 디스플레이 제어
 - VGA 출력은 실시간으로 처리된 영상 데이터를 화면에 표시하며, 안정적이고 정확한 신호를 보장하기 위해 클럭 및 프레임 버퍼와 연동된다.
6. 각 모듈 간 데이터 흐름: `top` 모듈은 모든 하위 모듈과 직접적으로 연결되어 데이터 전달과 제어를 수행한다.
- BGM 모듈은 시스템의 다양한 이벤트(점수, 상태 변화 등)를 기반으로 동작하며, 이를 온도 모듈 및 카메라 모듈과 연동한다.
 - Temp 모듈은 온도 데이터를 기준으로 시스템의 동작이나 디스플레이를 조정한다.
 - Cam 모듈은 영상 데이터를 처리하고 이를 VGA 디스플레이 모듈로 전달한다.
 - 모든 데이터는 `top` 모듈에서 통합되어 출력된다.
7. 전체 시스템의 주요 기능
- 이 시스템은 온도 데이터 수집 및 변환, 카메라 영상 처리 및 출력, 게임 상태에 따른 배경음악 및 효과음 출력을 포함한 다기능을 제공한다.
 - `top` 모듈을 중심으로 모든 데이터가 통합되어 정확하고 효율적으로 동작하도록 설계되었다.

이 계층도를 통해 시스템 설계가 모듈화되어 있음을 확인할 수 있으며, 각 기능이 독립적으로 동작하면서도 서로 상호작용하여 하나의 통합 시스템을 구성한다.

3-b. Code

```
module top(
  // BGM I/O
  output audio_output,
  // Temperature I/O
  inout TMP_SDA,      // i2c sda on temp sensor
  output TMP_SCL,     // i2c scl on temp sensor
  output [6:0] SEG,    // 7 segments of each display
  output [7:0] AN,     // 8 anodes of 8 displays
  // Other I/Os
);
```

```

output [15:0] LED,           // binary temp in deg C or deg F

// Camera I/O
input wire i_top_clk,
input wire i_top_RST,
input wire i_top_cam_start,
output wire o_top_cam_done,
output wire o_top_cam_ready,
input wire i_top_inc_sobel_thresh,
input wire i_top_dec_sobel_thresh,
// I/O to camera
input wire i_top_pclk,
input wire [7:0] i_top_pix_byte,
input wire i_top_pix_vsync,
input wire i_top_pix_href,
output wire o_top_reset,
output wire o_top_pwdn,
output wire o_top_xclk,
output wire o_top_siod,
output wire o_top_sioc,
// I/O to VGA
output wire [3:0] o_top_vga_red,
output wire [3:0] o_top_vga_green,
output wire [3:0] o_top_vga_blue,
output wire o_top_vga_vsync,
output wire o_top_vga_hsync );
// BGM CODE
wire i_top_sobel = i_top_inc_sobel_thresh | i_top_dec_sobel_thresh;
assign audio_output = en_audio ? speaker : voice_fre;
wire voice_fre;
top_bgm bgm(
.clk(i_top_clk),
.hit_wall(),
.ball_x(),
.change_mode(i_top_sobel),
.voice_fre(voice_fre),
.state() );
wire speaker;
top_temp temp(
.CLK100MHZ(i_top_clk),
.TMP_SDA(TMP_SDA),
.TMP_SCL(TMP_SCL),
.SEG(SEG),
.AN(AN),
.LED(LED),
.speaker(speaker),
.en(en_audio ) );
// top_cam 시작 신호를 VGA_ready 신호에 의해 제어
wire cam_start_cond = i_top_cam_start & vga_ready;

```

```

wire [3:0] o_top_vga_red2;
wire [3:0] o_top_vga_green2;
wire [3:0] o_top_vga_blue2;
wire      o_top_vga_vsync2;
wire      o_top_vga_hsync2;

top_cam cam(
    .i_top_clk(i_top_clk),
    .i_top_rst(i_top_rst),
    .i_top_cam_start(cam_start_cond),
    .o_top_cam_done(o_top_cam_done),
    .o_top_cam_ready(o_top_cam_ready),
    .i_top_inc_sobel_thresh(i_top_inc_sobel_thresh),
    .i_top_dec_sobel_thresh(i_top_dec_sobel_thresh),
    .i_top_pclk(i_top_pclk),
    .i_top_pix_byte(i_top_pix_byte),
    .i_top_pix_vsync(i_top_pix_vsync),
    .i_top_pix_href(i_top_pix_href),
    .o_top_reset(o_top_reset),
    .o_top_pwdn(o_top_pwdn),
    .o_top_xclk(o_top_xclk),
    .o_top_siod(o_top_siod),
    .o_top_sioc(o_top_sioc),
    .o_top_vga_red(o_top_vga_red2),
    .o_top_vga_green(o_top_vga_green2),
    .o_top_vga_blue(o_top_vga_blue2),
    .o_top_vga_vsync(o_top_vga_vsync2),
    .o_top_vga_hsync(o_top_vga_hsync2) );

assign o_top_vga_red = (vga_ready) ? o_top_vga_red2 : o_top_vga_red1;
assign o_top_vga_green = (vga_ready) ? o_top_vga_green2 : o_top_vga_green1;
assign o_top_vga_blue = (vga_ready) ? o_top_vga_blue2 : o_top_vga_blue1;
assign o_top_vga_vsync = (vga_ready) ? o_top_vga_vsync2 : o_top_vga_vsync1;
assign o_top_vga_hsync = (vga_ready) ? o_top_vga_hsync2 : o_top_vga_hsync1;

wire [3:0] o_top_vga_red1;
wire [3:0] o_top_vga_green1;
wire [3:0] o_top_vga_blue1;
wire      o_top_vga_vsync1;
wire      o_top_vga_hsync1;

VGA_mem_top top_vga_ready(
    .clk(i_top_clk),
    .rst(i_top_rst),
    .VGA_R(o_top_vga_red1),
    .VGA_G(o_top_vga_green1),
    .VGA_B(o_top_vga_blue1),
    .hsyncb(o_top_vga_hsync1),
    .vsyncb(o_top_vga_vsync1),
    .vga_ready(vga_ready) );

endmodule

```

[Code 3.1] top.v

```

module top_bgm (
    input clk, hit_wall, change_mode,

```

```

input [2:0] state,
input [5:0] ball_x,
output reg voice_fre );

reg [7:0] bps = 8'd2;
reg [7:0] bps_2 = 8'd1;

wire bump_2;
wire MODE_music;
wire P1_SCORE_music;
wire P2_SCORE_music;
wire OVER_music;
wire hit_paddle_music;
wire hit_wall_music;
wire change_mode_music;

reg start_MODE;
reg start_P1_SCORE;
reg start_P2_SCORE;
reg start_OVER;
reg hit_paddle;

wire [3:0] MODE_counter;
wire [3:0] P1_SCORE_counter;
wire [3:0] P2_SCORE_counter;
wire [3:0] OVER_counter;
wire [3:0] hit_paddle_counter;
wire [3:0] hit_wall_counter;
wire change_mode_counter;

parameter INIT = 3'd0, MODE = 3'd1, RUNNING = 3'd2,
P1_SCORE = 3'd3, P2_SCORE = 3'd4, OVER = 3'd5;

fre_divider fre_1Hz(
    .clk(clk),
    .clk_10Hz(clk_10Hz) );

bps bps_name_1(
    .clk(clk_10Hz),
    .bps(bps),
    .bump(bump) );

bps bps2_name_2(//bump per secone
    .clk(clk_10Hz),
    .bps(bps_2),
    .bump(bump_2) );
music_7 change_mode_1(.bump(bump_2),
.voice_fre(change_mode_music),.counter(change_mode_counter), .start(change_mode), .clk(clk));

always@(posedge clk)begin
    if(ball_x >= 6'd38 || ball_x <= 6'd1 && (state == RUNNING || state == P1_SCORE || state == P2_SCORE))
        hit_paddle = 1'd1;
    else
        hit_paddle = 1'd0;
end

always@(*)begin // get output signal from other module

```

```

if(MODE_counter > 4'd0)
    voice_fre = MODE_music;
else if(P1_SCORE_counter > 4'd0)
    voice_fre = P1_SCORE_music;
else if(P2_SCORE_counter > 4'd0)
    voice_fre = P2_SCORE_music;
else if(OVER_counter > 4'd0)
    voice_fre = OVER_music;
else if(hit_paddle_counter > 4'd0 && hit_wall == 1'd0)
    voice_fre = hit_paddle_music;
else if(hit_wall_counter > 4'd0)
    voice_fre = hit_wall_music;
else if(change_mode_counter > 4'd0)
    voice_fre = change_mode_music;
else
    voice_fre = 1'd0;

end

reg start_MODE_store = 1'd0;
reg start_P1_SCORE_store = 1'd0;
reg start_P2_SCORE_store = 1'd0;
reg start_OVER_store = 1'd0;

always @(posedge clk) begin//output start signal to music modules
case (state)
INIT:begin
    start_MODE_store = 1'd0;
    start_P1_SCORE_store = 1'd0;
    start_P2_SCORE_store = 1'd0;
    start_OVER_store = 1'd0; end
MODE: begin
    if(start_MODE_store == 1'd0)begin
        start_MODE = 1'b1;
        start_P1_SCORE = 1'b0;
        start_P2_SCORE = 1'b0;
        start_OVER = 1'b0;
        start_MODE_store = 1'd1;end
    else begin
        start_MODE = 1'b0;
        start_P1_SCORE = 1'b0;
        start_P2_SCORE = 1'b0;
        start_OVER = 1'b0; end
    end
RUNNING: begin
    start_MODE_store = 1'd0;
    start_P1_SCORE_store = 1'd0;
    start_P2_SCORE_store = 1'd0;
    start_OVER_store = 1'd0; end
P1_SCORE: begin
    if(start_P1_SCORE_store == 1'd0)begin
        start_MODE = 1'b0;
        start_P1_SCORE = 1'b1;
        start_P2_SCORE = 1'b0;
        start_OVER = 1'b0;
        start_P1_SCORE_store = 1'd1;
    end else begin
        start_MODE = 1'b0;
    end
end

```

```

    start_P1_SCORE = 1'b0;
    start_P2_SCORE = 1'b0;
    start_OVER = 1'b0; end
end
P2_SCORE: begin
if(start_P2_SCORE_store == 1'd0)begin
    start_MODE = 1'b0;
    start_P1_SCORE = 1'b0;
    start_P2_SCORE = 1'b1;
    start_OVER = 1'b0;
    start_P2_SCORE_store = 1'd1;
end else begin
    start_MODE = 1'b0;
    start_P1_SCORE = 1'b0;
    start_P2_SCORE = 1'b0;
    start_OVER = 1'b0; end
end
OVER: begin
if(start_OVER_store == 1'd0)begin
    start_MODE = 1'b0;
    start_P1_SCORE = 1'b0;
    start_P2_SCORE = 1'b0;
    start_OVER = 1'b1;
    start_OVER_store = 1'd1;
end else begin
    start_MODE = 1'b0;
    start_P1_SCORE = 1'b0;
    start_P2_SCORE = 1'b0;
    start_OVER = 1'b0; end
end
default: begin
    start_MODE = 1'b0;
    start_P1_SCORE = 1'b0;
    start_P2_SCORE = 1'b0;
    start_OVER = 1'b0; end
endcase
end
endmodule

```

[Code 3.2] top_bgm.v

```

module fre_divider ( input clk, output reg clk_10Hz );
reg [27:0] counter = 28'd0;

always @(posedge clk) begin
if (counter >= 28'd4000000)begin//10000000
    counter <= 28'd0; // Divide the clock frequency by 100000
    clk_10Hz <= 1'd1; end
else begin
    counter <= counter + 28'd1;
    clk_10Hz <= 1'd0; end
end
endmodule

```

[Code 3.3] fre_divider.v

```
module bps(input clk,input [7:0] bps,output reg bump);
reg [7:0] counter = 8'd0;
always@(posedge clk)begin
if(counter >= bps) begin
    bump = 1'd1;
    counter = 8'd0; end
else begin
    bump = 1'd0;
    counter = counter + 8'd1; end
end
endmodule
```

[Code 3.4] bps.v

```
module fre_divider_inv(input clk,input [27:0] number,output reg clk_after);
reg [27:0] counter = 28'd0;
always@(posedge clk)begin
if(counter > number)
begin
    clk_after = 1'd1;
    counter = 28'd0;
end else if(counter < 28'd110000) begin
    counter = counter + 28'd1;
    clk_after = 1'd1; end
else begin
    clk_after = 1'd0;
    counter = counter + 28'd1; end
end
endmodule
```

[Code 3.6] fre_divider_inv.v

```
module music_7 (
    input bump, start, clk
    output reg voice_fre,
    output reg [3:0] counter );

initial begin
    counter = 8'd0; end
reg store_start = 1'd0;

wire voice_fre_1;

reg [27:0] number_1 = 28'd381679;//c4

always @(posedge clk) begin
if(start == 1'd1)
    store_start = 1'd1;
else if(counter >= 8'd1) begin
    store_start = 1'd0; end
end
endmodule
```

```

end

always @@(posedge bump) begin
    if(store_start == 1'd1)begin
        counter = counter + 8'd1; end
    else counter = 8'd0;
end

always@(*)begin
    if(store_start == 1'd1)
    begin
        case (counter)
            8'd1: begin
                voice_fre = voice_fre_1; end
            8'd2: begin
                voice_fre = voice_fre_1; end
            default: begin
                voice_fre = 1'd0; end
            endcase
        end
    else voice_fre = 1'd0; end

fre_divider_inv fre_1(.clk(clk), .clk_after(voice_fre_1), .number(number_1));

endmodule

```

[Code 3.7] music_7.v

```

`timescale 1ns / 1ps

module top_temp(
    input      CLK100MHZ,
    inout     TMP_SDA,      // i2c sda on temp sensor - bidirectional
    output     TMP_SCL,      // i2c scl on temp sensor
    output [6:0] SEG,        // 7 segments of each display
    output [7:0] AN,         // 8 anodes of 8 displays
    output [15:0] LED,       // nexys leds = binary temp in deg C or deg F
    output     speaker,
    output reg   en);

    wire w_200KHz;          // 200kHz SCL
    wire [7:0] c_data;       // 8 bits of Celsius temperature data
    wire [7:0] f_data;       // 8 bits of Fahrenheit temperature data

    // Instantiate i2c master
    i2c_master i2cmaster(
        .clk_200KHz(w_200KHz),
        .temp_data(c_data),
        .SDA(TMP_SDA),
        .SCL(TMP_SCL) );

    clkgen_200KHz clkgen(
        .clk_100MHz(CLK100MHZ),
        .clk_200KHz(w_200KHz) );

    seg7c segcontrol(

```

```

.clk_100MHz(CLK100MHZ),
.c_data(c_data),
.f_data(f_data),
.SEG(SEG),
.AN(AN);

temp_converter tempconv(
.c(c_data),
.f(f_data) );

assign LED[15:8] = f_data;
assign LED[7:0] = c_data;

always @(posedge CLK100MHZ)
if(c_data >= 40) en=1'b1;
else en=1'b0;

alarm audio( .clk(CLK100MHZ), .en(en), .speaker(speaker) );

endmodule

```

[Code 3.8] top_temp.v

```

`timescale 1ns / 1ps

module i2c_master(
  input clk_200KHz,           // i_clk
  inout SDA,                 // i2c standard interface signal
  output [7:0] temp_data,    // 8 bits binary representation of deg C
  output SCL                 // i2c standard interface signal - 10KHZ
);

wire SDA_dir;               // SDA direction signal

reg [3:0] counter = 4'b0;   // count up to 9
reg clk_reg = 1'b1;

assign SCL = clk_reg;
parameter [7:0] sensor_address_plus_read = 8'b1001_0111;// 0x97
reg [7:0] tMSB = 8'b0;      // Temp data MSB
reg [7:0] tLSB = 8'b0;      // Temp data LSB
reg o_bit = 1'b1;           // output bit to SDA - starts HIGH
reg [11:0] count = 12'b0;   // State Machine Synchronizing Counter
reg [7:0] temp_data_reg;

localparam [4:0] POWER_UP = 5'h00,
  START = 5'h01,
  SEND_ADDR6 = 5'h02,
  SEND_ADDR5 = 5'h03,
  SEND_ADDR4 = 5'h04,
  SEND_ADDR3 = 5'h05,
  SEND_ADDR2 = 5'h06,
  SEND_ADDR1 = 5'h07,
  SEND_ADDR0 = 5'h08,
  SEND_RW = 5'h09,
  REC_ACK = 5'h0A,

```

```

REC_MSB7  = 5'h0B,
            REC_MSB6  = 5'h0C,
            REC_MSB5  = 5'h0D,
            REC_MSB4  = 5'h0E,
            REC_MSB3  = 5'h0F,
            REC_MSB2  = 5'h10,
            REC_MSB1  = 5'h11,
            REC_MSB0  = 5'h12,
SEND_ACK  = 5'h13,
REC_LSB7   = 5'h14,
            REC_LSB6  = 5'h15,
            REC_LSB5  = 5'h16,
            REC_LSB4  = 5'h17,
            REC_LSB3  = 5'h18,
            REC_LSB2  = 5'h19,
            REC_LSB1  = 5'h1A,
            REC_LSB0  = 5'h1B,
NACK      = 5'h1C;

reg [4:0] state_reg = POWER_UP;
always @(posedge clk_200KHz) begin
    // Counters Logic
    if(counter == 9) begin
        counter <= 4'b0;
        clk_reg <= ~clk_reg; end
    else
        counter <= counter + 1;
        count <= count + 1;

    case(state_reg)
        POWER_UP : begin
            if(count == 12'd1999)
                state_reg <= START; end
        START    : begin
            if(count == 12'd2004)
                o_bit <= 1'b0;      // send START condition 1/4 clock after SCL goes high
            if(count == 12'd2013)
                state_reg <= SEND_ADDR6; end
        SEND_ADDR6 : begin
            o_bit <= sensor_address_plus_read[7];
            if(count == 12'd2033)
                state_reg <= SEND_ADDR5; end
            SEND_ADDR5 : begin
                o_bit <= sensor_address_plus_read[6];
                if(count == 12'd2053)
                    state_reg <= SEND_ADDR4; end
                SEND_ADDR4 : begin
                    o_bit <= sensor_address_plus_read[5];
                    if(count == 12'd2073)
                        state_reg <= SEND_ADDR3; end
                    SEND_ADDR3 : begin
                        o_bit <= sensor_address_plus_read[4];
                        if(count == 12'd2093)
                            state_reg <= SEND_ADDR2; end
                        SEND_ADDR2 : begin
                            o_bit <= sensor_address_plus_read[3];
                            if(count == 12'd2113)
                                state_reg <= SEND_ADDR1; end
                        SEND_ADDR1 : begin

```

```

o_bit <= sensor_address_plus_read[2];
if(count == 12'd2133)
    state_reg <= SEND_ADDR0; end
    SEND_ADDR0 : begin
o_bit <= sensor_address_plus_read[1];
if(count == 12'd2153)
    state_reg <= SEND_RW; end
    SEND_RW : begin
o_bit <= sensor_address_plus_read[0];
if(count == 12'd2169)
    state_reg <= REC_ACK; end
REC_ACK : begin
if(count == 12'd2189)
    state_reg <= REC_MSB7; end
REC_MSB7 : begin
tMSB[7] <= i_bit;
if(count == 12'd2209)
    state_reg <= REC_MSB6; end
REC_MSB6 : begin
tMSB[6] <= i_bit;
if(count == 12'd2229)
    state_reg <= REC_MSB5; end
REC_MSB5 : begin
tMSB[5] <= i_bit;
if(count == 12'd2249)
    state_reg <= REC_MSB4; end
REC_MSB4 : begin
tMSB[4] <= i_bit;
if(count == 12'd2269)
    state_reg <= REC_MSB3; end
REC_MSB3 : begin
tMSB[3] <= i_bit;
if(count == 12'd2289)
    state_reg <= REC_MSB2; end
REC_MSB2 : begin
tMSB[2] <= i_bit;
if(count == 12'd2309)
    state_reg <= REC_MSB1; end
REC_MSB1 : begin
tMSB[1] <= i_bit;
if(count == 12'd2329)
    state_reg <= REC_MSB0; end
REC_MSB0 : begin
o_bit <= 1'b0;
tMSB[0] <= i_bit;
if(count == 12'd2349)
    state_reg <= SEND_ACK; end
SEND_ACK : begin
if(count == 12'd2369)
    state_reg <= REC_LSB7; end
REC_LSB7 : begin
tLSB[7] <= i_bit;
if(count == 12'd2389)
state_reg <= REC_LSB6; end
REC_LSB6 : begin
tLSB[6] <= i_bit;
if(count == 12'd2409) state_reg <= REC_LSB5; end
REC_LSB5 : begin

```

```

        tLSB[5] <= i_bit;
        if(count == 12'd2429) state_reg <= REC_LSB4; end
    REC_LSB4 : begin
        tLSB[4] <= i_bit;
        if(count == 12'd2449) state_reg <= REC_LSB3; end
    REC_LSB3 : begin
        tLSB[3] <= i_bit;
        if(count == 12'd2469) state_reg <= REC_LSB2; end
    REC_LSB2 : begin
        tLSB[2] <= i_bit;
        if(count == 12'd2489) state_reg <= REC_LSB1; end
    REC_LSB1 : begin
        tLSB[1] <= i_bit;
        if(count == 12'd2509) state_reg <= REC_LSB0; end
    REC_LSB0 : begin
        o_bit <= 1'b1;
        tLSB[0] <= i_bit;
        if(count == 12'd2529) state_reg <= NACK; end
    NACK : begin
        if(count == 12'd2559) begin
            count <= 12'd2000;
            state_reg <= START; end
        end
    endcase
end

always @(posedge clk_200KHz)
if(state_reg == NACK)
    temp_data_reg <= { tMSB[6:0], tLSB[7] };

assign SDA_dir = (state_reg == POWER_UP || state_reg == START || state_reg ==
SEND_ADDR6 || state_reg == SEND_ADDR5 ||
state_reg == SEND_ADDR4 || state_reg ==
SEND_ADDR3 || state_reg == SEND_ADDR2 || state_reg == SEND_ADDR1 ||
state_reg == SEND_ADDR0 || state_reg == SEND_RW || state_reg == SEND_ACK ||
state_reg == NACK) ? 1 : 0;
assign SDA = SDA_dir ? o_bit : 1'bz;
assign i_bit = SDA;
assign temp_data = temp_data_reg;

endmodule

```

[Code 3.9] iCc_master.v

```

`timescale 1ns / 1ps

module clkgen_200KHz(
    input clk_100MHz, //input 100MHz clock signal
    output clk_200KHz //generated output 200KHz clock signal );

reg [7:0] counter = 8'h00; //hexadecimal value - 8'b0 in binary - 0 value
reg clk_reg = 1'b1; // 1 bit initialized as 1

always @(posedge clk_100MHz) begin
    if(counter == 249) begin //if a cycle for the 200kHz clock is completed
        counter <= 8'h00; //reset counter
    end
end

```

```

    clk_reg <= ~clk_reg; //inverting to high/low logic level for output
end
else
    counter <= counter + 1; //incrementing by 1
end
assign clk_200KHz = clk_reg; //assign the state of the output signal (1 or 0)

endmodule

```

[Code 3.10] clkgen_200KHz.v

```

`timescale 1ns / 1ps
module seg7c(
    input clk_100MHz,           // Nexys 4 DDR clock
    input [7:0] c_data,          // Temp data from i2c master
    input [7:0] f_data,          // Temp data from temp converter
    output reg [6:0] SEG,        // 7 Segments of Displays
    output reg [7:0] AN          // 4 Anodes of 8 to display Temp C
);
wire [3:0] c_tens, c_ones;
assign c_tens = c_data / 10;      // Tens value of C temp data
assign c_ones = c_data % 10;      // Ones value of C temp data

wire [3:0] f_tens, f_ones;
assign f_tens = f_data / 10;      // Tens value of C temp data
assign f_ones = f_data % 10;      // Ones value of C temp data
parameter ZERO = 7'b000_0001; // 0
parameter ONE = 7'b100_1111; // 1
parameter TWO = 7'b001_0010; // 2
parameter THREE = 7'b000_0110; // 3
parameter FOUR = 7'b100_1100; // 4
parameter FIVE = 7'b010_0100; // 5
parameter SIX = 7'b010_0000; // 6
parameter SEVEN = 7'b000_1111; // 7
parameter EIGHT = 7'b000_0000; // 8
parameter NINE = 7'b000_0100; // 9
parameter DEG = 7'b001_1100; // degrees symbol
parameter C = 7'b011_0001; // C
parameter F = 7'b011_1000; // F
reg [2:0] anode_select; // 2 bit counter for selecting each of 4 digits
reg [16:0] anode_timer; // counter for digit refresh
always @(posedge clk_100MHz) begin

    if(anode_timer == 99_999) begin // The period of 100MHz clock is 10ns (1/100,000,000
seconds)
        anode_timer <= 0;           // 10ns x 100,000 = 1ms
        anode_select <= anode_select + 1; end
    else
        anode_timer <= anode_timer + 1; end

always @(anode_select) begin
    case(anode_select)
        3'o0 : AN = 8'b1111_1110;
        3'o1 : AN = 8'b1111_1101;
        3'o2 : AN = 8'b1111_1011;
        3'o3 : AN = 8'b1111_0111;
    endcase
end

```

```

3'o4 : AN = 8'b1110_1111;
3'o5 : AN = 8'b1101_1111;
3'o6 : AN = 8'b1011_1111;
3'o7 : AN = 8'b0111_1111;
endcase
end

always @*
case(anode_select)
3'o0 : SEG = C; // Set to C for Celsius
3'o1 : SEG = DEG; // Set to degrees symbol
3'o2 : begin // C TEMPERATURE ONES DIGIT
    case(c_ones)
        4'b0000 : SEG = ZERO;
        4'b0001 : SEG = ONE;
        4'b0010 : SEG = TWO;
        4'b0011 : SEG = THREE;
        4'b0100 : SEG = FOUR;
        4'b0101 : SEG = FIVE;
        4'b0110 : SEG = SIX;
        4'b0111 : SEG = SEVEN;
        4'b1000 : SEG = EIGHT;
        4'b1001 : SEG = NINE;
    endcase
end

3'o3 : begin // C TEMPERATURE TENS DIGIT
    case(c_tens)
        4'b0000 : SEG = ZERO;
        4'b0001 : SEG = ONE;
        4'b0010 : SEG = TWO;
        4'b0011 : SEG = THREE;
        4'b0100 : SEG = FOUR;
        4'b0101 : SEG = FIVE;
        4'b0110 : SEG = SIX;
        4'b0111 : SEG = SEVEN;
        4'b1000 : SEG = EIGHT;
        4'b1001 : SEG = NINE;
    endcase
end

3'o4 : SEG = F; // Set to F for Fahrenheit
3'o5 : SEG = DEG; // Set to degrees symbol
3'o6 : begin // F TEMPERATURE ONES DIGIT
    case(f_ones)
        4'b0000 : SEG = ZERO;
        4'b0001 : SEG = ONE;
        4'b0010 : SEG = TWO;
        4'b0011 : SEG = THREE;
        4'b0100 : SEG = FOUR;
        4'b0101 : SEG = FIVE;
        4'b0110 : SEG = SIX;
        4'b0111 : SEG = SEVEN;
        4'b1000 : SEG = EIGHT;
        4'b1001 : SEG = NINE;
    endcase
end

3'o7 : begin // F TEMPERATURE TENS DIGIT

```

```

case(f_tens)
  4'b0000 : SEG = ZERO;
  4'b0001 : SEG = ONE;
  4'b0010 : SEG = TWO;
  4'b0011 : SEG = THREE;
  4'b0100 : SEG = FOUR;
  4'b0101 : SEG = FIVE;
  4'b0110 : SEG = SIX;
  4'b0111 : SEG = SEVEN;
  4'b1000 : SEG = EIGHT;
  4'b1001 : SEG = NINE;
endcase
end
endcase
endmodule

```

[Code 3.11] seg7c.v

```

module temp_converter(
  input [7:0] c,
  output [7:0] f
);

  wire [15:0] p;
  wire [7:0] q;

  multiply_by_9 MULT9(.x(c), .y(p));
  divide_by_5 DIV5(.x(p), .y(q));
  add_32 ADD32(.x(q), .y(f));

endmodule

```

[Code 3.12] temp_converter

```

module temp_converter(
  input [7:0] c,
  output [7:0] f
);

  wire [15:0] p;
  wire [7:0] q;

  multiply_by_9 MULT9(.x(c), .y(p));
  divide_by_5 DIV5(.x(p), .y(q));
  add_32 ADD32(.x(q), .y(f));

endmodule

```

[Code 3.13] multiply_by_9

```

module multiply_by_9(
    input [7:0] x,
    output reg [15:0] y
);

    always @(*) begin
        y = x * 9;
    end
endmodule

```

[Code 3.14] divide_by_5

```

module divide_by_5(
    input [15:0] x,
    output reg [7:0] y
);

    always @(*) begin
        y = x / 5;
    end
endmodule

```

[Code 3.15] add_32.v

```

module alarm(
    input clk, en,
    output speaker );

reg [16:0] counter; //113632는 00011011101111100000 곧 2진수 17비트
reg speaker;

always @(posedge clk) begin
    if (counter == 113632) begin
        counter <= 0;
    if (en) begin
        speaker <= ~speaker; end
        end
    else begin
        counter <= counter + 1;end
        end
endmodule

```

[Code 3.16] alarm.v

```

module top_cam(
    input wire i_top_clk,
    input wire i_top_rst,
    input wire i_top_cam_start,

```

```

output wire o_top_cam_done,
output wire o_top_cam_ready,

input wire i_top_inc_sobel_thresh,
input wire i_top_dec_sobel_thresh,

input wire    i_top_pclk,
input wire [7:0] i_top_pix_byte,
input wire    i_top_pix_vsync,
input wire    i_top_pix_href,
output wire   o_top_reset,
output wire   o_top_pwdn,
output wire   o_top_xclk,
output wire   o_top_siod,
output wire   o_top_sioc,

output wire [3:0] o_top_vga_red,
output wire [3:0] o_top_vga_green,
output wire [3:0] o_top_vga_blue,
output wire    o_top_vga_vsync,
output wire    o_top_vga_hsync
);

assign o_top_cam_ready = ~o_top_cam_done;

wire      w_vp_top_data_ready;
wire      w_cam_top_data_valid;
wire      w_mem_top_data_ready;
wire      w_mem_top_data_valid;
wire [7:0] w_vp_top_data;
wire [7:0] w_vga_pix_data;
wire [18:0] w_vga_pix_addr;
reg r1_rstn_top_clk,  r2_rstn_top_clk;
reg r1_rstn_pclk,    r2_rstn_pclk;
reg r1_rstn_clk25m,  r2_rstn_clk25m;

wire w_clk25m;
clk_wiz_1 clk_gen (
    .clk_out1(w_clk25m),
    .clk_out2(o_top_xclk),
    .reset(i_top_RST),
    .locked(),
    .clk_in1(i_top_clk)
);
wire w_rstn_btn_db;

debouncer #( .DELAY(240_000) )
top_btn_db(
    .i_clk(i_top_clk),
    .i_btn_in(~i_top_RST),
    .o_btn_db(w_rstn_btn_db)
);
always @ (posedge i_top_clk or negedge w_rstn_btn_db)
begin
    if (!w_rstn_btn_db) {r2_rstn_top_clk, r1_rstn_top_clk} <= 0;
    else           {r2_rstn_top_clk, r1_rstn_top_clk} <= {r1_rstn_top_clk, 1'b1};
end
always @ (posedge w_clk25m or negedge w_rstn_btn_db)

```

```

begin
    if(!w_rstn_btn_db) {r2_rstn_clk25m, r1_rstn_clk25m} <= 0;
    else           {r2_rstn_clk25m, r1_rstn_clk25m} <= {r1_rstn_clk25m, 1'b1};
end
always @(posedge i_top_pclk or negedge w_rstn_btn_db)
begin
    if(!w_rstn_btn_db) {r2_rstn_pclk, r1_rstn_pclk} <= 0;
    else           {r2_rstn_pclk, r1_rstn_pclk} <= {r1_rstn_pclk, 1'b1};
end
cam_top #( .CAM_CONFIG_CLK(100_000_000) )
OV7670_cam(
    .i_clk(i_top_clk),
    .i_rstn_clk(r2_rstn_top_clk),
    .i_rstn_pclk(r2_rstn_pclk),
    .i_cam_start(i_top_cam_start),
    .o_cam_done(o_top_cam_done),
    .i_pclk(i_top_pclk),
    .i_pix_byte(i_top_pix_byte),
    .i_vsync(i_top_pix_vsync),
    .i_href(i_top_pix_href),
    .o_reset(o_top_reset),
    .o_pwdn(o_top_pwdn),
    .o_siiod(o_top_siiod),
    .o_sioc(o_top_sioc),
    .i_data_ready(w_vp_top_data_ready),
    .o_cam_data_valid(w_cam_top_data_valid),
    .o_cam_data(w_cam_top_data)
);

```

wire [7:0] w_sobel_thresh;

```

top_user_control control_filters (
    .i_clk(i_top_clk),
    .i_rstn(r2_rstn_top_clk),
    .i_inc_sobel_thresh(i_top_inc_sobel_thresh),
    .i_dec_sobel_thresh(i_top_dec_sobel_thresh),
    .o_sobel_thresh(w_sobel_thresh)
);

```

```

vp_top #( .DW(8), .RL(640) ) videoprocessing_sobel(
    .i_clk(i_top_clk),
    .i_rstn(r2_rstn_top_clk),
    .i_threshold(w_sobel_thresh),
    .o_data_ready(w_vp_top_data_ready),
    .i_data_valid(w_cam_top_data_valid),
    .i_data(w_cam_top_data),
    .i_data_ready(w_mem_top_data_ready),
    .o_data_valid(w_mem_top_data_valid),
    .o_data(w_vp_top_data)
);

```

```

mem_top #( .DW(8) ) memory (
    .i_clk(i_top_clk),
    .i_rstn(r2_rstn_top_clk),

```

```

.o_data_ready(w_mem_top_data_ready ),
.i_data_valid(w_mem_top_data_valid ),
.i_data(w_vp_top_data ),

.i_clk25m(w_clk25m ),
.i_vga_addr(w_vga_pix_addr ),
.o_vga_data(w_vga_pix_data )
);

vga_top display_interface (
.i_clk25m(w_clk25m ),
.i_rstn_clk25m(r2_rstn_clk25m ),

.o_VGA_x(),
.o_VGA_y(),
.o_VGA_vsync(o_top_vga_vsync ),
.o_VGA_hsync(o_top_vga_hsync ),
.o_VGA_video(),

.o_VGA_r(o_top_vga_red ),
.o_VGA_g(o_top_vga_green ),
.o_VGA_b(o_top_vga_blue ),

.i_pix_data(w_vga_pix_data ),
.o_pix_addr(w_vga_pix_addr )
);

endmodule

```

[Code 3.17] top_cam.v

```

module debouncer
#(parameter DELAY = 1_000_000)
(
    input wire i_clk,
    input wire i_btn_in,
    output wire o_btn_db
);

localparam MAX_COUNT = $clog2(DELAY);
reg [MAX_COUNT-1:0] counter;
reg             r_sample;

initial { counter, r_sample } = 0;

always @ (posedge i_clk)
begin
    if(i_btn_in !== r_sample && counter < DELAY)
        counter <= counter + 1'b1;
    else if(counter == DELAY)
        begin
            counter <= 0;
            r_sample <= i_btn_in;
        end
    end
else

```

```

        counter <= 0;
    end

assign o_btn_db = r_sample;

endmodule

```

[Code 3.18] debouncer.v

```

`timescale 1ns / 1ps

module cam_top
#( parameter CAM_CONFIG_CLK = 100_000_000)
(
    input wire      i_clk,
    input wire      i_rstn_clk,
    input wire      i_rstn_pclk,
    input wire      i_cam_start,
    output wire     o_cam_done,
    input wire      i_pclk,
    input wire [7:0] i_pix_byte,
    input wire      i_vsync,
    input wire      i_href,
    output wire     o_reset,
    output wire     o_pwdn,
    output wire     o_siiod,
    output wire     o_sioc,
    input wire      i_data_ready,
    output wire     o_cam_data_valid,
    output reg [11:0] o_cam_data
);

assign o_reset = 1; // 0: reset registers 1: normal mode
assign o_pwdn = 0; // 0: normal mode 1: power down mode

wire      w_start_db;
wire      w_pix_valid;
wire [11:0] w_pix_data;
wire      w_afifo_AE;

debouncer
#( .DELAY(240_000)      )
cam_btn_start_db
(
    .i_clk(i_clk      ),
    .i_btn_in(i_cam_start  ),
    // Debounced button to start cam init
    .o_btn_db(w_start_db  )
);

cam_init
#( .CLK_F(CAM_CONFIG_CLK      ),
    .SCCB_F(400_000)      )
configure_cam
(
    .i_clk(i_clk      ),
    .i_rstn(i_rstn_clk      ),

```

```

    .i_cam_init_start(w_start_db),
    .o_cam_init_done(o_cam_done),

    .o_siod(o_siod      ),
    .o_sioc(o_sioc      ),

    .o_data_sent_done(   ),
    .o_SCCB_dout(        )
);

cam_capture
cam_pixels
(
    .i_pclk(i_pclk      ),
    .i_rstn(i_rstn_pclk ),
    .i_vsync(i_vsync     ),
    .i_href(i_href      ),
    .i_cam_done(o_cam_done),
    .i_D(i_pix_byte   ),
    .o_pix_valid(w_pix_valid),
    .o_pix_data(w_pix_data)
);

wire [11:0] o_afifo_dat;
async_fifo
#( .DW(12           ),
  .AW(4           ))
cam_afifo
(
    .w_clk(i_pclk      ),
    .w_rstn(i_rstn_pclk ),
    .w_en(w_pix_valid  ),
    .i_dat(w_pix_data  ),
    .w_almost_full(    ),
    .w_full(          ),

    .r_clk(i_clk       ),
    .r_rstn(i_rstn_clk ),
    .r_en(i_data_ready ),
    .o_dat(o_afifo_dat ),
    .r_almost_empty(w_afifo_AE  ),
    .r_empty(          )
);

always @(posedge i_clk)
if(!i_rstn_clk) o_cam_data <= 0;
else o_cam_data <= o_afifo_dat;

assign o_cam_data_valid = !w_afifo_AE;

endmodule

```

[Code 3.19] cam_top.v

```

module cam_init
#(parameter CLK_F = 100_000_000,
  parameter SCCB_F = 400_000)

```

```

( input wire    i_clk,
  input wire    i_rstn,
  input wire    i_cam_init_start,
  output wire   o_sioc,
  output wire   o_siod,
  output wire   o_cam_init_done,
  output wire   o_data_sent_done,
  output wire [7:0] o_SCCB_dout
);

wire [7:0] w_cam_rom_addr;
wire [15:0] w_cam_rom_data;
wire [7:0] w_send_addr,  w_send_data;
wire      w_start_sccb,  w_ready_sccb;

cam_rom
OV7670_Registers
(
  .i_clk(i_clk),
  .i_rstn(i_rstn),
  .i_addr(w_cam_rom_addr),
  .o_dout(w_cam_rom_data)
);

cam_config
#( .CLK_F(CLK_F) )
OV7670_config
(
  .i_clk(i_clk),
  .i_rstn(i_rstn),
  .i_i2c_ready(w_ready_sccb),
  .o_i2c_start(w_start_sccb),
  .i_config_start(i_cam_init_start),
  .o_config_done(o_cam_init_done),
  .i_rom_data(w_cam_rom_data),
  .o_rom_addr(w_cam_rom_addr),
  .o_i2c_addr(w_send_addr),
  .o_i2c_data(w_send_data)
);

sccb_master
#( .CLK_F(CLK_F),
  .SCCB_F(SCCB_F) )
SCCB_HERE
(
  .i_clk(i_clk),
  .i_rstn(i_rstn),
  .i_read(1'b0),
  .i_write(1'b1),
  .i_start(w_start_sccb),
  .i_restart(1'b0),
  .i_stop(1'b0),
  .o_ready(w_ready_sccb),
  .i_din(w_send_data),
  .i_addr(w_send_addr),
  .o_dout(o_SCCB_dout),
  .o_done(o_data_sent_done),
  .o_ack(),
  .io_sda(o_sioc),
  .o_scl(o_siod)
);

```

```

);
endmodule

```

[Code 3.20] cam_init.v

```

module cam_rom
( input wire      i_clk,
  input wire      i_rstn,
  input wire [7:0] i_addr,
  output reg [15:0] o_dout
);

always @@(posedge i_clk or negedge i_rstn) begin
  if(!i_rstn) o_dout <= 0;
  else begin
    case(i_addr)
      0: o_dout <= 16'h12_80; // COM7:             Reset SCCB registers
      1: o_dout <= 16'hFF_F0; // Delay
      2: o_dout <= 16'h12_04; // COM7,
      3: o_dout <= 16'h11_00; // CLKRC
      4: o_dout <= 16'h0C_00; // COM3,
      5: o_dout <= 16'h3E_00; // COM14,
      6: o_dout <= 16'h04_00; // COM1,
      7: o_dout <= 16'h8C_02; // RGB444
      8: o_dout <= 16'h40_D0; // COM15,
      9: o_dout <= 16'h3a_04; // TSLB
     10: o_dout <= 16'h14_18; // COM9
     11: o_dout <= 16'h4F_B3; // MTX1
     12: o_dout <= 16'h50_B3; // MTX2
     13: o_dout <= 16'h51_00; // MTX3
     14: o_dout <= 16'h52_3d; // MTX4
     15: o_dout <= 16'h53_A7; // MTX5
     16: o_dout <= 16'h54_E4; // MTX6
     17: o_dout <= 16'h58_9E; // MTXS
     18: o_dout <= 16'h3D_C0; // COM13
bits, may be wrong?
     19: o_dout <= 16'h17_14; // HSTART
     20: o_dout <= 16'h18_02; // HSTOP
     21: o_dout <= 16'h32_80; // HREF
     22: o_dout <= 16'h19_03; // VSTART
     23: o_dout <= 16'h1A_7B; // VSTOP
     24: o_dout <= 16'h03_0A; // VREF
     25: o_dout <= 16'h0F_41; // COM6
     26: o_dout <= 16'h1E_00; // MVFP
     27: o_dout <= 16'h33_0B; // CHLF
     28: o_dout <= 16'h3C_78; // COM12
     29: o_dout <= 16'h69_00; // GFIX
     30: o_dout <= 16'h74_00; // REG74
     31: o_dout <= 16'hB0_84; // RSVD
color
     32: o_dout <= 16'hB1_0c; // ABLC1
     33: o_dout <= 16'hB2_0e; // RSVD
     34: o_dout <= 16'hB3_80; // THL_ST
     35: o_dout <= 16'h70_3a; // SCALING_XSC *Leave as default. No test pattern
output.
more magic internet values

```

Set RGB color output
Internal PLL matches input clock (24 MHz).
*Leave as default.
*Leave as default. No scaling, normal pclock
*Leave as default. Disable CCIR656
Enable RGB444 mode with xR GB.
Output full range for RGB 444.
set correct output data sequence (magic)
MAX AGC value x4
all of these are magical matrix coefficients

sets gamma enable, does not preserve reserved

start high 8 bits
stop high 8 bits //these kill the odd colored line
edge offset
start high 8 bits
stop high 8 bits
vsync edge offset
reset timings
disable mirror / flip //might have magic value of 03
magic value from the internet
no HREF when VSYNC low
fix gain control
Digital gain control
magic value from the internet *required* for good

*Leave as default. No test pattern

```

36: o_dout <= 16'h71_35; // SCALING_YSC      *Leave as default. No test pattern
output.
37: o_dout <= 16'h72_11; // SCALING DCWCTR    *Leave as default. Vertical down
sample by 2. Horizontal down sample by 2.
38: o_dout <= 16'h73_f0; // SCALING PCLK_DIV
39: o_dout <= 16'ha2_02; // SCALING PCLK DELAY  *Leave as deafult.
40: o_dout <= 16'h7a_20; // SLOP
41: o_dout <= 16'h7b_10; // GAM1
42: o_dout <= 16'h7c_1e; // GAM2
43: o_dout <= 16'h7d_35; // GAM3
44: o_dout <= 16'h7e_5a; // GAM4
45: o_dout <= 16'h7f_69; // GAM5
46: o_dout <= 16'h80_76; // GAM6
47: o_dout <= 16'h81_80; // GAM7
48: o_dout <= 16'h82_88; // GAM8
49: o_dout <= 16'h83_8f; // GAM9
50: o_dout <= 16'h84_96; // GAM10
51: o_dout <= 16'h85_a3; // GAM11
52: o_dout <= 16'h86_af; // GAM12
53: o_dout <= 16'h87_c4; // GAM13
54: o_dout <= 16'h88_d7; // GAM14
55: o_dout <= 16'h89_e8; // GAM15
56: o_dout <= 16'h13_e0; // COM8  disable AGC / AEC
57: o_dout <= 16'h00_00; // set gain reg to 0 for AGC
58: o_dout <= 16'h10_00; // set ARCJ reg to 0
59: o_dout <= 16'h0d_40; // magic reserved bit for COM4
60: o_dout <= 16'h14_18; // COM9, 4x gain + magic bit
61: o_dout <= 16'ha5_05; // BD50MAX
62: o_dout <= 16'hab_07; // DB60MAX
63: o_dout <= 16'h24_95; // AGC upper limit
64: o_dout <= 16'h25_33; // AGC lower limit
65: o_dout <= 16'h26_e3; // AGC/AEC fast mode op region
66: o_dout <= 16'h9f_78; // HAECC1
67: o_dout <= 16'ha0_68; // HAECC2
68: o_dout <= 16'ha1_03; // magic
69: o_dout <= 16'ha6_d8; // HAECC3
70: o_dout <= 16'ha7_d8; // HAECC4
71: o_dout <= 16'ha8_f0; // HAECC5
72: o_dout <= 16'ha9_90; // HAECC6
73: o_dout <= 16'haa_94; // HAECC7
74: o_dout <= 16'h13_a7; // COM8, enable AGC / AEC
75: o_dout <= 16'h69_06;
default: o_dout <= 16'hFF_FF;      //mark end of ROM
endcase
end
end
endmodule

```

[Code 3.21] cam_rom.v

```

module cam_config
#(parameter CLK_F = 100_000_000)
(
  input wire    i_clk,
  input wire    i_rstn,
  input wire    i_i2c_ready,
  input wire    i_config_start,

```

```

input wire [15:0] i_rom_data,
output reg [7:0] o_rom_addr,
output reg      o_i2c_start,
output reg [7:0] o_i2c_addr,
output reg [7:0] o_i2c_data,
output reg      o_config_done
);

localparam ten_ms_delay = (CLK_F * 10) / 1000;
localparam timer_size  = $clog2(ten_ms_delay);
reg [timer_size - 1: 0] timer;

localparam SM_IDLE = 0;
localparam SM_SEND = 1;
localparam SM_DONE = 2;
localparam SM_TIMER = 3;

reg [2:0] SM_state;
reg [2:0] SM_return_state;
reg [1:0] byte_index;

always @(posedge i_clk or negedge i_rstn)
begin
    if(i_rstn) begin
        o_config_done <= 0;
        byte_index   <= 0;
        o_rom_addr   <= 0;
        o_i2c_addr   <= 0;
        o_i2c_start  <= 0;
        o_i2c_data   <= 0;
        SM_state     <= SM_IDLE;
    end
    else begin
        case(SM_state)
            SM_IDLE:
                begin
                    SM_state <= (i_config_start) ? SM_SEND : SM_IDLE;
                end
            SM_SEND:
                begin
                    if(i_i2c_ready)
                        case(i_rom_data)
                            16'hFF_FF: SM_state <= SM_DONE;
                            16'hFF_F0: begin
                                SM_state     <= SM_TIMER;
                                SM_return_state <= SM_SEND;
                                timer        <= ten_ms_delay;
                                o_rom_addr   <= o_rom_addr + 1;
                            end
                end
            default:
                begin
                    SM_state     <= SM_TIMER;
                    SM_return_state <= SM_SEND;
                    timer        <= 1;
                    o_i2c_start  <= 1;
                    o_i2c_addr   <= i_rom_data[15:8];
                    o_i2c_data   <= i_rom_data[7:0];
                    o_rom_addr   <= o_rom_addr + 1;
                end
        endcase
    end
end

```

```

        endcase
    end
    SM_DONE:
    begin
        SM_state    <= SM_IDLE;
        o_config_done <= 1;
    end
    SM_TIMER:
    begin
        SM_state  <= (timer == 1) ? SM_return_state : SM_TIMER;
        timer     <= (timer == 1) ?      0      : timer - 1;
        o_i2c_start <= 0;
    end
endcase
end
end

endmodule

```

[Code 3.22] cam_config.v

```

module sccb_master
#(parameter CLK_F = 100_000_000,
parameter SCCR_F = 400_000)
(
    input wire      i_clk,
    input wire      i_rstn,

    input wire      i_read,      // I2C Commands. Assume read/write are mutually exclusive
    input wire      i_write,
    input wire      i_start,
    input wire      i_restart,
    input wire      i_stop,
    input wire [7:0] i_din,
    input wire [7:0] i_addr,

    output wire [7:0] o_dout,
    output wire      o_ready,
    output wire      o_done,      // 1-cycle tick when a transaction is completed
    output wire      o_ack,
    inout wire      io_sda,
    output wire      o_scl
);

localparam CAM_ADDR = 7'h21;
localparam [3:0] IDLE    = 0,
                START_1  = 1,
                START_2  = 2,
                WAIT     = 3,
                DATA_1   = 4,
                DATA_2   = 5,
                DATA_3   = 6,
                DATA_4   = 7,
                DATA_DONE = 8,
                RESTART  = 9,

```

```

    END_1    = 10,
    END_2    = 11;

localparam TIMER_WIDTH = $clog2(CLK_F/SCCB_F);
localparam HALF      = CLK_F/(2*SCCB_F);
localparam QUARTER   = HALF/2;

reg [TIMER_WIDTH - 1: 0] timer;
reg [3:0]           state;
reg [8:0]           r_data_bit_index;
reg [8:0]           r_tx;
reg [8:0]           r_rx;
reg [7:0]           r_latched_data, r_latched_addr;
reg [1:0]           r_byte_index;
reg                 data_state;
wire                i_sda;

reg r_done;
reg r_ready;

reg r_scl, r2_scl;
reg r_sda, r2_en_sda;
reg r_read;
reg r_write;

initial begin
    state    = IDLE;
    r_ready  = 1'b1;
    r_scl   = 1'b1;
    r_sda   = 1'b1;
    r2_scl  = 1'b1;
    r2_en_sda = 1'b1;
    r_done   = 1'b0;
end

always @(posedge i_clk or negedge i_rstn)
begin
    if(i_rstn) begin
        r2_scl  <= 1'b1;
        r2_en_sda <= 1'b1;
    end
    else begin
        r2_scl  <= r_scl;
        r2_en_sda <= r_sda;
    end
end

always @(posedge i_clk)
begin
    r_read  <= i_read;
    r_write <= i_write;
end

assign i_sda = (data_state && r_read) || (data_state && r_write && r_data_bit_index == 8);
assign o_scl = (r2_scl) ? 1'bZ : 1'b0;
assign io_sda = (i_sda || r2_en_sda) ? 1'bZ : 1'b0;

always @(posedge i_clk)
begin

```

```

timer <= timer + 1'b1;           // Free Running Counter
case(state)
    IDLE: begin                  // SDA and SCL line high; Wait for start command
        timer      <= 0;
        r_ready    <= 1'b1;
        r_done     <= 1'b0;
        data_state <= 1'b0;
        r_data_bit_index <= 9'b0;
        r_byte_index   <= 2'b0;
        r_scl       <= 1'b1;
        r_sda       <= 1'b1;
        r_latched_data <= 8'hZZ;
        r_latched_addr  <= 8'hZZ;
        if(i_start) begin
            state    <= START_1;
            timer    <= 0;
            r_latched_data <= i_din;
            r_latched_addr <= i_addr;
            r_ready  <= 1'b0;
        end
    end
    START_1: begin    // Bring SDA line low; Wait for 1/2 period of SCL
        r_sda    <= 1'b0;
        if(timer == (HALF-1)) begin
            timer <= 0;
            state <= START_2;
        end
    end
    START_2: begin    // Bring SCL line low; Wait for 1/2 period of SCL
        r_scl    <= 1'b0;
        if(timer == (HALF-1)) begin
            timer    <= 0;
            state    <= WAIT;
        end
    end
    WAIT: begin      // Both SCL/SDA low; Wait for Control Signal (Read or Write)
        r_scl    <= 1'b0;
        r_sda    <= 1'b0;
        timer    <= 0;
        r_data_bit_index <= 0;
        r_byte_index   <= r_byte_index + 1'b1;
        state     <= (r_byte_index == 3) ? END_1 : DATA_1;
        case(r_byte_index)          // 3-Phase Write Cycle (no ack in SCCB)
            2'b00: r_tx <= {CAM_ADDR, ~i_write, 1'b1}; // byte1 = {SLAVE ADDRESS, WR
BIT, Don't Care Bit}
            2'b01: r_tx <= {r_latched_addr, 1'b1};      // byte2 = {Register Addr, Don't
Care Bit}
            2'b10: r_tx <= {r_latched_data, 1'b1};      // byte3 = {Data to Register, Don't
Care Bit}
            default: r_tx <= {r_latched_data, 1'b1};
        endcase
        if((!i_write) && (!i_read)) begin
            if(i_stop)          state <= END_1;
            else if(i_restart || i_start) state <= RESTART;
            else                state <= IDLE;
        end
    end
    DATA_1: begin    // Load Data Bit to SDA before sampled by SCL

```

```

r_sda    <= r_tx[8];
r_scl    <= 1'b0;
data_state <= 1'b1;
if(timer == (QUARTER-1)) begin
    timer <= 0;
    state <= DATA_2;
end
end
DATA_2: begin // SCL Samples the Data Bit (Shift in for read/Shift out for write)
    r_sda <= r_tx[8];
    r_scl <= 1'b1;
    if(timer == (QUARTER-1)) begin
        timer <= 0;
        state <= DATA_3;
        r_rx <= {r_rx[7:0], io_sda};
    end
end
DATA_3: begin // Wait another quarter SCL cycle of it being HIGH
    r_sda <= r_tx[8];
    r_scl <= 1'b1;
    if(timer == (QUARTER-1)) begin
        timer <= 0;
        state <= DATA_4; // Shift Data In
    end
end
DATA_4: begin // Bring SCL Low again; Wait another quarter of a cycle
    r_sda    <= r_tx[8];
    r_scl    <= 1'b0;
    if(timer == (QUARTER-1)) begin
        timer <= 0;
        if(r_data_bit_index == 8) begin
            state    <= DATA_DONE;
            r_done   <= 1'b1; // Set done signal HIGH
            data_state <= 1'b0;
        end
        else begin
            r_tx      <= {r_tx[7:0], 1'b0};
            r_data_bit_index <= r_data_bit_index + 1'b1;
            state    <= DATA_1;
        end
    end
end
DATA_DONE: begin
    r_done <= 1'b0; // Set done signal LOW since it's a tick
    r_sda <= 1'b0;
    r_scl <= 1'b0;
    if(timer == (QUARTER-1)) begin
        timer <= 0;
        state <= WAIT;
    end
end
RESTART: begin
    if(timer == (HALF-1)) begin
        timer <= 0;
        state <= START_1;
    end
end
END_1: begin
    r_scl <= 1'b1;

```

```

r_sda <= 1'b0;
if(timer == (HALF-1)) begin
    timer <= 0;
    state <= END_2;
end
END_2: begin
    r_scl <= 1'b1;
    r_sda <= 1'b1;
    if(timer == (HALF-1)) begin
        timer <= 0;
        state <= IDLE;
    end
end
endcase
end

assign o_dout = r_rx[8:1];
assign o_ack = r_rx[0];
assign o_ready = r_ready;
assign o_done = r_done;

endmodule

```

[Code 3.23] sccbmaster.v

```

module cam_capture
(
    input wire      i_pclk,
    input wire      i_rstn,
    input wire      i_vsync,
    input wire      i_href,
    input wire [7:0] i_D,
    input wire      i_cam_done,
    output reg [11:0] o_pix_data,
    output reg      o_pix_valid
);
reg      r1_vsync,  r2_vsync;
wire     frame_start, frame_done;

initial { r1_vsync, r2_vsync } = 0;
always @(posedge i_pclk or negedge i_rstn)
if(!i_rstn) {r2_vsync, r1_vsync} <= 2'b00;
else       {r2_vsync, r1_vsync} <= {r1_vsync, i_vsync};

assign frame_start = (r1_vsync == 0) && (r2_vsync == 1); // Negative Edge of vsync
assign frame_done = (r1_vsync == 1) && (r2_vsync == 0); // Positive Edge of vsync
localparam [1:0] WAIT  = 2'd0,
                IDLE  = 2'd1,
                CAPTURE = 2'd2;

reg      r_half_data;
reg [1:0] SM_state;
reg [3:0] pixel_data;

always @(posedge i_pclk or negedge i_rstn)
if(!i_rstn)

```

```

begin
    SM_state  <= WAIT;
    o_pix_valid <= 0;
    o_pix_data  <= 0;
    r_half_data <= 0;
end
else
begin
    r_half_data <= 0;
    o_pix_valid <= 0;

    case(SM_state)
        WAIT:
            begin
                SM_state  <= (frame_start && i_cam_done) ? IDLE : WAIT;
            end
        IDLE:
            begin
                SM_state  <= (frame_start) ? CAPTURE : IDLE;
            end
        CAPTURE:
            begin
                SM_state  <= (frame_done) ? IDLE : CAPTURE;
                if(i_href)
                    begin
                        if(!r_half_data)
                            pixel_data <= i_D[3:0];
                        r_half_data  <= ~r_half_data;
                        o_pix_valid  <= (r_half_data) ? 1'b1 : 1'b0;
                        o_pix_data   <= (r_half_data) ? {pixel_data, i_D} : o_pix_data;
                    end
            end
    endcase
end
endmodule

```

[Code 3.24] cam_capture.v

```

`timescale 1ns / 1ps

module async_fifo
#( parameter DW = 4,
  parameter AW = 4 )
( input wire      w_clk,
  input wire      w_rstn,
  input wire      w_en,
  input wire [DW-1:0] i_dat,
  output reg      w_almost_full,
  output reg      w_full,

  input wire      r_clk,
  input wire      r_rstn,
  input wire      r_en,
  output wire [DW-1:0] o_dat,
  output reg      r_almost_empty,

```

```

    output reg      r_empty
  );
localparam AF = (1 << AW) - 4;

wire [AW - 1 : 0] waddr;           // write address in memory
wire [AW : 0] wgraynext;
wire [AW : 0] wbinnext;
wire      wfull_val;
wire      walmost_full_val;

reg [AW: 0] wbin;                // write pointer in binary to address memory
reg [AW: 0] wptr;                // write pointer in gray for (almost) empty/(almost) full flags
reg [AW: 0] wq1_rptr;
reg [AW: 0] wq2_rptr;

initial { wq2_rptr, wq1_rptr } = 0;
initial { wbin, wptra } = 0;
initial w_full = 0;
wire [AW - 1 :0] raddr;          // read address in memory
wire [AW :0] rgraynext;
wire [AW :0] rbinnext;
wire      rempty_val;
wire      ralmost_empty_val;

reg [AW :0] rbin;                // read pointer in binary to address memory
reg [AW :0] rptr;                // read pointer in gray for (almost) empty/(almost) full flags
reg [AW :0] rq1_wptr;
reg [AW :0] rq2_wptr;

initial { rq2_wptr, rq1_wptr } = 0;
initial { rbin, rptr } = 0;
initial r_empty = 1'b1;
reg [DW-1:0] mem [0:((1<<AW)-1)];
assign o_dat = mem[raddr];

always @(posedge w_clk)
  if((w_en) && (!w_full))
    mem[waddr] <= i_dat;

always @(posedge r_clk or negedge r_rstn)
begin
  if(!r_rstn) {rq2_wptr, rq1_wptr} <= 0;
  else      {rq2_wptr, rq1_wptr} <= {rq1_wptr, wptra};
end
always @(posedge r_clk or negedge r_rstn)
begin
  if(!r_rstn) {rbin, rptr} <= 0;
  else      {rbin, rptr} <= {rbinnext, rgraynext};
end
assign raddr   = rbin[AW-1:0];
assign rbinnext = rbin + { {(AW){1'b0}}, (r_en && (!r_empty)) };
assign rgraynext = (rbinnext >> 1) ^ rbinnext;
always @(posedge r_clk or negedge r_rstn)
begin
  if(!r_rstn) r_empty <= 1'b1;
  else      r_empty <= rempty_val;
end
assign rempty_val = (rq2_wptr == rgraynext);

```

```

wire [AW :0] rq2_wptr_bin;
wire [AW :0] rbin_wbin_diff;
assign rq2_wptr_bin[AW] = rq2_wptr[AW];
generate
    genvar k;
    for(k = AW - 1; k >= 0; k = k - 1)
        begin
            assign rq2_wptr_bin[k] = rq2_wptr_bin[k+1] ^ rq2_wptr[k];
        end
    endgenerate

assign rbin_wbin_diff = (rbinnext > rq2_wptr_bin) ? (rq2_wptr_bin - rbinnext
                                                       + (1 << (AW+1)))
                                                       : (rq2_wptr_bin - rbinnext);
assign ralmost_empty_val = (rbin_wbin_diff <= 4);

always @(posedge r_clk or negedge r_rstn)
begin
    if(!r_rstn) r_almost_empty <= 1'b1;
    else r_almost_empty <= ralmost_empty_val;
end

always @(posedge w_clk or negedge w_rstn)
begin
    if(!w_rstn) {wq2_rptr, wq1_rptr} <= 0;
    else {wq2_rptr, wq1_rptr} <= {wq1_rptr, rptr};
end

always @(posedge w_clk or negedge w_rstn)
begin
    if(!w_rstn) {wbin, wptr} <= 0;
    else {wbin, wptr} <= {wbinnext, wgraynext};
end

empty/empty flags
assign waddr = wbin[AW-1:0];
assign wbinnext = wbin + { {(AW){1'b0}}, (w_en && (!w_full)) };
assign wgraynext = (wbinnext >> 1) ^ wbinnext;

always @(posedge w_clk or negedge w_rstn)
begin
    if(!w_rstn) w_full <= 1'b0;
    else w_full <= wfull_val;
end

assign wfull_val = (wgraynext == {~wq2_rptr[AW:AW-1],
                                 wq2_rptr[AW-2:0]});

wire [AW :0] wq2_rptr_bin;
wire [AW :0] wbin_rbin_diff;
assign wq2_rptr_bin[AW] = wq2_rptr[AW];
generate
    genvar j;
    for(j = AW - 1; j >= 0; j = j - 1)
        begin
            assign wq2_rptr_bin[j] = wq2_rptr_bin[j+1] ^ wq2_rptr[j];
        end
    endgenerate

```

```

assign wbin_rbin_diff = (wbinnext > wq2_rptr_bin) ? (wbinnext - wq2_rptr_bin)
      : (wbinnext - wq2_rptr_bin
         + (1 << (AW+1)));
assign walmost_full_val = (wbin_rbin_diff >= AF);

always @(posedge w_clk or negedge w_rstn)
begin
  if(!w_rstn) w_almost_full <= 1'b0;
  else      w_almost_full <= walmost_full_val;
end

`ifdef FORMAL
reg f_past_valid_rd;
reg f_past_valid_wr;
reg f_past_valid_gbl;
initial
begin
  f_past_valid_rd = 0;
  f_past_valid_wr = 0;
  f_past_valid_gbl = 0;
end

always @($global_clock)
  f_past_valid_gbl <= 1'b1;

always @(posedge w_clk)
  f_past_valid_wr <= 1'b1;

always @(posedge r_clk)
  f_past_valid_rd <= 1'b1;

always @(*)
  if(!f_past_valid_gbl)
    assert((!f_past_valid_wr) && (!f_past_valid_rd));
`ifdef AFIFO
localparam F_CLKBITS = 5;
wire [F_CLKBITS-1:0] f_wclk_step;
wire [F_CLKBITS-1:0] f_rclk_step;

assign f_wclk_step = $anyconst;
assign f_rclk_step = $anyconst;
always @(*)
begin
  assume(f_wclk_step != 0);
  assume(f_rclk_step != 0);
end

reg [F_CLKBITS-1:0] f_wclk_count;
reg [F_CLKBITS-1:0] f_rclk_count;
always @($global_clock)
begin
  f_wclk_count <= f_wclk_count + f_wclk_step;
  f_rclk_count <= f_rclk_count + f_rclk_step;
end
always @(*)
begin
  assume(w_clk == (f_wclk_count[F_CLKBITS-1]));
  assume(r_clk == (f_rclk_count[F_CLKBITS-1]));
`endif
`endif

```

```

    end
`endif

initial assume(w_rstn == r_rstn);

always @($global_clock)
  assume($fell(w_rstn) == $fell(r_rstn));
always @($global_clock)
  if(!$rose(w_clk))
    assume(!$rose(w_rstn));
always @($global_clock)
  if(!$rose(r_clk))
    assume(!$rose(r_rstn));
always @($global_clock)
  if(!w_rstn)
    assert(rbin == 0);

always @($global_clock)
  if(f_past_valid_gbl)
begin
  if(!$rose(w_clk))
    begin
      assume($stable(w_en));
      assume($stable(i_dat));
      assert($stable(w_full) || (!w_rstn));
    end
  if(!$rose(r_clk))
    begin
      assume($stable(r_en));
      assert((r_empty) || ($stable(o_dat)));
      assert((!r_rstn) || ($stable(r_empty)));
    end
end
  always @($global_clock)
    if((!f_past_valid_wr) || (!w_rstn)) // Either at reset or at initial start of formal
begin
  assume(w_en == 0);

  assert(wptr == 0);
  assert(wbin == 0);
  assert(!w_full);

  assert(wq1_rptr == 0);
  assert(wq2_rptr == 0);
  assert(rq1_wptr == 0);
  assert(rq2_wptr == 0);
  assert(rbin == 0);
  assert(r_empty);
end
  always @($global_clock)
    if((!f_past_valid_rd) || (!r_rstn))
begin
  assume(r_en == 0);

  assert(rptr == 0);
  assert(rbin == 0);

```

```

    assert(wq1_rptr == 0);
    assert(wq2_rptr == 0);
    assert(rq1_wptr == 0);
    assert(rq2_wptr == 0);

end

wire [AW:0] f_fill;
assign f_fill = (wbin - rbin);
initial assert(f_fill == 0);
always @($global_clock)
    assert(f_fill <= { 1'b1, {(AW){1'b0}} });
always @($global_clock)
    if(f_fill == { 1'b1, {(AW){1'b0}} })
        assert(w_full);
always @($global_clock)
    if(f_fill == { 1'b0, {(AW){1'b1}} })
        assert((wfull_val)||(!w_en)||!(w_full));
always @($global_clock)
    if(f_fill == 0)
        assert(r_empty);
always @($global_clock)
    if(f_fill == 1)
        assert((rempy_val)||(!r_en)||!(r_empty));
always @(*)
begin
    assert(wptr == ((wbin >> 1) ^ wbin));
end
always @(*)
begin
    assert(rptr == ((rbin >> 1) ^ rbin));
end
always @(*)
assert( (rptr == { ~wptr[AW:AW-1], wptr[AW-2:0] })
    == (f_fill == { 1'b1, {(AW){1'b0}} }));
always @(*)
assert((rptr == wptr) == (f_fill == 0));
reg [AW:0] f_w2r_rbin;
reg [AW:0] f_w1r_rbin;
reg [AW:0] f_r2w_wbin;
reg [AW:0] f_r1w_wbin;
wire [AW:0] f_w2r_fill;
wire [AW:0] f_r2w_fill;
initial { f_w2r_rbin, f_w1r_rbin } = 0;
initial { f_r2w_wbin, f_r1w_wbin } = 0;
always @(posedge w_clk or negedge w_rstn)
    if(!w_rstn)
        { f_w2r_rbin, f_w1r_rbin } <= 0;
    else
        { f_w2r_rbin, f_w1r_rbin } <= { f_w1r_rbin, rbin };
always @(posedge r_clk or negedge r_rstn)
    if(!r_rstn)
        { f_r2w_wbin, f_r1w_wbin } <= 0;
    else
        { f_r2w_wbin, f_r1w_wbin } <= { f_r1w_wbin, wbin };
always @(*)
    assert(rq1_wptr == ((f_r1w_wbin >> 1)^f_r1w_wbin));
always @(*)
    assert(rq2_wptr == ((f_r2w_wbin >> 1)^f_r2w_wbin));

```

```

always @(*)
    assert(wq1_rptr == ((f_w1r_rbin >> 1)^f_w1r_rbin));
always @(*)
    assert(wq2_rptr == ((f_w2r_rbin >> 1)^f_w2r_rbin));
assign f_w2r_fill = wbin - f_w2r_rbin;
assign f_r2w_fill = f_r2w_wbin - rbin;
always @(*)
    assert(f_w2r_fill <= { 1'b1, {(AW){1'b0}} });
always @(*)
    assert(f_r2w_fill <= { 1'b1, {(AW){1'b0}} });
always @(*)
    if(wptr == { ~wq2_rptr[AW:AW-1], wq2_rptr[AW-2:0] })
        assert(w_full);
always @(*)
    if(rptr == rq2_wptr)
        assert(r_empty);
genvar i;
generate
    for(i = 0; i <= AW; i=i+1)
begin: CHECK_ONEHOT_WGRAY
    always @(*)
        assert((wptr[i] == wgraynext[i])
            ||(wptr ^ wgraynext ^ (1<<i) == 0));
    always @(*)
        assert((rq2_wptr[i] == rq1_wptr[i])
            ||(rq2_wptr ^ rq1_wptr ^ (1<<i) == 0));
end
endgenerate
genvar k;
generate
    for(k = 0; k <= AW; k=k+1)
begin: CHECK_ONEHOT_RGRAY
    always @(*)
        assert((rptr[k] == rgraynext[k])
            || (rptr ^ rgraynext ^ (1<<k) == 0));
    always @(*)
        assert((wq2_rptr[k] == wq1_rptr[k])
            ||(wq2_rptr ^ wq1_rptr ^ (1<<k) == 0));
end
endgenerate

`ifdef AFIFO

(* anyconst *) wire [AW:0] f_const_addr;
    wire [AW:0] f_const_next_addr;
assign f_const_next_addr = f_const_addr + 1'b1;

(* anyconst *) reg [DW-1:0] f_const_first_data;
(* anyconst *) reg [DW-1:0] f_const_next_data;

    reg f_addr_valid;
    reg f_addr_next_valid;
always @(*)
begin
    f_addr_valid = 1'b0;
    if((wbin > rbin) && (wbin > f_const_addr) && (rbin <= f_const_addr))
        f_addr_valid = 1'b1;
    else if((wbin < rbin) && (wbin > f_const_addr))
        f_addr_valid = 1'b1;

```

```

        else if((wbin < rbin) && (rbin <= f_const_addr))
            f_addr_valid = 1'b1;
    end

    always @(*)
    begin
        f_addr_next_valid = 1'b0;
        if((wbin > rbin)&&(wbin > f_const_next_addr)&&(rbin <= f_const_next_addr))
            f_addr_next_valid = 1'b1;
        else if ((wbin < rbin)&&(f_const_next_addr < wbin))
            f_addr_next_valid = 1'b1;
        else if ((wbin < rbin)&&(rbin <= f_const_next_addr))
            f_addr_next_valid = 1'b1;
    end

    reg f_first_in_fifo;
    reg f_second_in_fifo;
    reg f_both_in_fifo;

    always @(*)
        f_first_in_fifo = (f_addr_valid)
            && (mem[f_const_addr[AW-1:0]] == f_const_first_data);
    always @(*)
        f_second_in_fifo = (f_addr_next_valid)
            && (mem[f_const_next_addr[AW-1:0]]==f_const_next_data);
    always @(*)
        f_both_in_fifo = (f_first_in_fifo) && (f_second_in_fifo);

    reg f_wait_for_first_read;
    reg f_read_first;
    reg f_read_second;
    reg f_wait_for_second_read;

    always @(*)
        f_wait_for_first_read = (f_both_in_fifo)
            && ((!r_en)||!(f_const_addr != rbin)||!(r_empty));
    always @(*)
        f_read_first = (r_en) && (o_dat == f_const_first_data)&&(!r_empty)
            && (rbin == f_const_addr)&&(f_both_in_fifo);
    always @(*)
        f_wait_for_second_read = (f_second_in_fifo)
            && ((!r_en) || (r_empty))
            && (f_const_next_addr == rbin);
    always @(*)
        f_read_second = (r_en) && (o_dat == f_const_next_data) && (!r_empty)
            && (rbin == f_const_next_addr)
            && (f_second_in_fifo);

    always @($global_clock)
        if((f_past_valid_gbl) && (w_rstn))
            begin
                if(!$past(f_read_first))&&($past(f_both_in_fifo))
                    assert((f_wait_for_first_read) || (($rose(r_clk))&&(f_read_first)));
                if($past(f_read_first))
                    assert( (!$rose(r_clk))&&(f_read_first))
                        || ($rose(r_clk)&&(f_read_second)
                            || (f_wait_for_second_read)));
                if($past(f_wait_for_second_read))
                    assert((f_wait_for_second_read)

```

```

        || (($rose(r_clk)) && (f_read_second)));
    end
`endif

    always @(posedge w_clk)
        cover(w_rstn);
    always @(posedge r_clk)
        cover(r_rstn);
    always @($global_clock)
        if(f_past_valid_gbl)
            cover((r_empty)&&(!$past(r_empty)));
    always @(*)
        if(f_past_valid_gbl)
            cover(w_full);
    always @(posedge w_clk)
        if(f_past_valid_wr)
            cover($past(w_full)&&($past(w_en))&&(w_full));
    always @(posedge w_clk)
        if (f_past_valid_wr)
            cover($past(w_full)&&(!w_full));
    always @(posedge w_clk)
        cover((w_full)&&(w_en));
    always @(posedge w_clk)
        cover(w_en);
    always @(posedge r_clk)
        cover((r_empty) && (r_en));

`endif
endmodule

```

[Code 3.25] async_fifo

```

module top_user_control(
    input wire i_clk,
    input wire i_rstn,

    input wire i_inc_sobel_thresh,
    input wire i_dec_sobel_thresh,

    output reg [7:0] o_sobel_thresh
);

    wire w_inc_sobel_thresh;
    wire w_dec_sobel_thresh;

    debouncer
    #(.DELAY(240_000))
    inc_sobel_db
    (
        .i_clk(i_clk),
        .i_btn_in(i_inc_sobel_thresh),
        .o_btn_db(w_inc_sobel_thresh)
    );

    debouncer
    #(.DELAY(240_000))

```

```

dec_sobel_db
(
    .i_clk(i_clk),
    .i_btn_in(i_dec_sobel_thresh),
    .o_btn_db(w_dec_sobel_thresh)
);

reg r1_inc_sobel_thresh;
reg r2_inc_sobel_thresh;
wire p_inc_sobel_thresh;
reg r1_dec_sobel_thresh;
reg r2_dec_sobel_thresh;
wire p_dec_sobel_thresh;

always @(posedge i_clk)
begin
    if(!i_rstn)
        begin
            { r2_inc_sobel_thresh, r1_inc_sobel_thresh } <= 0;
        end
    else begin
        { r2_inc_sobel_thresh, r1_inc_sobel_thresh } <= { r1_inc_sobel_thresh, w_inc_sobel_thresh
    };
        end
    end

always @(posedge i_clk)
begin
    if(!i_rstn)
        begin
            { r2_dec_sobel_thresh, r1_dec_sobel_thresh } <= 0;
        end
    else begin
        { r2_dec_sobel_thresh, r1_dec_sobel_thresh } <= { r1_dec_sobel_thresh,
w_dec_sobel_thresh };
        end
    end

assign p_inc_sobel_thresh = (~r2_inc_sobel_thresh & r1_inc_sobel_thresh);
assign p_dec_sobel_thresh = (~r2_dec_sobel_thresh & r1_dec_sobel_thresh);

localparam IDLE = 0,
ACTIVE_INC = 1;

localparam ACTIVE_DEC = 1;

reg state_inc;
reg state_dec;
always @(posedge i_clk)
begin
    if(!i_rstn)
        begin
            state_inc    <= IDLE;
            state_dec   <= IDLE;
            o_sobel_thresh <= 16;
        end
    else begin
        case(state_inc)
            IDLE: begin

```

```

    if(p_inc_sobel_thresh)
        state_inc <= ACTIVE_INC;
    end
    ACTIVE_INC: begin
        if((o_sobel_thresh + 1) >= {8}{1'b1})
            o_sobel_thresh <= o_sobel_thresh;
        else
            o_sobel_thresh <= o_sobel_thresh + 1;

        state_inc     <= IDLE;
    end
    endcase
    case(state_dec)
    IDLE: begin
        if(p_dec_sobel_thresh)
            state_dec <= ACTIVE_DEC;
    end
    ACTIVE_DEC: begin
        if((o_sobel_thresh - 1) <= 0)
            o_sobel_thresh <= 0;
        else
            o_sobel_thresh <= o_sobel_thresh - 1;

        state_dec     <= IDLE;
    end
    endcase
end
end

endmodule

```

[Code 3.26] top_user_control.v

```

module vp_top
#( parameter DW = 8,
  parameter RL = 640 )
(
  input wire i_clk,
  input wire i_rstn,

  input wire [7:0] i_threshold,

  output reg      o_data_ready,
  input wire      i_data_valid,
  input wire [11:0] i_data,
  input wire      i_data_ready,
  output wire     o_data_valid,
  output wire [DW-1:0] o_data
);
  wire [11:0] w_gray_data;
  wire [DW-1:0] w_gray_byte;
  wire      w_gray_data_valid;
  wire [9*DW-1:0] w_vp_control_data;
  wire      w_vp_control_valid;

```

```

wire [DW-1:0] w_sobel_data;
wire      w_sobel_data_valid;
wire      w_fifo_AF;
wire      w_fifo_AE;
localparam IDLE = 0,
          TRANSFER = 1;
reg state;
reg r_data_valid;

always @(posedge i_clk)
begin
  if(!i_rstn)
    begin
      state     <= IDLE;
      o_data_ready <= 0;
      r_data_valid <= 0;
    end
  else begin
    case(state)
      IDLE: begin
        r_data_valid <= 0;
        o_data_ready <= 0;
        if(i_data_valid)
          begin
            o_data_ready <= (!w_fifo_AF);
            state     <= TRANSFER;
          end
        end
      TRANSFER: begin
        r_data_valid <= o_data_ready;
        o_data_ready <= 0;
        state <= IDLE;
      end
    endcase
  end
end
grayscale
vp_gray
(
  .i_clk(i_clk),
  .i_rstn(i_rstn),
  .i_data(i_data),
  .i_data_valid(r_data_valid),
  .o_gray_data(w_gray_data),
  .o_gray_data_valid(w_gray_data_valid)
);

assign w_gray_byte = w_gray_data[11:4];

vp_control
#( .DW(DW),
  .RL(RL) )
vp_lb
(
  .i_clk(i_clk),
  .i_rstn(i_rstn),

```

```

    .i_pixel_data(w_gray_byte      ),
    .i_pixel_data_valid(w_gray_data_valid),
    .o_pixel_data(w_vp_control_data   ),
    .o_pixel_valid(w_vp_control_valid  )
);
conv
#( .DW(DW)           )
vp_sobel
(
    .i_clk(i_clk        ),
    .i_rstn(i_rstn       ),
    .i_sobel_thresh(i_threshold ),
    .i_data(w_vp_control_data  ),
    .i_valid(w_vp_control_valid ),
    .o_data(w_sobel_data      ),
    .o_valid(w_sobel_data_valid )
);
sync_fifo
#( .DW(DW)           ,
    .AW(10            ),
    .AFW(9             ),
    .AEW(1            )
)
vp_outputFIFO
(
    .i_clk(i_clk        ),
    .i_rstn(i_rstn       ),
    .i_wr(w_sobel_data_valid  ),
    .i_data(w_sobel_data      ),
    .o_full(          ),
    .o_almost_full(w_fifo_AF  ),
    .i_rd(i_data_ready     ),
    .o_data(o_data        ),
    .o_empty(          ),
    .o_almost_empty(w_fifo_AE ),
    .o_fill(          )
);
assign o_data_valid = !w_fifo_AE;
endmodule

```

[Code 3.27] vp_top.v

```

module grayscale
(
    input wire    i_clk,
    input wire    i_rstn,
    input wire [11:0] i_data,
    input wire    i_data_valid,
    output reg [11:0] o_gray_data,
    output reg     o_gray_data_valid
)

```

```

);

wire [7:0] R;
wire [7:0] G;
wire [7:0] B;

assign R = (i_data[11:8] << 4);
assign G = (i_data[7:4] << 4);
assign B = (i_data[3:0] << 4);

always @(posedge i_clk)
begin
    if(!i_rstn)
        begin
            o_gray_data     <= 0;
            o_gray_data_valid <= 0;
        end
    else begin
        o_gray_data     <= 0;
        o_gray_data_valid <= 0;

        if(i_data_valid)
            begin
                o_gray_data <= (R >> 2) + (R >> 5) +
                    (G >> 1) + (G >> 4) +
                    (B >> 4) + (B >> 5);
                o_gray_data_valid <= 1;
            end
        end
    end
endmodule

```

[Code 3.28] grayscale.v

```

module vp_control
#( parameter DW = 8,
  parameter RL = 640)
(
  input wire      i_clk,
  input wire      i_rstn,
  input wire [DW-1:0] i_pixel_data,
  input wire      i_pixel_data_valid,
  output reg [9*DW-1:0] o_pixel_data,
  output wire      o_pixel_valid
);

localparam CW  = $clog2(RL);
localparam RL_T = 3*RL;
localparam CW_T = $clog2(RL_T);
reg [CW-1:0] pixelCounter;
reg [1:0]   currWrlinebuffer;
reg [3:0]   dataValidlinebuffer;

```

```

reg [CW-1:0] rdCounter;
reg [1:0] currRdlinebuffer;
reg [3:0] rdlinebuffer;
reg rd;
reg rd_state;
localparam IDLE = 0;
localparam READ = 1;
reg [CW_T - 1: 0] totalpixelCounter;
wire [3*DW - 1: 0] lb0data,
    lb1data,
    lb2data,
    lb3data;
always @(posedge i_clk)
begin
    if(!i_rstn)
        totalpixelCounter <= 0;
    else begin

        if(i_pixel_data_valid && !rd)
            totalpixelCounter <= totalpixelCounter + 1'b1;
        else if(i_pixel_data_valid && rd)
            totalpixelCounter <= totalpixelCounter - 1'b1;
    end
end

always @(posedge i_clk)
begin
    if(!i_rstn)
        begin
            rd      <= 0;
            rd_state  <= IDLE;
        end
    else begin
        case(rd_state)
            IDLE: begin
                if(totalpixelCounter >= (3*RL - 1))
                    begin
                        rd      <= 1'b1;
                        rd_state  <= READ;
                    end
            end
            READ: begin
                if(rdCounter == (RL - 1))
                    begin
                        rd <= 0;
                        rd_state <= IDLE;
                    end
            end
        endcase
    end
end

always @(posedge i_clk)
begin
    if(!i_rstn)
        begin
            pixelCounter <= 0;
        end
    else begin

```

```

        if(i_pixel_data_valid)
            pixelCounter <= (pixelCounter == (RL-1)) ? 0 : pixelCounter + 1'b1;
    end
end

always @(posedge i_clk)
begin
    if(!i_rstn)
    begin
        currWrlinebuffer <= 0;
    end
    else begin
        if((pixelCounter == (RL-1)) && i_pixel_data_valid)
        begin
            currWrlinebuffer <= currWrlinebuffer + 1'b1;
        end
    end
end

always @(*)
begin
    dataValidlinebuffer = 4'h0;
    dataValidlinebuffer[currWrlinebuffer] = i_pixel_data_valid;
end

always @(posedge i_clk)
begin
    if(!i_rstn)
        rdCounter <= 0;
    else begin
        if(rd)
            rdCounter <= (rdCounter == (RL - 1)) ? 0 : rdCounter + 1'b1;
    end
end

always @(posedge i_clk)
begin
    if(!i_rstn)
    begin
        currRdlinebuffer <= 0;
    end
    else begin
        if((rdCounter == (RL-1)) && rd)
        begin
            currRdlinebuffer <= currRdlinebuffer + 1'b1;
        end
    end
end

always @(*)
begin
    case(currRdlinebuffer)
        0: begin
            rdlinebuffer[0] = rd;
            rdlinebuffer[1] = rd;
            rdlinebuffer[2] = rd;
            rdlinebuffer[3] = 0;
        end
        1: begin

```

```

rdlinebuffer[0] = 0;
rdlinebuffer[1] = rd;
rdlinebuffer[2] = rd;
rdlinebuffer[3] = rd;
end
2: begin
    rdlinebuffer[0] = rd;
    rdlinebuffer[1] = 0;
    rdlinebuffer[2] = rd;
    rdlinebuffer[3] = rd;
end
3: begin
    rdlinebuffer[0] = rd;
    rdlinebuffer[1] = rd;
    rdlinebuffer[2] = 0;
    rdlinebuffer[3] = rd;
end
endcase
end
assign o_pixel_valid = rd;

always @(*)
begin
    case(currRdlinebuffer)
        0: begin
            o_pixel_data = {lb2data, lb1data, lb0data};
        end
        1: begin
            o_pixel_data = {lb3data, lb2data, lb1data};
        end
        2: begin
            o_pixel_data = {lb0data, lb3data, lb2data};
        end
        3: begin
            o_pixel_data = {lb1data, lb0data, lb3data};
        end
    endcase
end

linebuffer
#( .DW(DW),
    .RL(RL) )
IB0
(
    .i_clk(i_clk),
    .i_rstn(i_rstn),
    .i_wr_data(dataValidlinebuffer[0]),
    .i_data(i_pixel_data),
    .i_rd_data(rdlinebuffer[0]),
    .o_data(lb0data)
);
linebuffer
#( .DW(DW),
    .RL(RL) )
IB1
(
    .i_clk(i_clk),

```

```

    .i_rstn(i_rstn          ),
    .i_wr_data(dataValidlinebuffer[1]  ),
    .i_data(i_pixel_data        ),
    .i_rd_data(rdlinebuffer[1]      ),
    .o_data(lb1data            )
);

linebuffer
#( .DW(DW),
  .RL(RL)           )
IB2
(
  .i_clk(i_clk          ),
  .i_rstn(i_rstn        ),
  .i_wr_data(dataValidlinebuffer[2]  ),
  .i_data(i_pixel_data        ),
  .i_rd_data(rdlinebuffer[2]      ),
  .o_data(lb2data            )
);

linebuffer
#( .DW(DW),
  .RL(RL)           )
IB3
(
  .i_clk(i_clk          ),
  .i_rstn(i_rstn        ),
  .i_wr_data(dataValidlinebuffer[3]  ),
  .i_data(i_pixel_data        ),
  .i_rd_data(rdlinebuffer[3]      ),
  .o_data(lb3data            )
);

endmodule

```

[Code 3.29] vp_control.v

```

module linebuffer
#( parameter DW = 12,
  parameter RL = 640)
( input wire      i_clk,
  input wire      i_rstn,
  input wire      i_wr_data,
  input wire [DW-1:0] i_data,
  input wire      i_rd_data,
  output wire [3*DW-1:0] o_data
);

localparam PW = $clog2(RL);

```

```

reg [DW-1:0] line [RL-1:0];
reg [PW-1:0] wptr;
reg [PW-1:0] rptr;

always @(posedge i_clk)
begin
    if(i_wr_data)
        line[wptr] <= i_data;
end

always @(posedge i_clk)
begin
    if(!i_rstn)
        wptr <= 0;
    else if(i_wr_data)
        wptr <= (wptr == RL - 1) ? 0 : wptr + 1'b1;
end

always @(posedge i_clk)
begin
    if(!i_rstn)
        rptr <= 0;
    else if(i_rd_data)
        rptr <= (rptr == RL - 1) ? 0 : rptr + 1'b1;
end
assign o_data = { line[rptr], line[rptr+1], line[rptr+2] };

endmodule

```

[Code 3.30] line_buffer.v

```

`module conv
#(parameter DW = 8)
(
    input wire      i_clk,
    input wire      i_rstn,
    input wire [7:0] i_sobel_thresh,
    input wire [9*DW-1: 0] i_data,
    input wire      i_valid,
    output reg [DW-1: 0] o_data,
    output reg       o_valid
);
integer i;
localparam KW = 8;
reg [KW-1:0] kernel1 [8:0];
reg [KW-1:0] kernel2 [8:0];
reg [10:0] multData1 [8:0];
reg [10:0] multData2 [8:0];
reg      multDataValid;
reg [10:0] sumDataInt1;

```

```

reg [10:0] sumDataInt2;
reg [10:0] sumData1;
reg [10:0] sumData2;
reg      sumDataValid;
reg [20:0] convolved_data_int1;
reg [20:0] convolved_data_int2;
wire[21:0] convolved_data_int;
reg      convolved_data_int_valid;

initial
begin
    kernel1[0] = 1;
    kernel1[1] = 0;
    kernel1[2] = -1;
    kernel1[3] = 2;
    kernel1[4] = 0;
    kernel1[5] = -2;
    kernel1[6] = 1;
    kernel1[7] = 0;
    kernel1[8] = -1;

    kernel2[0] = 1;
    kernel2[1] = 2;
    kernel2[2] = 1;
    kernel2[3] = 0;
    kernel2[4] = 0;
    kernel2[5] = 0;
    kernel2[6] = -1;
    kernel2[7] = -2;
    kernel2[8] = -1;
end

always @(posedge i_clk)
if(!i_rstn)
begin
    for(i=0;i<9;i=i+1)
    begin
        multData1[i] <= 0;
        multData2[i] <= 0;
    end
    multDataValid <= 0;
end
else begin
    for(i = 0;i<9;i=i+1)
    begin
        multData1[i] <= $signed(kernel1[i]) * $signed({1'b0, i_data[i*Dw+: Dw]});
        multData2[i] <= $signed(kernel2[i]) * $signed({1'b0, i_data[i*Dw+: Dw]});
    end
    multDataValid <= i_valid;
end

always @(*)
begin
    sumDataInt1 = 0;
    sumDataInt2 = 0;
    for(i=0;i<9;i=i+1)
    begin
        sumDataInt1 = $signed(sumDataInt1) + $signed(multData1[i]);
        sumDataInt2 = $signed(sumDataInt2) + $signed(multData2[i]);
    end
end

```

```

    end
end
always @(posedge i_clk)
if(!i_rstn)
begin
    sumData1    <= 0;
    sumData2    <= 0;
    sumDataValid <= 0;
end
else begin
    sumData1    <= sumDataInt1;
    sumData2    <= sumDataInt2;
    sumDataValid <= multDataValid;
end
always @(posedge i_clk)
if(!i_rstn)
begin
    convolved_data_int1    <= 0;
    convolved_data_int2    <= 0;
    convolved_data_int_valid <= 0;
end
else begin
    convolved_data_int1    <= $signed(sumData1) * $signed(sumData1);
    convolved_data_int2    <= $signed(sumData2) * $signed(sumData2);
    convolved_data_int_valid <= sumDataValid;
end
assign convolved_data_int = convolved_data_int1 + convolved_data_int2;

always @(posedge i_clk)
if(!i_rstn)
begin
    o_data <= 0;
    o_valid <= 0;
end
else begin
    o_data <= (convolved_data_int > i_sobel_thresh) ? {(DW){1'b1}} : 0;
    o_valid <= convolved_data_int_valid;
end
endmodule

```

[Code 3.31] conv.v

```

module sync_fifo
#(
    parameter DW = 8,
    parameter AW = 4,
    parameter AFW = 2,
    parameter AEW = 2 )
(
    input wire      i_clk,
    input wire      i_rstn,
    input wire      i_wr,
    input wire [DW-1:0] i_data,
    output reg      o_full,
    output reg      o_almost_full,

```

```

input wire      i_rd,
output reg [DW-1:0] o_data,
output reg      o_empty,
output reg      o_almost_empty,
output reg [AW:0] o_fill
);
    reg [DW-1:0] mem [0: (1<<AW)-1];

reg [AW:0] wptr;
reg [AW:0] rptr;
reg [AW-1:0] rptr_nxt;
wire w_rd = i_rd && !o_empty;
wire w_wr = i_wr && !o_full;
always @(posedge i_clk)
    if(w_wr)
        mem[wptr[AW-1:0]] <= i_data;
always @(posedge i_clk)
    o_data <= mem[ (w_rd) ? rptr_nxt : rptr[AW-1:0] ];
always @(posedge i_clk or negedge i_rstn)
begin
    if(!i_rstn)
        begin
            wptr <= 0;
            rptr <= 0;
        end
    else begin
        if(w_wr)
            wptr <= wptr + 1'b1;
        if(w_rd)
            rptr <= rptr + 1'b1;
    end
end
always @(*)
    rptr_nxt = rptr[AW-1:0] + 1'b1;
always @(posedge i_clk or negedge i_rstn)
begin
    if(!i_rstn)
        begin
            o_fill      <= 0;
            o_full     <= 0;
            o_almost_full <= 0;
            o_almost_empty <= 1'b1;
            o_empty    <= 1'b1;
        end
    else begin
        if(w_rd && (!w_wr))
            o_fill <= o_fill - 1'b1;
        else if((!w_rd) && w_wr)
            o_fill <= o_fill + 1'b1;
        if((o_fill > 1) || ((o_fill == 1) && (!w_rd)))
            o_empty <= 0;
        else
            o_empty <= 1'b1;
        if((o_fill > (1<<AEW)) || ((o_fill == (1<<AEW)) && (!w_rd)))
            o_almost_empty <= 0;
        else
            o_almost_empty <= 1'b1;
        if((o_fill < { 1'b0, {(AW){1'b1}}}) || ((o_fill == { 1'b0, {(AW){1'b1}}}) && (!w_wr)))
            o_full <= 0;
    end
end

```

```

        else
            o_full <= 1'b1;
        if((o_fill < { 1'b0, {(AW-AFW){1'b0}}, {(AFW}{1'b1}}}) || ((o_fill == { 1'b0, {(AW-AFW){1'b0}}, {(AFW}{1'b1}}}) && (!w_wr)))
            o_almost_full <= 0;
        else
            o_almost_full <= 1'b1;
    end
end

endmodule

```

[Code 3.32] sync_fifo.v

```

#( parameter DW = 8)
(
    input wire i_clk,
    input wire i_clk25m,
    input wire i_rstn,
    output reg      o_data_ready,
    input wire      i_data_valid,
    input wire [DW-1:0] i_data,
    input wire [18:0] i_vga_addr,
    output wire [DW-1:0] o_vga_data
);
localparam IDLE = 0,
    TRANSFER = 1;
reg      state;
reg      r_bram_wr;
reg [18:0] r_vp_addr;

always @(posedge i_clk)
begin
    if(!i_rstn)
begin
    state      <= IDLE;
    o_data_ready <= 0;
    r_bram_wr <= 0;
    r_vp_addr <= 0;
end
else begin
    case(state)
        IDLE: begin
            o_data_ready <= 0;
            r_bram_wr  <= 0;

            if(i_data_valid)
begin
                state      <= TRANSFER;
                r_bram_wr  <= 1'b1;
                o_data_ready <= 1'b1;
end
        end
        TRANSFER: begin
            state      <= IDLE;
            r_bram_wr  <= 0;
        end
    endcase
end
end

```

```

    o_data_ready <= 0;
    r_vp_addr <= (r_vp_addr == 307199) ? 0 : r_vp_addr + 1'b1;

end
endcase
end
end

mem_bram
#( .WIDTH(DW),
  .DEPTH(640*480) )
pixel_memory
(
  .i_wclk(i_clk),
  .i_wr(r_bram_wr),
  .i_wr_addr(r_vp_addr),
  .i_bram_data(i_data),
  .i_bram_en(1'b1),
  .i_rclk(i_clk25m),
  .i_rd(1'b1),
  .i_rd_addr(i_vga_addr),
  .o_bram_data(o_vga_data)
);

endmodule

`timescale 1ns / 1ps

module mem_bram
#(parameter WIDTH = 11,
  parameter DEPTH = 640*480)
( input wire      i_wclk,
  input wire      i_wr,
  input wire [${clog2(DEPTH)-1}:0] i_wr_addr,
  input wire      i_rclk,
  input wire      i_rd,
  input wire [${clog2(DEPTH)-1}:0] i_rd_addr,
  input wire      i_bram_en,
  input wire [WIDTH-1:0]   i_bram_data,
  output reg [WIDTH-1:0]  o_bram_data
);
reg [WIDTH-1:0] ram [0:DEPTH-1];

always @(posedge i_wclk)
if(i_bram_en)
  if(i_wr)
    ram[i_wr_addr] <= i_bram_data;

always @(posedge i_rclk)
if(i_rd)
  o_bram_data <= ram[i_rd_addr];

endmodule

```

[Code 3.33] mem_top.v

```

module mem_bram
#(parameter WIDTH = 11,
  parameter DEPTH = 640*480)
(
  input wire          i_wclk,
  input wire          i_wr,
  input wire [$clog2(DEPTH)-1:0] i_wr_addr,
  input wire          i_rclk,
  input wire          i_rd,
  input wire [$clog2(DEPTH)-1:0] i_rd_addr,
  input wire          i_bram_en,
  input wire [WIDTH-1:0]   i_bram_data,
  output reg [WIDTH-1:0]  o_bram_data
);
// Infer dual-port BRAM with dual clocks
// https://docs.xilinx.com/v/u/2019.2-English/ug901-vivado-synthesis (page 126)
reg [WIDTH-1:0] ram [0:DEPTH-1];
always @ (posedge i_wclk)
if(i_bram_en)
  if(i_wr)
    ram[i_wr_addr] <= i_bram_data;
always @ (posedge i_rclk)
if(i_rd)
  o_bram_data <= ram[i_rd_addr];
endmodule

```

[Code 3.34] mem_bram.v

```

`timescale 1ns / 1ps

module vga_top
(
  input wire      i_clk25m,
  input wire      i_rstn_clk25m,
  output wire [9:0] o_VGA_x,
  output wire [9:0] o_VGA_y,
  output wire      o_VGA_vsync,
  output wire      o_VGA_hsync,
  output wire      o_VGA_video,
  output reg [3:0] o_VGA_r,
  output reg [3:0] o_VGA_g,
  output reg [3:0] o_VGA_b,
  input wire [7:0] i_pix_data,
  output reg [18:0] o_pix_addr
);
vga_driver
#(.hDisp(640      ),
 .hFp(16       ),
 .hPulse(96     ),
 .hBp(48       ),
 .vDisp(480     ),
 .vFp(10       ),

```

```

.vPulse(2      ),
.vBp(33      )
vga_timing_signals
( .i_clk(i_clk25m      ),
.i_rstn(i_rstn_clk25m  ),

// VGA timing signals
.o_x_counter(o_VGA_x  ),
.o_y_counter(o_VGA_y  ),
.o_video(o_VGA_video  ),
.o_vsync(o_VGA_vsync  ),
.o_hsync(o_VGA_hsync  )
);
reg [1:0] r_SM_state;
localparam [1:0] WAIT_1 = 0,
               WAIT_2 = 'd1,
               READ   = 'd2;

always @@(posedge i_clk25m or negedge i_rstn_clk25m)
if(!i_rstn_clk25m)
begin
  r_SM_state <= WAIT_1;
  o_pix_addr <= 0;
end
else
begin
  case(r_SM_state)
    WAIT_1: r_SM_state <= (o_VGA_x == 640 && o_VGA_y == 480) ? WAIT_2 : WAIT_1;
    WAIT_2: r_SM_state <= (o_VGA_x == 640 && o_VGA_y == 480) ? READ : WAIT_2;

    READ: begin
      if((o_VGA_y < 480) && (o_VGA_x < 639))
        o_pix_addr <= (o_pix_addr == 307199) ? 0 : o_pix_addr + 1'b1;
      else begin
        if( (o_VGA_x == 799) && ( (o_VGA_y == 524) || (o_VGA_y < 480) ) )
          o_pix_addr <= o_pix_addr + 1'b1;
        else if(o_VGA_y >= 480)
          o_pix_addr <= 0;
      end
    end
  endcase
end

always @(*)
begin
  if(o_VGA_video)
    begin
      o_VGA_r = (i_pix_data[3:0]);
      o_VGA_g = (i_pix_data[3:0]);
      o_VGA_b = (i_pix_data[3:0]);
    end
  else begin
    o_VGA_r = 0;
    o_VGA_g = 0;
    o_VGA_b = 0;
  end
end

endmodule

```

[Code 3.35] vga_top.v

```

`timescale 1ns / 1ps

module vga_driver
#(parameter hDisp = 640,
 parameter hFp = 16,
 parameter hPulse = 96,
 parameter hBp = 48,
 parameter vDisp = 480,
 parameter vFp = 10,
 parameter vPulse = 2,
 parameter vBp = 33)
( input wire      i_clk,
  input wire      i_rstn,
  output wire [9:0] o_x_counter,
  output wire [9:0] o_y_counter,
  output wire      o_video,
  output wire      o_hsync,
  output wire      o_vsync
);

// Horizontal timing  hEND = 800
localparam hEND      = hDisp + hFp + hPulse + hBp;
localparam hSyncStart = hDisp + hFp;
localparam hSyncEnd   = hDisp + hFp + hPulse;

// Vertical timing   vEND = 524
localparam vEND      = vDisp + vFp + vPulse + vBp;
localparam vSyncStart = vDisp + vFp;
localparam vSyncEnd   = vDisp + vFp + vPulse;

reg [9:0] hc;
reg [9:0] vc;

always@(posedge i_clk or negedge i_rstn)
begin
  if(!i_rstn) begin
    hc  <= 0;
    vc  <= 0;
  end
  else begin
    if(hc == hEND-1)
      begin
        hc <= 0;
        if(vc == vEND-1)
          vc <= 0;
        else
          vc <= vc + 1'b1;
      end
    else
      hc <= hc + 1'b1;
  end
end

assign o_x_counter = hc;
assign o_y_counter = vc;

```

```

assign o_video = ((hc >= 0) && (hc < hDisp) && (vc >= 0) && (vc < vDisp));
assign o_hsync = ~((hc >= hSyncStart) && (hc < hSyncEnd));
assign o_vsync = ~((vc >= vSyncStart) && (vc < vSyncEnd));

endmodule

```

[Code 3.36] vga_driver.v

```

`timescale 1ns / 1ps

module VGA_mem_top(
    input      clk,          // Clock input 100MHZ
    input      rst,          // Active HIGH reset
    output [3:0] VGA_R, VGA_G, VGA_B,
    output     hsyncb,
    output     vsyncb,
    output reg   vga_ready
);

wire [11:0] bgr;
wire clk_100, clk_50, clk_25;
wire [18:0] address;
//wire [7:0] data1, data2; // Two data outputs for two .coe files
wire [7:0] data;        // Selected data (auto switching after 3 seconds)

reg [29:0] counter;    // Counter for 3-second delay
//reg switched;        // Flag to indicate data switch

// Clock core
clk_wiz_0 clk_inst (
    .clk_out1(clk_100),
    .clk_out2(clk_50),
    .clk_out3(clk_25),
    .reset(rst),
    .locked(),
    .clk_in1(clk)
);

// Frame buffer for .coe file 1
blk_mem_gen_0 frame_buffer_1 (
    .clka(clk_50),
    .wea(1'b0),      // Read-only
    .addra(address),
    .dina(8'd0),
    .douta(data)
);

always @(posedge clk_50 or posedge rst) begin
if (rst) begin
    counter <= 28'd0;
    vga_ready <= 1'b0;
end else begin
    if (counter >= 28'd250_000_000) begin // 5 seconds at 50MHz
        vga_ready <= 1'b1; // 5초 후 활성화
    end else begin
        counter <= counter + 1'b1; // 카운터 증가
    end
end

```

```

    end
end
end

// VGA Signal Assignment
assign {VGA_B, VGA_G, VGA_R} = bgr;

TOP_VGA vga (
    .clk(clk_25),
    .clk_en(1'b1), // Enables VGA
    .rst(rst),
    .din(data),
    .WES(),
    .addr(address),
    .bgr(bgr),
    .hsyncb(hsyncb),
    .vsyncb(vsyncb),
    .Yoffset_top(3'b000) // Fixed offset
);
endmodule

```

[Code 3.37] VGA_mem_top.v

```

`timescale 1ns / 1ps

module TOP_VGA #(
    parameter DATA_WIDTH = 8,
    parameter ADDR_WIDTH = 19
) (
    input wire clk, // 25MHz
    input wire rst,
    input wire clk_en,
    input wire [DATA_WIDTH-1:0] din,
    output wire WES,
    output wire [ADDR_WIDTH-1:0] addr,
    output wire [11:0] bgr,
    output wire hsyncb,
    output wire vsyncb,
    input wire [3:0] Yoffset_top );

    wire [9:0] Xoffset;
    wire [8:0] Yoffset;
    wire video_off;
    wire [DATA_WIDTH-1:0] Wdata;
    wire [ADDR_WIDTH-1:0] Waddr;
    wire [DATA_WIDTH-1:0] Rdata;
    wire [ADDR_WIDTH-1:0] raddr;
    wire [3:0] X_OFFSET = 4'b0000;
    wire [3:0] Y_OFFSET = Yoffset_top; // or 4'b0000
    wire [DATA_WIDTH-1:0] rgb8;

    assign Xoffset = {X_OFFSET, 6'd00};
    assign Yoffset = {Y_OFFSET, 5'd00};

```

```

localparam XRES = 10'd640;
localparam YRES = 9'd480;

assign bgr = { rgb8[7:6],2'b00, rgb8[5:3],1'b0, rgb8[2:0], 1'b0};

assign Rdata = din;
assign addr = raddr;
assign WES = 1'b0;

VGA_module vga (
    .clk(clk),
    .rst(rst),
    .clk_en(clk_en),
    .data(Rdata), // input
    .hsyncb(hsyncb), // output
    .vsyncb(vsyncb), // output
    .Xoffset(Xoffset), //input
    .Yoffset(Yoffset), // input
    .imageWidth(XRES), // input
    .imageHeight(YRES), // input
    .addr(raddr), // output
    .video_off(video_off), // output
    .rgb(rgb8) );
endmodule

```

[Code 3.38] TOP_VGA.v

```

module VGA_module(
    input wire clk, // 25MHz clock
    input wire clk_en,
    input wire rst,
        input wire [7:0] data,
        input wire [9:0] Xoffset,
        input wire [8:0] Yoffset,
    input wire [9:0] imageWidth,
    input wire [8:0] imageHeight,
        output wire hsyncb,
        output wire vsyncb,
        output wire [18:0] addr,
        output wire video_off,
    output wire [7:0] rgb
);

wire [9:0] hcnt, vcnt;
wire detect_neg_vsyncb;

vga_display VGAdisp (
    .clk(clk),
    .rst(rst),
    .clk_en(clk_en),
    .hcnt(hcnt),
    .vcnt(vcnd),
    .hsyncb(hsyncb),
    .vsyncb(vsyncb),
    .detect_neg_vsyncb(detect_neg_vsyncb),
);

```

```

        .video_off(video_off)
    );

vga_data VGAdata (
    .clk(clk),
    .rst(rst),
    .clk_en(clk_en),
    .hcnt(hcnt),
    .vcnt(vcnt),
    .Xposition1(Xoffset),
    .Yposition1(Yoffset),
    .imageWidth(imageWidth),
    .imageHeight(imageHeight),
    .data(data),
    .addr(addr),
    .detect_neg_vsyncb(detect_neg_vsyncb),
    .rgb(rgb)
);

endmodule

```

[Code 3.39] VGA_module.v

```

`timescale 1ns / 1ps

module vga_display(
    input wire clk,
    input wire clk_en,
    input wire rst,
    output reg [9:0] hcnt,
    output reg [9:0] vcnt,
    output reg hsyncb,
    output reg vsyncb,
    output wire detect_neg_vsyncb,
    output reg video_off );

reg vsyncb_1, vsyncb_2, vsyncb_3, vsyncb_4, vsyncb_5;
reg hsyncb_1, hsyncb_2, hsyncb_3, hsyncb_4, hsyncb_5;

localparam [9:0] ACTIVE_VIDEO_TIC = 640;
localparam [9:0] FRONT_PORCH_TIC = 16;
localparam [9:0] HRZ_SYNC_TIC = 96;
localparam [9:0] BACK_PORCH_TIC = 48;

localparam [9:0] ACTIVE_VERT_LINE = 480;
localparam [9:0] FRONT_PORCH_LINE = 10;
localparam [9:0] VERT_SYNC_LINE = 2;
localparam [9:0] BACK_PORCH_LINE = 29;

localparam [9:0] SCAN_LINE_TIME_MAX_TIC =
ACTIVE_VIDEO_TIC+FRONT_PORCH_TIC+HRZ_SYNC_TIC+BACK_PORCH_TIC;
localparam [9:0] LINE_NUM_LIM = 521;// 524

reg buf_hsyncb, buf_vsyncb;

always @(posedge clk, posedge rst)

```

```

if (rst)
    hcnt <= 'b0;
else if(clk_en) begin
if (hcnt == SCAN_LINE_TIME_MAX_TIC-1 )           // 799-48
    hcnt <= 0;
else
    hcnt <= hcnt + 1;
end

always @(posedge clk, posedge rst)
if (rst)
    buf_hsyncb <= 'b1;
else if(clk_en)
    buf_hsyncb <= hsyncb;

// hsyncb의 rising edge를 detect한다.
//assign detect_hsyncb = ( hsyncb & ~buf_hsyncb);

always @(posedge clk, posedge rst)
if (rst)
    vcnt <= 'b0;
else if(clk_en) begin
if (hcnt == SCAN_LINE_TIME_MAX_TIC-1) begin
    if ( vcnt == LINE_NUM_LIM-1)
        vcnt <= 0;
    else
        vcnt <= vcnt + 1;
end
end

always @(posedge clk, posedge rst)
if (rst) begin
    hsyncb <= 1; end
else if(clk_en) begin
    if ((hcnt >= 655+BACK_PORCH_TIC ) && (hcnt <
655+BACK_PORCH_TIC+HRZ_SYNC_TIC)) //96
//        if ((hcnt >= 655) && (hcnt < 655+HRZ_SYNC_TIC))
        hsyncb <= 'b0;
    else
        hsyncb <= 'b1;
end
always @(posedge clk, posedge rst)
if (rst) begin
    vsyncb <= 1; end
else if(clk_en) begin
    if ((vcnt >= 489) && (vcnt < 489 + VERT_SYNC_LINE))
        vsyncb <= 'b0;
    else
        vsyncb <= 'b1;
end

always @(posedge clk, posedge rst) begin
if(rst)
    buf_vsyncb <= 1;
else if (clk_en)
    buf_vsyncb <= vsyncb;
end

assign detect_neg_vsyncb = (buf_vsyncb & ~vsyncb);

```

```

always @ (posedge clk, posedge rst) begin
    if(rst)
        video_off <= 1;
    else begin if(clk_en)
        //if( (vcnt < BACK_PORCH_LINE) || (vcnt > (480+BACK_PORCH_LINE-1) && (vcnt <
LINE_NUM_LIM)))
            if( vcnt > 480 && vcnt < LINE_NUM_LIM)
                video_off <= 1;
            else
                video_off <= 0;
        end
    end
endmodule

```

[Code 3.40] vga_display.v

```

`timescale 1ns / 1ps

module vga_data(
    input wire clk,clk_en, rst,
    input wire [9:0] imageWidth,
    input wire [8:0] imageHeight,
    input wire [9:0] hcnt, vcnt,
    input wire [7:0] data,
    input wire [9:0] Xposition1,
    input wire [8:0] Yposition1,
    input wire detect_neg_vsyncb,
    output reg [18:0] addr,
    output reg [7:0] rgb);

localparam [9:0] BACK_PORCH_TIC = 48;

reg [1:0] cnt2;
wire [9:0] Xposition2, Yposition2;
wire inXrange, inYrange;
reg [7:0] rgb_buffer;
reg buff;
wire screen_on;
wire plus_addr;

assign Xposition2 = ((Xposition1 + imageWidth)> 640) ? 640:(Xposition1 + imageWidth);
assign Yposition2 = ((Yposition1 + imageHeight)>480)? 480:(Yposition1 + imageHeight);

assign inYrange = ((vcnt >= Yposition1) && (vcnt < Yposition2)) ? 'b1 : 'b0;
assign inXrange = ((hcnt >= (Xposition1 + BACK_PORCH_TIC)) && (hcnt < (Xposition2 + BACK_PORCH_TIC))) ? 'b1 : 'b0;

assign screen_on = (inXrange & inYrange) ;

always @ (posedge clk, posedge rst)
    if (rst) begin
        rgb_buffer <= 0;
        rgb <= 0; end
    else if(clk_en) begin

```

```

        if ((screen_on) || ((hcnt == Xposition2+BACK_PORCH_TIC) && inYrange))
            rgb <= data[7:0]; end
        else  rgb <= 0; end

assign plus_addr = screen_on;

always @(posedge clk, posedge rst)
    if (rst) addr <= 'b0;
    else if(clk_en) begin
        if (detect_neg_vsyncb == 'b1) begin
            addr <= 'b0; end
        else if (plus_addr == 'b1) begin
            addr <= addr + 1; end
    end
end

endmodule

```

[Code 3.41] vga_data.v

4. Simulation (시뮬레이션)

4-a. Simulation Code & Result

1. top_bgm의 testbench code

```

`timescale 1ns / 1ps

module top_bgm_tb;

reg clk, hi_wall, change_mode;
reg [2:0] state;
reg [5:0] ball_x;
wire voice_fre;

top_bgm uut (
    .clk(clk),
    .state(state),
    .hit_wall(hit_wall),
    .ball_x(ball_x),
    .change_mode(change_mode),
    .voice_fre(voice_fre) );

always begin
    clk = 1'b0;
    #5 clk = 1'b1;
    #5; end

initial begin
    clk = 0;
    state = 3'b000;
    hit_wall = 0;
    ball_x = 6'd20;
    change_mode = 0;

```

```

#10; state = 3'b001;
#10; state = 3'b010;
#10; ball_x = 6'd1;
#10; hit_wall = 1;
#10; state = 3'b011; hit_wall = 0;
#10; state = 3'b100;
#10; state = 3'b101;
#10; state = 3'b000;
#10; change_mode = 1;
#10; change_mode = 0;
#10; $stop;
end

initial begin
$monitor("At time %t, state = %b, hit_wall = %b, ball_x = %d, voice_fre = %b",
$time, state, hit_wall, ball_x, voice_fre); end

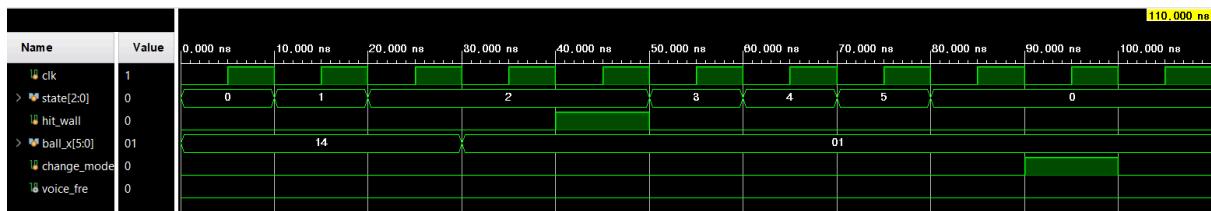
endmodule

```

[Testbench Code 4.1] top_bgm의 Testbench Code

해석: top_bgm 모듈은 주어진 상태와 입력 신호에 따라 여러 종류의 음악과 소리 주파수를 생성하여 voice_fre 출력으로 전달한다. 각 상태에 대해 다양한 음악이나 소리가 출력되며, 각 상태별 음악 생성 모듈은 bps와 fre_divider 모듈을 통해 주파수를 생성하고 이는 현재 상태의 진행에 맞춰 적절한 음향 효과를 제공한다.

state = MODE 상태일 때, voice_fre가 music_1의 주파수로 설정되고, start_MODE 신호가 활성화되어 music_1 모듈이 작동된다. 이때 counter가 12에 도달하면 store_start가 0으로 바뀌고, voice_fre는 0으로 돌아간다. 이는 아래의 waveform을 보면 쉽게 이해할 수 있다.



[Figure 4.1] top_bgm의 Testbench Code_Waveform

2. top_cam의 testbench code

```
'timescale 1ns / 1ps
```

```

module top_cam_tb;

reg i_top_clk;
reg i_top_rst;
reg i_top_cam_start;
reg i_top_inc_sobel_thresh;
reg i_top_dec_sobel_thresh;
reg i_top_pclk;

```

```

reg [7:0] i_top_pix_byte;
reg i_top_pix_vsync;
reg i_top_pix_href;

wire o_top_cam_done;
wire o_top_cam_ready;
wire o_top_reset;
wire o_top_pwdn;
wire o_top_xclk;
wire o_top_siod;
wire o_top_sioc;
wire [3:0] o_top_vga_red;
wire [3:0] o_top_vga_green;
wire [3:0] o_top_vga_blue;
wire o_top_vga_vsync;
wire o_top_vga_hsync;

top_cam uut (
    .i_top_clk(i_top_clk),
    .i_top_rst(i_top_rst),
    .i_top_cam_start(i_top_cam_start),
    .o_top_cam_done(o_top_cam_done),
    .o_top_cam_ready(o_top_cam_ready),
    .i_top_inc_sobel_thresh(i_top_inc_sobel_thresh),
    .i_top_dec_sobel_thresh(i_top_dec_sobel_thresh),
    .i_top_pcclk(i_top_pcclk),
    .i_top_pix_byte(i_top_pix_byte),
    .i_top_pix_vsync(i_top_pix_vsync),
    .i_top_pix_href(i_top_pix_href),
    .o_top_reset(o_top_reset),
    .o_top_pwdn(o_top_pwdn),
    .o_top_xclk(o_top_xclk),
    .o_top_siod(o_top_siod),
    .o_top_sioc(o_top_sioc),
    .o_top_vga_red(o_top_vga_red),
    .o_top_vga_green(o_top_vga_green),
    .o_top_vga_blue(o_top_vga_blue),
    .o_top_vga_vsync(o_top_vga_vsync),
    .o_top_vga_hsync(o_top_vga_hsync) );

always begin
    i_top_clk = 0;
    #5 i_top_clk = 1;
    #5;
end

always begin
    i_top_pcclk = 0;
    #5 i_top_pcclk = 1;
    #5;
end

initial begin
    i_top_rst = 0;
    i_top_cam_start = 0;
    i_top_inc_sobel_thresh = 0;
    i_top_dec_sobel_thresh = 0;
    i_top_pix_byte = 8'h00;
    i_top_pix_vsync = 0;

```

```

i_top_pix_href = 0;

#10; i_top_rst = 1;
#10; i_top_rst = 0;
#10; i_top_cam_start = 1;
#10; i_top_cam_start = 0;
#10; i_top_inc_sobel_thresh = 1;
#10; i_top_inc_sobel_thresh = 0; i_top_dec_sobel_thresh = 1;
#10; i_top_dec_sobel_thresh = 0;

#10;
i_top_pix_byte = 8'hAA; i_top_pix_vsync = 1; i_top_pix_href = 1;
#10; i_top_pix_vsync = 0; i_top_pix_href = 0;

#10; i_top_pix_byte = 8'h55; i_top_pix_vsync = 1; i_top_pix_href = 1;
#10; i_top_pix_vsync = 0; i_top_pix_href = 0;
#10; $stop;
end

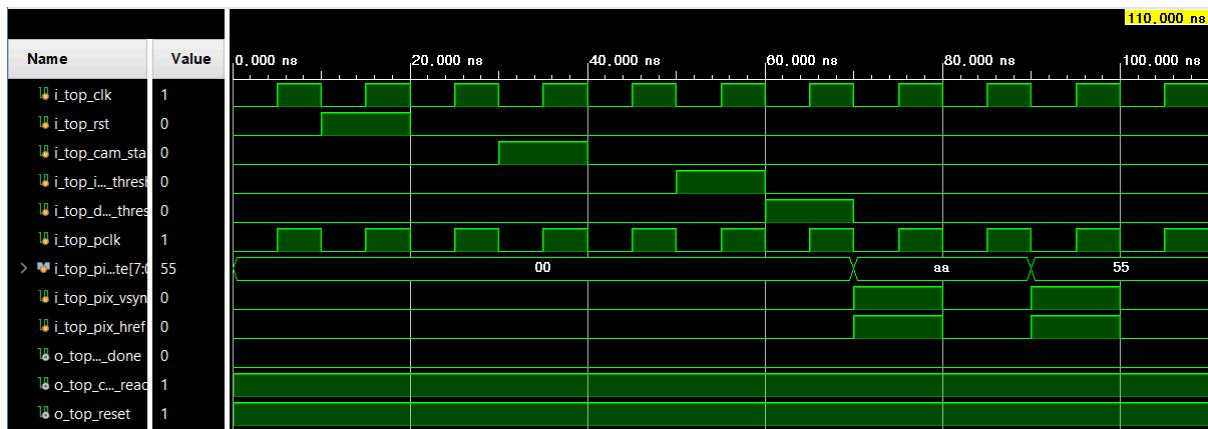
initial begin
$monitor("At time %t, o_top_cam_done = %b, o_top_cam_ready = %b, o_top_vga_red = %h,
o_top_vga_green = %h, o_top_vga_blue = %h, o_top_vga_vsync = %b, o_top_vga_hsync = %b",
$time, o_top_cam_done, o_top_cam_ready, o_top_vga_red, o_top_vga_green,
o_top_vga_blue, o_top_vga_vsync, o_top_vga_hsync);
end

endmodule

```

[Testbench Code 4.2] top_cam의 Testbench Code

해석: 카메라로부터 픽셀 데이터는 i_top_pix_byte, i_top_pix_vsync, i_top_pix_href를 통해 입력된다. 입력된 데이터는 i_top_pclk와 i_top_clk 클록 신호에 맞춰 동기화되며, Sobel 필터링을 위한 임계값은 w_sobel_thresh로 설정된다. cam_top에서 받은 픽셀 데이터는 vp_top을 통해 처리되고, mem_top으로 전달된다. 처리된 데이터는 vga_top에서 VGA 형식으로 변환되어 o_top_vga_red, o_top_vga_green, o_top_vga_blue 신호로 출력된다. o_top_cam_done은 카메라 데이터 캡처가 완료되었음을 나타내며, o_top_cam_ready는 카메라 준비 완료 상태를 나타낸다. 각 클록(i_top_clk, w_clk25m)은 시스템 동작을 동기화하며, i_top_RST 신호로 초기화된다. 이는 아래의 waveform을 보면 쉽게 이해할 수 있다.



[Figure 4.2] top_cam의 Testbench Code_Waveform

3. top_cam의 testbench code

```
'timescale 1ns / 1ps

module VGA_mem_top_tb;

reg clk, rst;
wire [3:0] VGA_R, VGA_G, VGA_B;
wire hsyncb, vsyncb, vga_ready;

VGA_mem_top uut (
    .clk(clk),
    .rst(rst),
    .VGA_R(VGA_R),
    .VGA_G(VGA_G),
    .VGA_B(VGA_B),
    .hsyncb(hsyncb),
    .vsyncb(vsyncb),
    .vga_ready(vga_ready) );

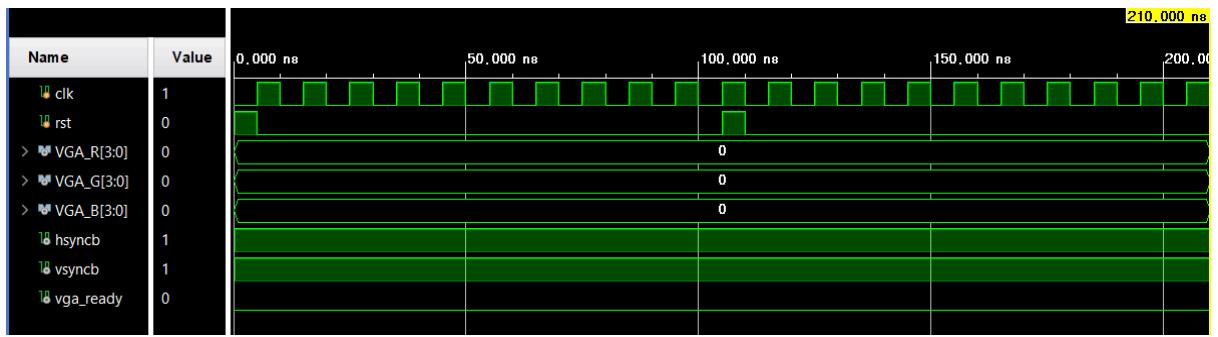
always begin
    clk = 0;
    #5 clk = 1;
    #5;
end

initial begin
    rst = 1;
    #5; rst = 0;
    #100; rst = 1;
    #5; rst = 0;
    #100; $stop;
end

endmodule
```

[Testbench Code 4.3] VGA-mem_top의 Testbench Code

해석: 시뮬레이션이 시작되면 **rst** 신호에 의해 시스템이 초기화된다. **rst**가 1일 때 모듈의 동작은 초기화되고, 5ns 후에 **rst**가 0으로 설정되면 정상 동작을 시작한다. 5초 후(250,000,000 클록 주기 후) **vga_ready** 신호가 활성화되며, 이 시점에서 OV7670 Camera 데이터를 VGA 화면에 출력을 위한 준비가 완료된다. **VGA_R**, **VGA_G**, **VGA_B**는 색상 데이터를 나타내며, **hsyncb**와 **vsyncb**는 화면에 출력되는 데이터의 수평 및 수직 위치를 제어한다. 결국 이 웨이브폼에서는 **rst** 신호가 1로 설정되고, 5ns 후에 0으로 설정되는 리셋 동작을 통해 VGA 출력이 준비되며, 이후 5초 후에 **vga_ready**가 활성화되어 VGA 출력을 위한 준비가 완료되는 과정을 확인할 수 있다.



[Figure 4.3] VGA-mem_top의 Testbench Code_Waveform

4. top의 testbench code

```
'timescale 1ns / 1ps

module tb_top;

reg i_top_clk;
reg i_top_rst;
reg i_top_cam_start;
reg i_top_inc_sobel_thresh;
reg i_top_dec_sobel_thresh;
reg i_top_pclk;
reg [7:0] i_top_pix_byte;
reg i_top_pix_vsync;
reg i_top_pix_href;

wire o_top_cam_done;
wire o_top_cam_ready;
wire o_top_reset;
wire o_top_pwdn;
wire o_top_xclk;
wire o_top_siiod;
wire o_top_sioc;
wire [3:0] o_top_vga_red;
wire [3:0] o_top_vga_green;
wire [3:0] o_top_vga_blue;
wire o_top_vga_vsync;
wire o_top_vga_hsync;
wire [6:0] SEG;
wire [7:0] AN;
wire [15:0] LED;
wire TMP_SCL;
wire TMP_SDA;
wire audio_output;

top uut (
    .i_top_clk(i_top_clk),
    .i_top_rst(i_top_rst),
    .i_top_cam_start(i_top_cam_start),
    .o_top_cam_done(o_top_cam_done),
    .o_top_cam_ready(o_top_cam_ready),
    .i_top_inc_sobel_thresh(i_top_inc_sobel_thresh),
    .i_top_dec_sobel_thresh(i_top_dec_sobel_thresh),
    .i_top_pclk(i_top_pclk),
```

```

.i_top_pix_byte(i_top_pix_byte),
.i_top_pix_vsync(i_top_pix_vsync),
.i_top_pix_href(i_top_pix_href),
.o_top_reset(o_top_reset),
.o_top_pwdn(o_top_pwdn),
.o_top_xclk(o_top_xclk),
.o_top_sioc(o_top_sioc),
.o_top_sioc(o_top_sioc),
.o_top_vga_red(o_top_vga_red),
.o_top_vga_green(o_top_vga_green),
.o_top_vga_blue(o_top_vga_blue),
.o_top_vga_vsync(o_top_vga_vsync),
.o_top_vga_hsync(o_top_vga_hsync),
.SEG(SEG),
.AN(AN),
.LED(LED),
.TMP_SCL(TMP_SCL),
.TMP_SDA(TMP_SDA),
.audio_output(audio_output) );

initial begin
    i_top_clk = 0;
    forever #5 i_top_clk = ~i_top_clk; end

initial begin
    i_top_rst = 1;
    i_top_cam_start = 0;
    i_top_inc_sobel_thresh = 0;
    i_top_dec_sobel_thresh = 0;
    i_top_pclk = 0;
    i_top_pix_byte = 0;
    i_top_pix_vsync = 0;
    i_top_pix_href = 0;

#20 i_top_rst = 0;
#50 i_top_cam_start = 1;
#50 i_top_cam_start = 0;
#100 i_top_inc_sobel_thresh = 1;
#10 i_top_inc_sobel_thresh = 0;
#100 i_top_dec_sobel_thresh = 1;
#10 i_top_dec_sobel_thresh = 0;
#200 i_top_pix_byte = 8'hFF;
i_top_pix_vsync = 1; i_top_pix_href = 1;
#10 i_top_pix_vsync = 0; i_top_pix_href = 0;
#500 $finish;mend

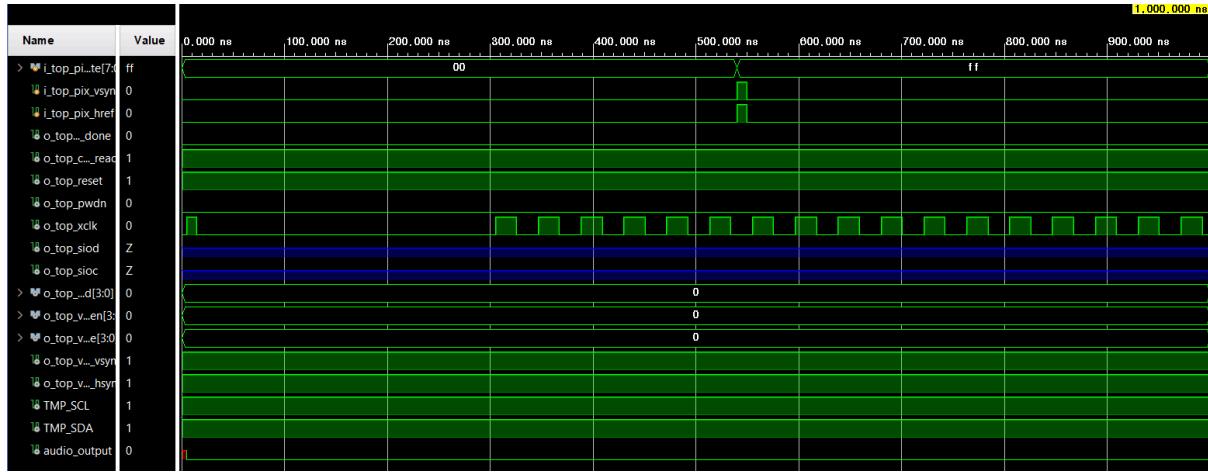
initial begin
    $monitor("Time=%0t | Reset=%b | Cam_Start=%b | Cam_Done=%b | VGA_Red=%h |
VGA_Green=%h | VGA_Blue=%h | LED=%h",
            $time, i_top_rst, i_top_cam_start, o_top_cam_done, o_top_vga_red, o_top_vga_green,
            o_top_vga_blue, LED);
    end

endmodule

```

[Testbench Code 4.4] top의 Testbench Code

해석: top 모듈의 주요 기능을 시뮬레이션하며 클럭과 리셋 신호를 통해 모듈의 초기화와 동작을 확인한다. 리셋 해제 후 카메라 시작 신호에 따라 o_top_cam_done과 VGA 출력 신호가 활성화된다. Sobel 임계값 조정 신호는 VGA 출력 데이터에 영향을 미치며, 픽셀 데이터 입력 시 VGA 출력 반응을 확인할 수 있다. 온도 데이터는 I2C 통신을 통해 업데이트되어 LED와 7세그먼트 디스플레이에 출력된다. 특정 온도 조건에서 알람 신호가 활성화되어 스피커를 통해 경고음을 발생시킨다. 전체적으로 입력 신호에 따른 데이터 흐름과 출력 변화를 검증하여 모듈의 동작을 평가할 수 있다. 아래의 waveform을 보면 데이터의 흐름을 보다 더 이해하기 쉽다. 이 testbench 코드와 그 결과를 통해 프로젝트의 기능을 잘 실행됨을 알 수 있다.

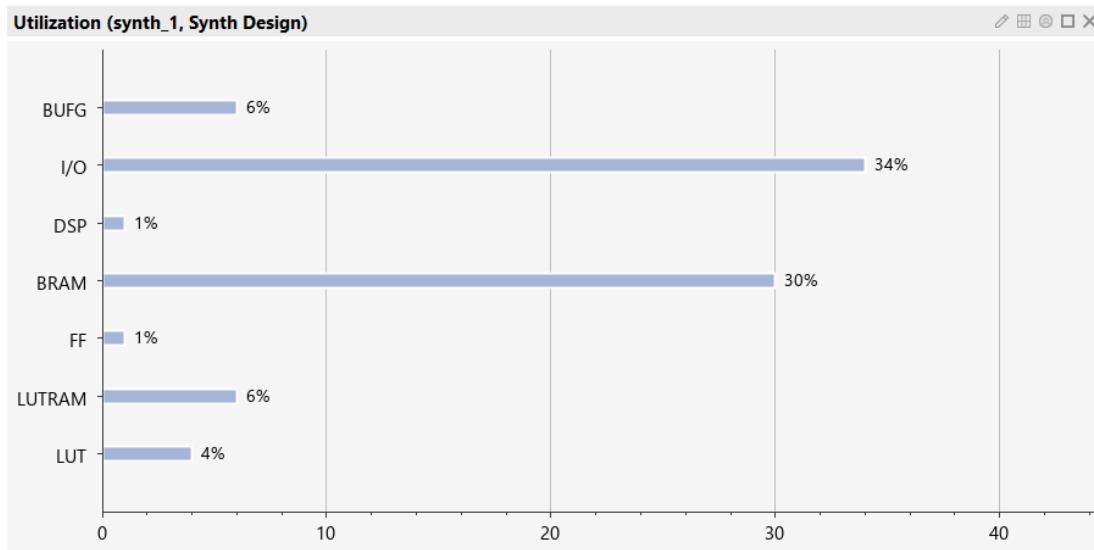


[Figure 4.4] top의 Testbench Code_Waveform

5. Implementation (구현)

5-a. Synthesis Utilization & Result

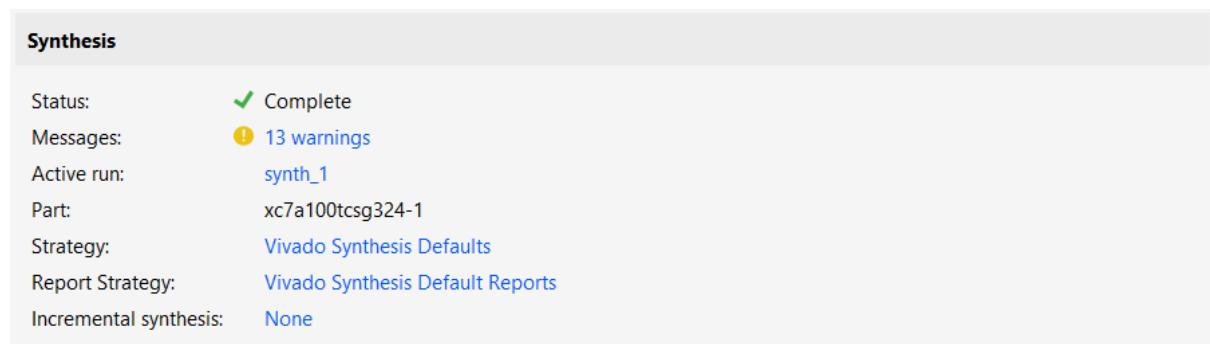
1. LUT 사용률: 전체 LUT의 **4.52%**만 사용되었으며, 그중 논리용 LUT는 **2.61%**, 메모리용 LUT는 **6.36%**를 차지한다.
2. 레지스터 사용률: 레지스터는 **0.68%**만 사용되어 여유가 많다.
3. F7/F8 Muxes 사용률: 각각 0.25% 및 **0.20%**만 사용되었다.
4. Block RAM 사용률: RAMB360이 대부분 사용되었고 전체 **85.56%** 사용되었다.
5. DSP 사용률: 0.83%로, DSP 자원은 대부분 남아 있다.
6. IO 핀 사용률: 33.81%로, 입출력 핀이 비교적 많이 사용되었지만 여유가 있다.
7. 클럭 자원 사용률: BUFG 28.13%, MMCM 33.33%로 클럭이 적절히 분배되었다.



[Figure 5.1] topmodule ☈ Synthesis_Utilization Summary



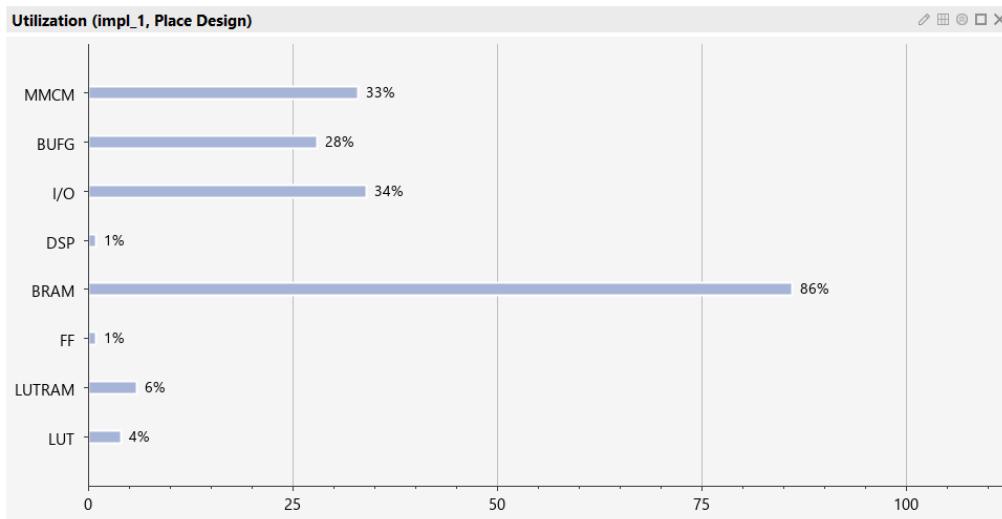
[Figure 5.2] topmodule ☈ Synthesis_Clock Networks



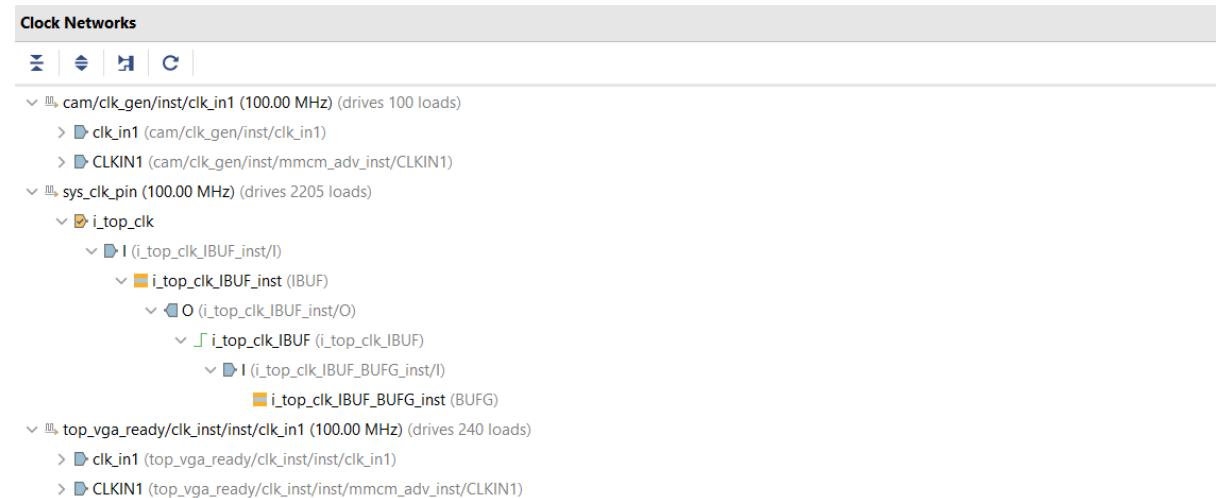
[Figure 5.3] topmodule ☈ Synthesis Summary

5-b. Implementation Utilization

1. LUT 사용률: 논리용 LUT와 메모리용 LUT를 합쳐 4.44% 사용되었다.
2. 레지스터 사용률: 전체 레지스터 중 0.69% 사용되었다.
3. Block RAM 사용률: RAMB36이 대부분 사용되었고 전체 85.56% 사용되었다.
4. DSP 사용률: 0.83%로, DSP 자원은 대부분 남아 있다.
5. IO 핀 사용률: 33.81%로, 약 1/3의 핀이 사용되었다.
6. 클럭 자원 사용률: BUFG 28.13%, MMCM 33.33%로 클럭이 적절히 분배되었다.



[Figure 5.4] topmodule의 Implementation_Utilization Summary



[Figure 5.5] topmodule의 Implementation_Clock Networks

Implementation		Summary Route Status
Status:	✓ Complete	
Messages:	⚠ 92 warnings	
Active run:	impl_1	
Part:	xc7a100tcs324-1	
Strategy:	Vivado Implementation Defaults	
Report Strategy:	Vivado Implementation Default Reports	
Incremental implementation:	None	

[Figure 5.6] topmodule의 Implementation Summary

5-c. Area:



[Figure 5.7] topmodule의 Device

1. Slice LUTs: 4.52% 사용 (논리용 2.61%, 메모리용 6.36%)
2. Slice Registers: 0.68% 사용
3. Block RAM: 85.56% 사용 (매우 높음)
4. DSP 블록: 0.83% 사용
5. IO 핀: 33.81% 사용

5-d. Timing Analysis:

1. Setup 타이밍 (Setup Time)
 - a. Worst Negative Slack (WNS): 0.709 ns
 - i. WNS 값이 양수이므로 타이밍 제약을 모두 만족하고 있다. (*WNS는 타이밍 경로 중 가장 여유가 적은 값을 의미한다.)
 - b. Total Negative Slack (TNS): 0.000 ns

모든 타이밍 경로에서 타이밍 제약을 준수하고 있어 Slack 위반이 없다.
 - c. Failing Endpoints: 0

타이밍 위반이 발생한 엔드포인트는 없다.

2. Hold 타이밍 (Hold Time)

- a. Worst Hold Slack (WHS): 0.055 ns
WHS 값이 양수이며, 이는 Hold 타이밍 제약도 만족하고 있음을 의미한다.
- b. Total Hold Slack (THS): 0.000 ns
Hold 타이밍 위반이 발생하지 않았다.
- c. Failing Endpoints: 0
모든 경로에서 Hold 타이밍이 만족된다.

3. Pulse Width (펄스 폭)

- a. Worst Pulse Width Slack (WPWS): 3.000 ns
WPWS 값이 양수로, 펄스 폭 타이밍 제약도 준수되고 있다.
- b. Failing Endpoints: 0
펄스 폭과 관련된 경로에서 타이밍 위반은 없다.

4. 타이밍 제약 만족: 모든 타이밍 제약(Setup, Hold, Pulse Width)이 충족되었다.

5. Slack 여유:

- a. Setup Slack: 0.709 ns
- b. Hold Slack: 0.055 ns
- c. Pulse Width Slack: 3.000 ns

6. 시스템의 타이밍 경로: 충분한 여유(Slack)를 가지고 동작한다.

이 결과는 설계가 타이밍 측면에서 안정적임을 의미한다. 추가적인 최적화를 통해 클럭 주파수를 더 높이는 것도 가능할 수 있다.

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 0.709 ns	Worst Hold Slack (WHS): 0.055 ns	Worst Pulse Width Slack (WPWS): 3.000 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 13622	Total Number of Endpoints: 13622	Total Number of Endpoints: 2219

All user specified timing constraints are met.

[Figure 5.8] topmodule의 Implementation_Timing Summary

Timing	Setup Hold Pulse Width
Worst Negative Slack (WNS):	0.709 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	13622
Implemented Timing Report	

[Figure 5.9] topmodule의 Implementation_Timing Report

6. Conclusion (결론)

6-a. Conclusion

이 프로젝트는 카메라 입력, 비디오 처리, VGA 출력, 온도 센서 데이터 처리 및 BGM 생성이라는 다양한 하드웨어 및 소프트웨어 모듈을 통합한 시스템을 설계하고 구현하는 데 성공하였다. 프로젝트의 주요 결과와 성과를 아래와 같이 정리할 수 있다.

1. 주요 성과

- a. 하드웨어-소프트웨어 통합
 - i. 카메라 입력과 VGA 디스플레이의 동작을 안정적으로 통합하여 실시간으로 데이터를 처리하고 출력하는 시스템을 구현하였다.
 - ii. 온도 센서 데이터를 처리하여 LED와 7-Segment 디스플레이에 시각적으로 출력하며, 특정 조건에서 알람을 활성화하였다.
- b. 효율적인 데이터 경로 설계
 - i. 데이터의 실시간 처리를 위한 효율적인 데이터 경로(Datapath)를 설계하였으며, FIFO 및 BRAM을 활용하여 데이터 손실 없이 처리 속도를 최적화하였다.
 - ii. Sobel 필터와 같은 경계 검출 알고리즘을 적용하여 데이터 처리 기능을 강화하였다.
- c. 안정적인 동작
 - i. 상태 머신(FSM)을 기반으로 컨트롤러를 설계하여 모듈 간 동기화와 안정성을 확보하였다.
 - ii. FIFO를 통해 서로 다른 클럭 도메인 간 데이터 전송을 안정적으로 수행하였다.
- d. 확장 가능성 확보
 - i. 독립적인 모듈 기반 설계를 통해 시스템의 유지보수성과 확장성을 보장하였다.
 - ii. 새로운 데이터 처리 알고리즘, 입력 장치 또는 출력 장치를 추가할 수 있는 유연한 구조를 구축하였다.

2. 프로젝트 의의

- a. 다분야 통합 학습: 하드웨어 설계(VHDL/Verilog), 소프트웨어 제어, 신호 처리, 그리고 디스플레이 시스템에 대한 종합적인 이해를 통해 전반적인 시스템 설계 능력을 향상시켰다.
- b. 실용적인 문제 해결 능력 배양: 사용자 입력, 실시간 데이터 처리, 그리고 출력 간의 상호작용 문제를 해결하며 실용적인 엔지니어링 문제를 다뤘다.
- c. 연구 및 개발 기초 마련: 본 프로젝트에서 설계된 구조는 향후 더욱 복잡한 데이터 처리 및 멀티미디어 응용 개발에 기초가 될 수 있었다.

3. 한계점 및 개선 가능성

- a. 하드웨어 자원 사용 최적화: FPGA 내부 자원의 활용이 최적화되지 않아, 복잡한 알고리즘 추가 시 자원 부족 가능성이 있다.
- b. 처리 속도의 제한: VGA 출력과 데이터 처리 간 동기화는 성공했으나, 고해상도 데이터

- c. 처리 시 성능 병목이 발생할 가능성이 있다.
- c. 사용자 인터페이스: 현재의 사용자 인터페이스는 기본적인 입력만 처리하며, 직관적이고 확장된 인터페이스 설계가 필요하다.

6-b. Future Plan & Possibility

이 프로젝트는 현재 설계를 기반으로 더욱 발전된 기능과 성능을 추가하여 다양한 응용으로 확장할 가능성을 가지고 있다. 향후 계획과 확장 가능성을 아래와 같이 제시한다.

- 1. 향후 계획**
 - a. 고해상도 데이터 처리
 - i. 고해상도 카메라를 지원하기 위해 BRAM 용량을 확장하고, 데이터 압축 기법을 추가할 수 있다.
 - ii. VGA 출력 해상도를 1080p 이상으로 개선하여 더욱 선명한 디스플레이 출력 제공할 수 있다.
 - b. 고급 비디오 처리 알고리즘 적용
 - i. Sobel 필터 외에도 Gaussian Blur, Laplacian Edge Detection 등의 고급 필터를 추가하여 영상 처리 기능을 확장.
 - ii. 머신러닝 기반의 객체 탐지 및 분류 기능 추가하여 자율 주행 기능(ADAS) 프로그램에 데이터를 제공할 수 있다.
 - c. 실시간 통신 기능
 - i. Ethernet 또는 Wi-Fi 모듈을 추가하여 실시간 스트리밍 및 데이터 전송 기능 구현할 수 있다.
 - ii. IoT 플랫폼과 연동하여 센서 데이터 및 처리 결과를 원격으로 모니터링할 수 있다.
 - d. 다양한 입력 및 출력 지원
 - i. 추가적인 입력 장치(예: 키보드, 마우스)와 출력 장치(예: HDMI 디스플레이) 지원할 수 있다.
 - ii. 멀티채널 오디오 출력 및 음성 인식 기능 구현할 수 있다.
 - iii. LCD 디스플레이를 연결하여 조작하는 보드와 로봇과 카메라 및 여러 센서를 연결한 보드의 통신을 통해 로봇 게임을 개발할 수 있다.
- 2. 확장 가능성**
 - a. 멀티미디어 응용:
 - i. 실시간 비디오 스트리밍, 게임 엔진, 또는 인터랙티브 디스플레이 시스템 개발할 수 있다.
 - ii. 3D 렌더링 및 가상현실(VR) 응용할 수 있을 것이다.
 - b. 산업 응용:
 - i. 산업용 머신 비전 시스템에 적용하여 실시간 품질 관리 및 결함 검출할 수 있다.
 - ii. 자동화 공정에서 카메라 데이터를 기반으로 한 로봇 제어 시스템을 제작할 수 있다.
 - c. 교육 및 연구:
 - i. FPGA 기반 실습 키트로 활용하여 학생들에게 하드웨어 설계 및 영상 처리 기법 교육할 수 있다.
 - ii. 연구 목적으로 고급 신호 처리 및 시스템 설계의 기초로 사용할 수 있을 것이다.
 - d. 확장된 센서 네트워크:
 - i. 온도 센서 외에 습도, 압력, 가스 센서를 추가하여 다중 센서 네트워크로 확장할 수 있다.
 - ii. 센서 데이터를 기반으로 한 지능형 제어 시스템 개발할 수 있다.
- 3. 기대효과**
 - a. 응용의 다양성: 본 설계는 다양한 산업과 연구 분야에서 사용될 수 있는 다목적 플랫폼을 제공한다.
 - b. 미래 기술과의 융합: IoT, AI, 고해상도 디스플레이 기술과 통합하여 첨단 시스템 개발의

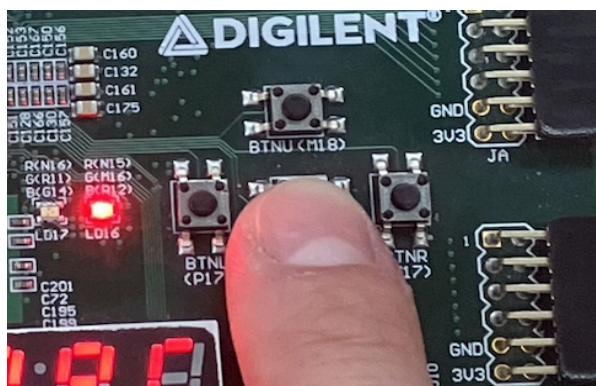
- 기반이 될 수 있다.
- c. 지속적인 발전 가능성: 모듈화된 설계 구조는 지속적인 업그레이드와 새로운 기능 추가를 용이하게 한다.

본 프로젝트는 현재의 설계로도 충분히 실용적인 응용이 가능하지만, 향후 계획과 확장 가능성을 통해 더 나은 성능과 기능을 가진 시스템으로 발전할 수 있다. 이러한 확장은 기술적 도전 과제와 함께 더 큰 가능성을 제공할 것이다.

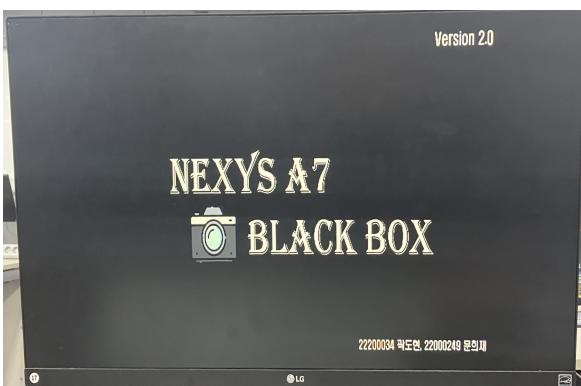
7. Appendix (참고 문헌)

7-a. Usage Description & Demo

- 초기화: BTNC버튼을 클릭하여 전원을 켜 블랙박스 초기화면이 또는 것을 확인한다.

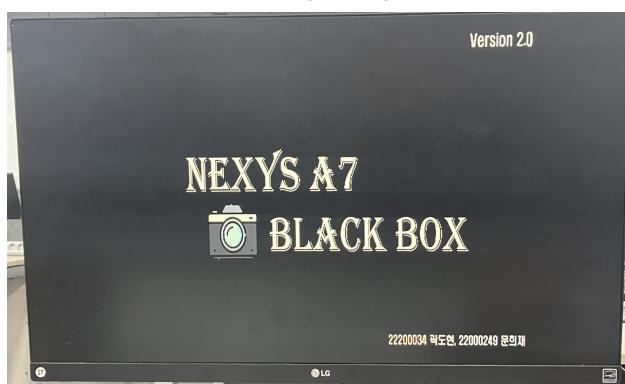


[Figure 7.1] press_BTNC (initial)

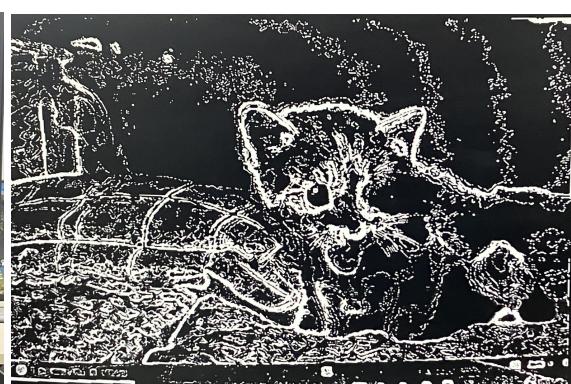


[Figure 7.2] vga_Ready (initial)

- 준비: 일정 시간뒤(3~5초) 초기화면에서 카메라 송출 준비상태로 넘어가는 것을 확인한다.



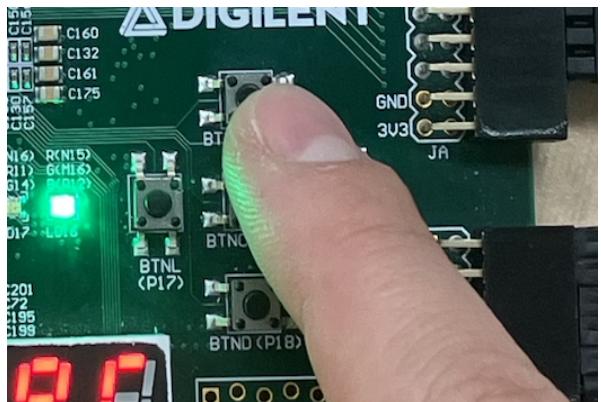
[Figure 7.2] vga_Ready (initial)



[Figure 7.3] vga_Ready2Cam (initial)

- 프로그램 시작: BTNU버튼을 클릭하여 카메라로 인식해 Lane Detection된 화면을 VGA에 출력한다.

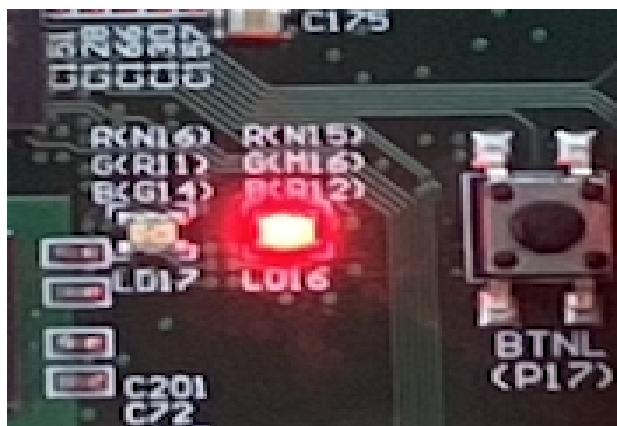
- 카메라 송출 시 RGB LED 색상이 빨간색에서 초록색으로 변하는 것을 확인한다.



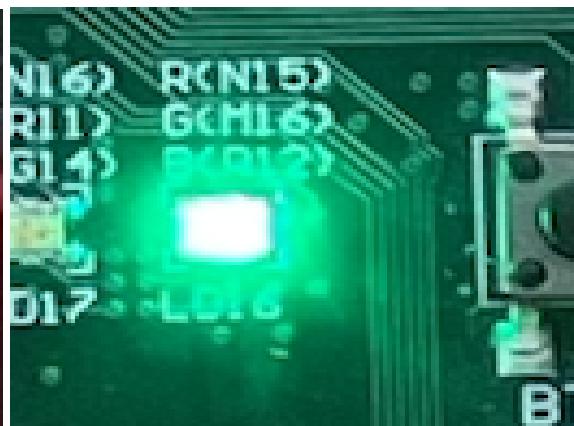
[Figure 7.4] press_BTNU (ready)



[Figure 7.5] vga_LaneDetection (initial)

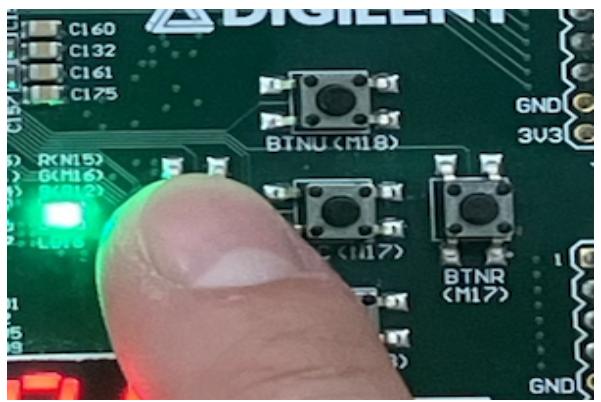


[Figure 7.6] vga_RGB_Led (initial)

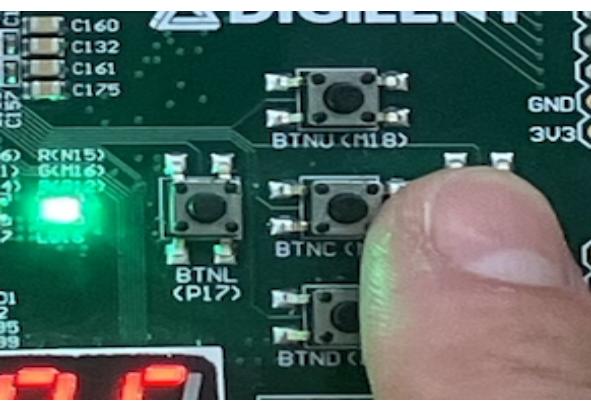


[Figure 7.7] vga_RGB_Led (ready)

4. 환경에 따라 BTNL/BTNR를 눌러 임계값(Threshold)을 조정한다. 이때, 각 버튼을 눌렀을 때 스피커를 통해 효과음이 출력되는 것을 확인한다.
 - a. BTNL : 임계값 감소, 흰색 선이 증가되는 것을 확인한다.
 - b. BTNR : 임계값 증가, 흰색 선이 감소되는 것을 확인한다.

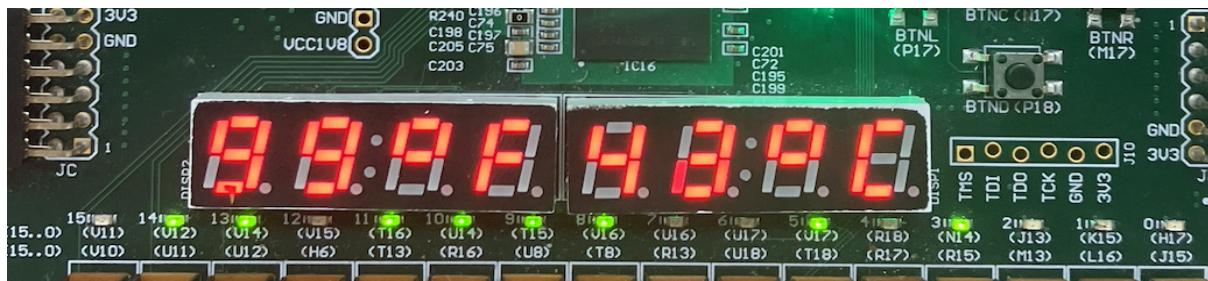


[Figure 7.8] press_BTNU (threshold dec)



[Figure 7.9] press_BTNR (threshold inc)

5. Temperature Sensor에 온도변화를 주어 실시간으로 변화되는 섭씨온도, 화씨온도가 7 Segment와 LED에 출력되는 것을 확인한다. 이때 섭씨온도 40도 이상일 때 스피커를 통해 경고음이 출력되는 것을 확인한다.



[Figure 7.10] temp_7seg_led

7-b. Individual Role

1. 곽도현

- a. 시스템 아키텍처 설계
 - i. 프로젝트의 전반적인 구조를 구상하고, 모듈 간 데이터 흐름과 인터페이스 설계
 - ii. Sobel 필터 적용 및 영상 처리 파이프라인 설계
- b. 카메라 및 영상 처리 구현
 - i. top_cam 모듈 설계 및 코딩
 - ii. 카메라 초기화(cam_init) 및 픽셀 데이터 캡처 모듈(cam_capture) 구현
 - iii. vp_top 모듈 설계와 Sobel 필터(conv) 적용
- c. 디버깅 및 테스트
 - i. 카메라 데이터 처리 결과를 확인하고 디버깅
 - ii. Sobel 필터의 임계값 조정 기능과 사용자 입력 반응 테스트
- d. 보고서 작성
 - i. 영상 처리와 관련된 부분의 기술 문서 작성

2. 문희재

- a. BGM 및 온도 센서 시스템 설계
 - i. top_bgm과 top_temp 모듈 구현
 - ii. BGM 상태 제어와 알람 기능 설계
 - iii. 온도 데이터를 I2C로 읽어 7-Segment 디스플레이와 LED로 출력
- b. VGA 출력 구현
 - i. vga_top 및 VGA_mem_top 모듈 설계와 코딩
 - ii. BRAM에서 픽셀 데이터를 읽어 VGA 디스플레이로 출력
 - iii. VGA 타이밍 신호 생성(vga_driver)
- c. 사용자 인터페이스 설계
 - i. top_user_control 모듈 설계 및 구현
 - ii. 버튼 입력에 따른 Sobel 필터 임계값 조정과 BGM 변경 기능 추가
- d. 프로젝트 관리
 - i. 일정 관리와 모듈 통합 테스트 조율
 - ii. 최종 보고서와 발표 자료 작성

7-b. Schedule

1. 13주차: 제안서 작성 및 프로젝트 구상

- a. 초기 제안서 작성: 기술사 관리 시스템 개발을 목표로 설계
- b. 기능 분석 및 구현 가능성 검토
- c. 각자 역할 분담 및 일정 초안 작성

2. **14주차:** 프로젝트 목표 변경 및 새로운 설계 착수
 - a. 컴퓨터 비전 과목 실습(메모리에 COE 파일 입력 및 출력)
 - b. 이를 기반으로 컴퓨터 비전과 VGA 디스플레이를 결합한 새로운 프로젝트 구상
 - c. 카메라와 VGA 디스플레이 연결 구조 설계

3. **15주차:** 모듈 구현 및 통합
 - a. 멤버 1: 카메라 데이터 처리 모듈(`top_cam, vp_top`) 구현 시작
 - b. 멤버 2: BGM 및 온도 센서 모듈(`top_bgm, top_temp`) 설계 및 코딩
 - c. VGA 디스플레이 초기 신호 출력 테스트
 - d. 사용자 입력 처리 기능(`top_user_control`) 설계

4. **16-17주차:** 테스트, 디버깅 및 최종 통합
 - a. 모든 모듈 통합 및 전체 시스템 테스트
 - b. Sobel 필터 결과와 VGA 출력 확인
 - c. 사용자 입력 반응 테스트 및 최적화
 - d. 프로젝트 발표 자료 및 최종 보고서 작성

5. 결과: 4주간의 협업을 통해 시스템 설계, 모듈 개발, 디버깅, 그리고 통합 테스트를 성공적으로 완료하였다. 프로젝트 목표는 실시간 영상 처리와 VGA 디스플레이 출력 기능의 안정성을 입증하며 달성되었다.

7-d. References

1. `top`: 직접 작성

2. `top_bgm`: github 소스에서 필요한 부분만 사용(`bgm` 관련된 모듈만 사용하였고, 약간의 코드만 수정하였습니다. 소스는 Pong Game이라는 github 소스에서 사용하였습니다.)

출처:

https://github.com/xu842251462/EC551_FinalProject/tree/ffd5ea84502b3241a851d53a1589fbf6ec750456

3. `top_temp`: digilent 온도 센서에 대한 기본 코드를 사용하여 약간의 코드 수정, 그리고 알림음을 직접 작성하여 두 코드를 병합하여 사용하였습니다. (즉, 온도 센서에 대한 데이터 처리는 github에 있는 오픈 소스를 사용하였고 그에 대한 일정 온도가 올라갔을 때 소리를 내는 코드만 작성하였습니다.)

출처:

https://github.com/FPGADude/Digital-Design/tree/main/FPGA%20Projects/NexysA7_Temp_Sensor_12C

<https://github.com/RVZWN/TempSensorWithFPGAs>

4. `top_cam`: 카메라를 바로 vga 화면에 출력하는 오픈 소스 코드(OV7670 camera 관련된 verilog 코드 중 하나를 사용)를 사용, 컴퓨터비전 내용과 인터넷 서칭하여 `grayscale`로 변환하는 코드만 추가하였습니다.

"`vp_top` 모듈의 `grayscale` 서브 모듈에서 RGB 데이터를 8비트 그레이스케일로 변환한다.

변환 공식: $\text{Gray} = 0.3*\text{R} + 0.59*\text{G} + 0.11*\text{B}$

"

이를 활용하여 각 색상별로 4비트 시프트하여 소수점을 추가하고 비율별로 근사치로 변환하였습니다.

이와 관련된 코드는 다음과 같습니다.

```

assign R = (i_data[11:8] << 4);
assign G = (i_data[7:4] << 4);
assign B = (i_data[3:0] << 4);
o_gray_data <= (R >> 2) + (R >> 5) +
(G >> 1) + (G >> 4) +
(B >> 4) + (B >> 5);

```

camera에 대한 모듈 작성은 직접 모든 것을 작성한 것이 아니라 다른 여러 코드를 참고하면서 작성하였다는 점 말씀 드립니다.

출처: <https://nottoday2020.tistory.com/182>

https://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf

<https://github.com/bornhorst/Nexys-4-DDR-using-OV7670-Camera>

<https://github.com/ykgpgrr/Lane-Detection-on-FPGA-HW-part-with-VIVADO/tree/master>

<https://github.com/LBL-ICS/A-Scalable-Image-Video-Processing-Platform-with-Open-Source-Design-and-Verification-Environment/tree/master>

5. vga_mem_top: 수업 시간에 활용한 소스 코드를 변형하여 일정 시간 뒤에 카메라 화면으로 넘어갈 수 있게끔 코드 수정하였습니다. 기본 틀은 수업 시간에 배부해주신 코드와 거의 동일합니다.

7-e. Individual Impressions

1. 곽도현

이번 프로젝트는 컴퓨터 비전 기술과 임베디드 시스템을 통합한 흥미로운 경험이었습니다. 특히 VGA 디스플레이와 카메라 데이터를 처리하는 과정을 직접 설계하고 구현하면서 많은 경험을 했던 것 같습니다. 처음 기숙사 관리 시스템을 계획했을 때와 비교하여, 최종적으로 컴퓨터 비전과 VGA를 결합한 프로젝트로 변경하는 과정에서 제가 배운 여러 학문(전산과목과 전자과목)을 통합할 수 있다는 것과 하드웨어와 소프트웨어 디자인을 함께 진행하면서 많은 것들을 배웠습니다.

컴퓨터 비전 수업 때 코드로만 보던 것과는 다르게 카메라 데이터를 처리하는 과정은 생각보다 복잡했습니다. 특히, OV7670 카메라 초기화 과정과 픽셀 데이터를 수집하는 데 있어, SCCB 프로토콜을 사용하는 부분은 예상보다 어려웠습니다. 하지만 데이터 동기화를 위해 FIFO를 사용하고 Sobel 필터를 적용하는 과정에서 이론과 실제 구현을 연결하는 방법을 깊이 이해할 수 있었습니다. 2명이 팀을 이루어 진행하다 보니, 서로의 역할과 책임을 명확히 나누는 것이 매우 중요하다고 느꼈습니다. 특히 각자의 작업 내용을 공유하고 모듈 간 통합 테스트를 조율하는 과정에서 팀워크와 커뮤니케이션 능력이 프로젝트의 성공에 핵심이라는 것을 느꼈습니다. VGA 디스플레이를 직접 제어하고, Sobel 필터와 같은 영상 처리 알고리즘을 FPGA에서 구현하는 경험은 이론을 실무에 적용하는 좋은 기회였습니다.

이러한 경험은 향후 임베디드 시스템 및 컴퓨터 비전 분야에서 더 깊이 있는 프로젝트를 진행하는 데 큰 도움이 될 것이라고 생각합니다. 프로젝트 초기에 제안서 작성과 설계에 너무 많은 시간을 사용하다 보니, 최종 테스트 단계에서 시간이 부족했습니다. 다음 프로젝트에서는 초기에 시간 관리를 더 철저히 하고, 테스트와 디버깅에 충분한 시간을 확보하고 싶습니다. 또한 캡스톤 디자인 주제로 조금 더 넓고 깊은 분야를 다루는 FPGA 설계를 해보고 싶다는 생각이 들었습니다. 그리고 내년 여름 방학 때는 로봇(하드웨어)을 제작하여 자율주행자동차 경진대회 또는 자동차 로봇 조종 시스템을 제작해보고 싶다는 생각이 들었습니다. 이 프로젝트 경험이 현재 스트라드비전에서 현장실습에 많은 도움이 되고 있고, 앞으로도 저의 진로와 공부에 많은 도움이 될 것 같습니다.

2. 문희재

이 프로젝트는 하드웨어와 소프트웨어의 통합 작업에 대한 실질적인 경험을 제공했으며, 다양한

기술을 배우고 적용할 수 있는 좋은 기회였습니다. 특히 BGM 생성과 VGA 디스플레이 제어를 담당하면서 많은 교훈을 얻게 되었습니다. 특히, BGM 생성 모듈을 설계하면서 주파수 분할기와 상타 기반 음악 출력을 구현하는 과정이 매우 흥미로웠습니다. 또한, I2C 프로토콜을 통해 온도 데이터를 읽어오고, 이를 7-Segment 디스플레이와 LED로 출력하는 작업은 임베디드 시스템 설계의 기본적인 기술을 다질 수 있는 기회였습니다.

VGA 디스플레이를 제어하는 작업은 초기에는 복잡하고 어려웠지만, 타이밍 신호와 RGB 데이터의 상관관계를 이해하면서 점차 자신감을 얻었습니다. 특히 BRAM에서 데이터를 읽어와 VGA로 출력하는 모듈을 설계하며, 하드웨어 동작 원리에 대한 깊이 있는 이해를 할 수 있었습니다. 프로젝트 중간에 발생한 주요 문제 중 하나는 사용자 입력 처리에서 디바운싱 처리의 누락으로 인해 예상치 못한 결과가 발생한 것이었습니다. 이를 해결하기 위해 디바운싱 로직을 추가하고 테스트하는 과정을 통해 문제 해결 능력이 크게 향상되었습니다. 또한 카메라 신호의 노이즈를 잡기 위해 저항을 연결하여 노이즈 감소에 기여할 수 있었습니다.

다른 팀원과 역할을 분담하여 프로젝트를 진행하며, 서로의 작업 결과에 따라 전체 시스템이 잘 동작할 수 있다는 점에서 협업의 중요성을 다시 한번 깨달았습니다. 특히 모듈 간 인터페이스를 정의하고 통합 테스트를 진행할 때, 철저한 커뮤니케이션이 필수적이었습니다. 이번 프로젝트를 통해 배운 기술과 경험은 향후 다양한 응용 분야에 적용할 수 있을 것이라 생각합니다. 특히, 임베디드 시스템 설계와 컴퓨터 비전 분야에서 이번 경험은 좋은 밑거름이 될 것입니다.