OV7670 & FPGA

Filter Piano

조 이 름 : 건반 만든 김에(필터도 만들었)조

발 표 일 : 25. 12. 05

INDEX

Introduction

팀원 소개

이서영

Keyboard 구현
Filter 설계

박찬호

SCCB 설계
RED TRACKER 설계
PIANO Controller 설계
Top 모듈 통합

김호준

Python을 이용한 구현
FPGA 연동

이승후

Filter 설계

Introduction

- **프로젝트 개요**

- OV7670 카메라의 레지스터 설정을 위해 SCCB Controller–I2C Master–ROM 구조를 직접 설계하고, FPGA에서 실시간 영상 스트리밍 수행.
- 입력 영상의 좌표 기반 샘플링을 활용하여 Pixel 모드·Keyboard 모드의 피아노 인터페이스를 구현, UART 통신을 통해 PC(python)와 음계 데이터 연동.
- FPGA 내부에서 영상 필터를 설계하여 실시간으로 오버레이 처리하는 하드웨어 기반 이미지 필터링 시스템 구축.

- **개발 환경**

VIVADO HLx Editions

python

OV7670

Basys3 Board

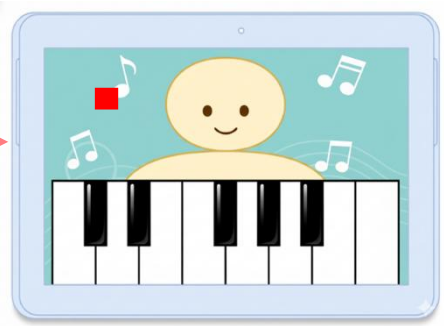
OV7670 to PC

Operation mode

- Pixel mode

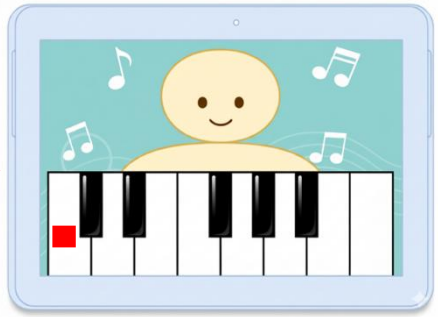
‘A’ 입력

‘255’ 쓰레기 값 전송



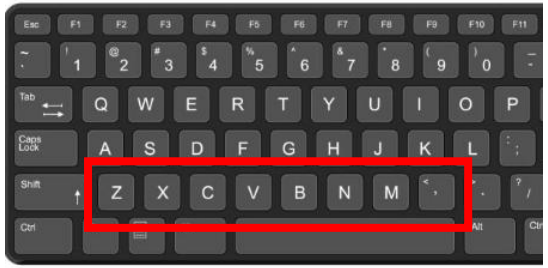
묵음 출력

Sampling Data 전송



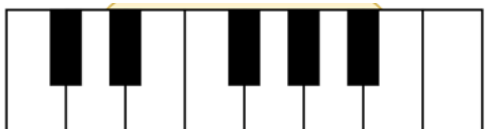
음계 출력

- Keyboard mode



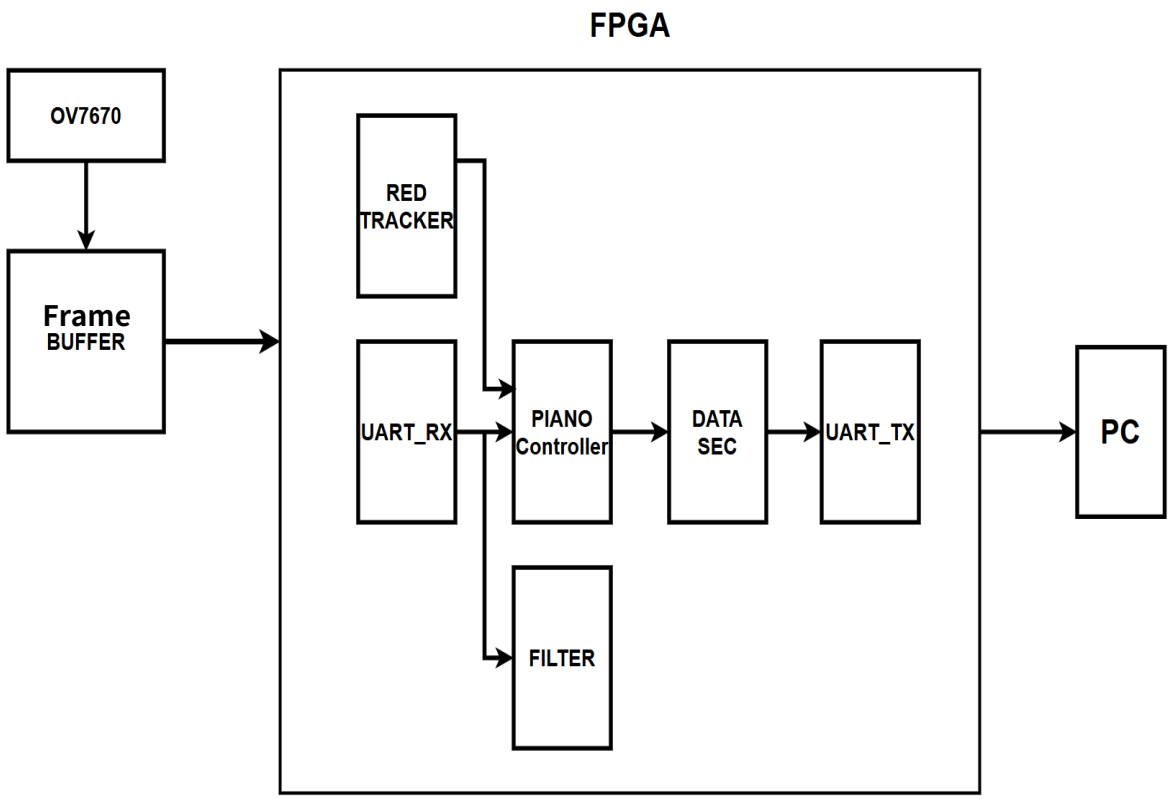
‘z’~ ‘,’ 입력

좌표값 UART Data



음계 출력

Summary



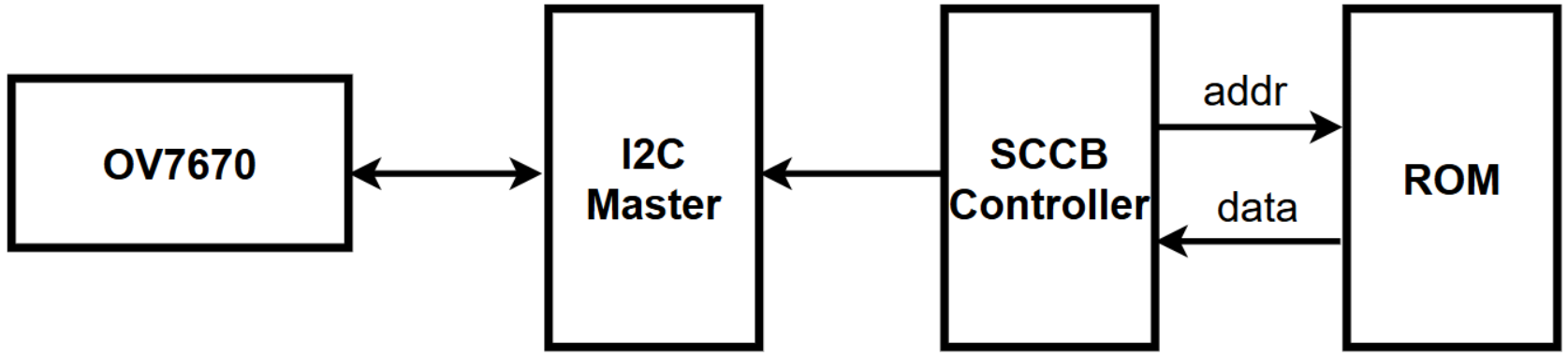
1. Video Input & Configuration
- **OV7670 Camera:** 실시간 RGB 영상 데이터 입력

2. FPGA Image Processing
- **Red Tracker:** Shift Register와 RGB 비교 알고리즘을 통해 적색 영역 좌표 추출
- **Piano Controller:** 추출된 좌표를 분석하여 건반 위치 매핑 및 음계 데이터 생성
- **Visual Filter:** 눈/벚꽃/번개 필터 실시간 오버레이

3. Communication & Output
- **UART Interface:** 'DATA_SEC' 모듈을 통해 중복 입력을 방지하며 PC로 데이터 전송
- **PC Application:** Python 기반 수신 프로그램이 Serial 데이터를 파싱하여 실시간 사운드 재생

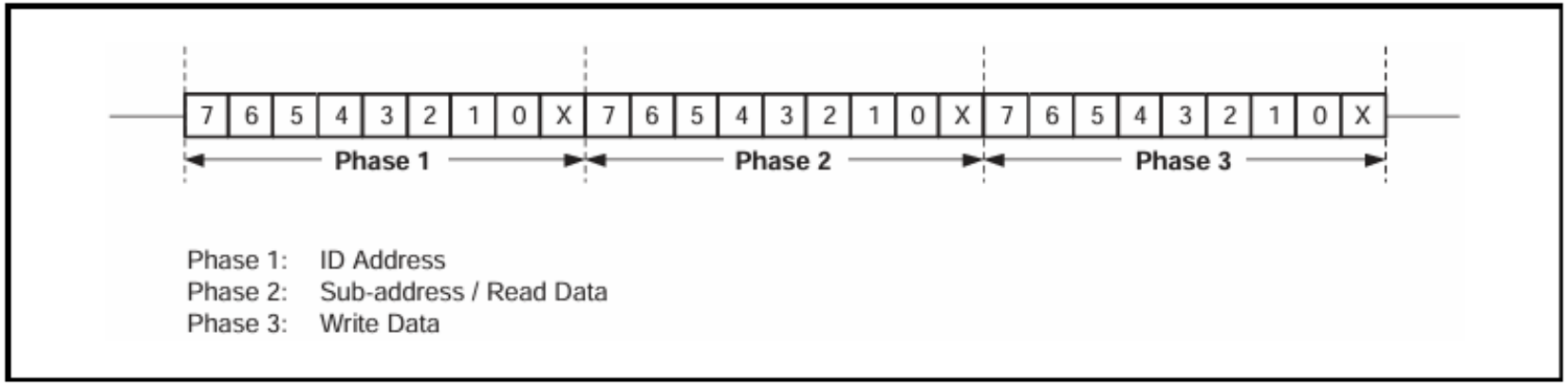
OV7670 Setting

- 카메라 레지스터 초기화 시스템 Block Diagram

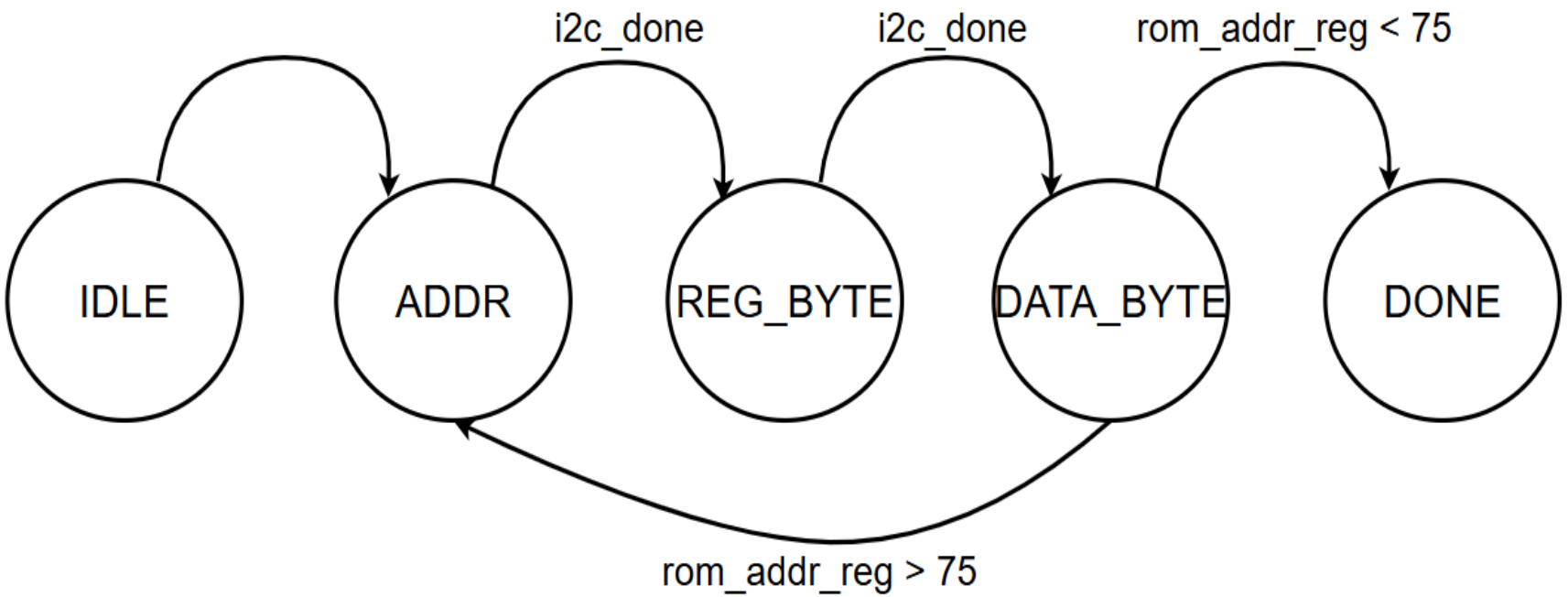


Data : 해상도, 색상 포맷, 프레임 속도 등

Figure 3-4 Transmission Phases

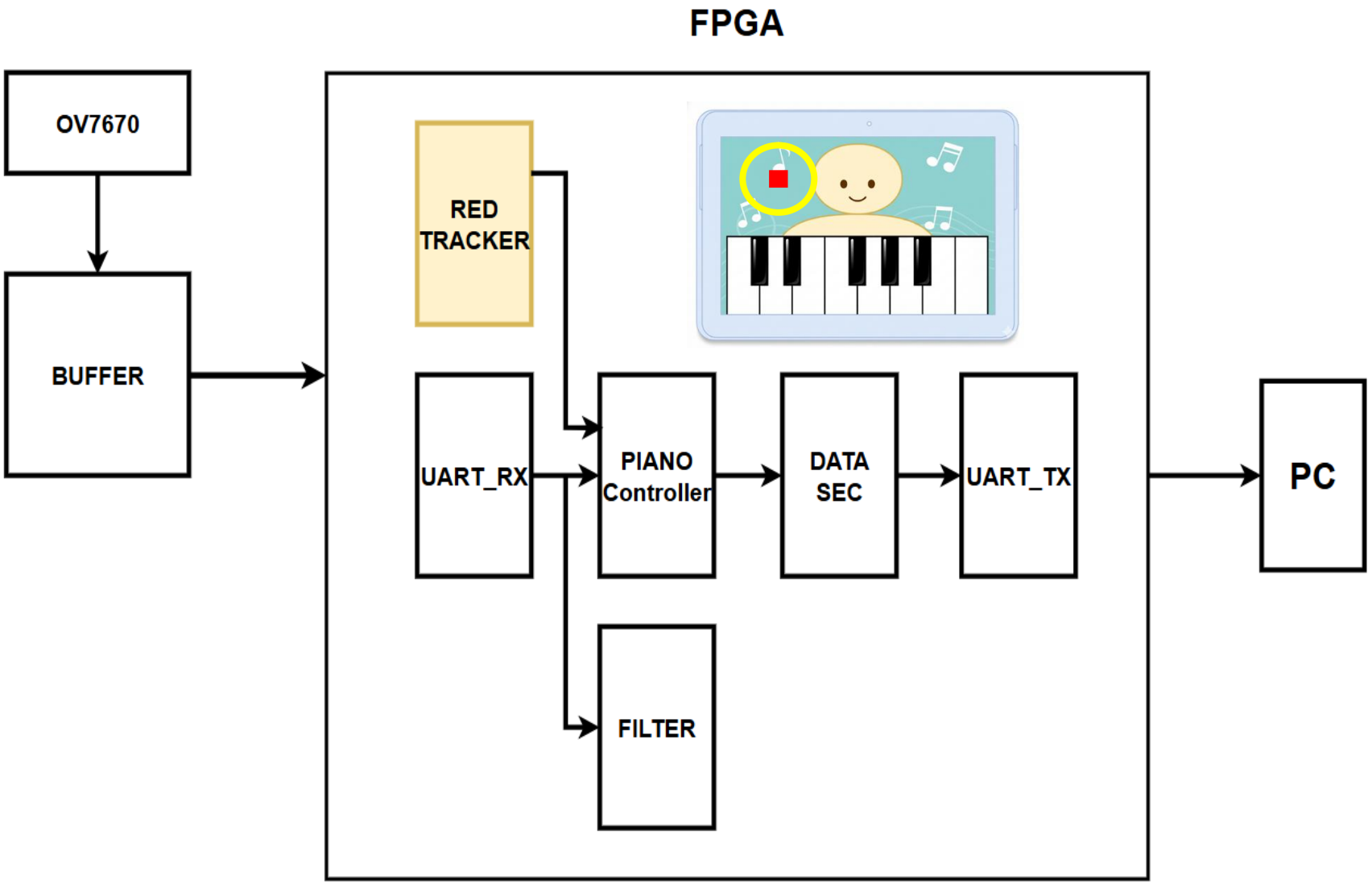


SCCB - FSM



- ADDR : OV7670의 address를 전송.
- REG_BYTE : 센서 내부 레지스터의 address 전송.
- DATA_BYTE : 내부 레지스터에 저장할 데이터를 저장.
- DONE : ROM에 센서에 쓸 레지스터 주소와 데이터가 모두 저장되어 설정이 끝나면 DONE으로.

RED TRACKER


FPGA
OV7670
BUFFER
RED TRACKER
UART_RX
PIANO Controller
DATA SEC
UART_TX
FILTER
PC

RED TRACKER

- **Color Classification**

: Red 성분이 Green,Blue 성분보다 압도적으로 높은지 비교 방식 적용

$$\text{IsRed} = \underbrace{(R > T_{\min})}_{\text{1. Brightness}} \wedge \underbrace{(R > G + M) \wedge (R > B + M)}_{\text{2. Dominance}} \wedge \underbrace{(G < T_{\max}) \wedge (B < T_{\max})}_{\text{3. Purity}}$$

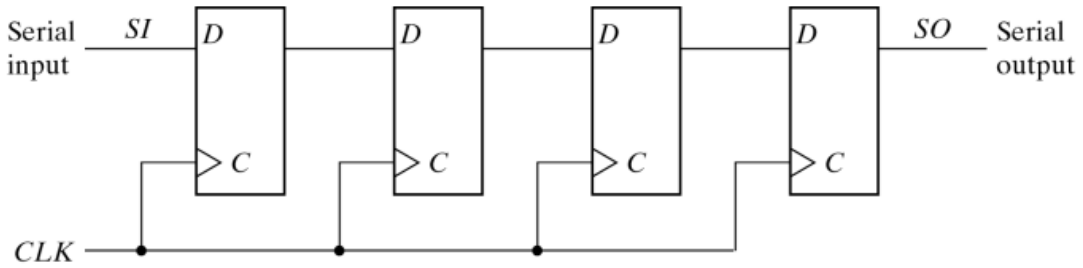
→ 1. Brightness : R값이 일정 수준(Tmin) 이상

→ 2. Dominance : G, B가 Margin 이상 커야 유효한 Red 값으로 판단

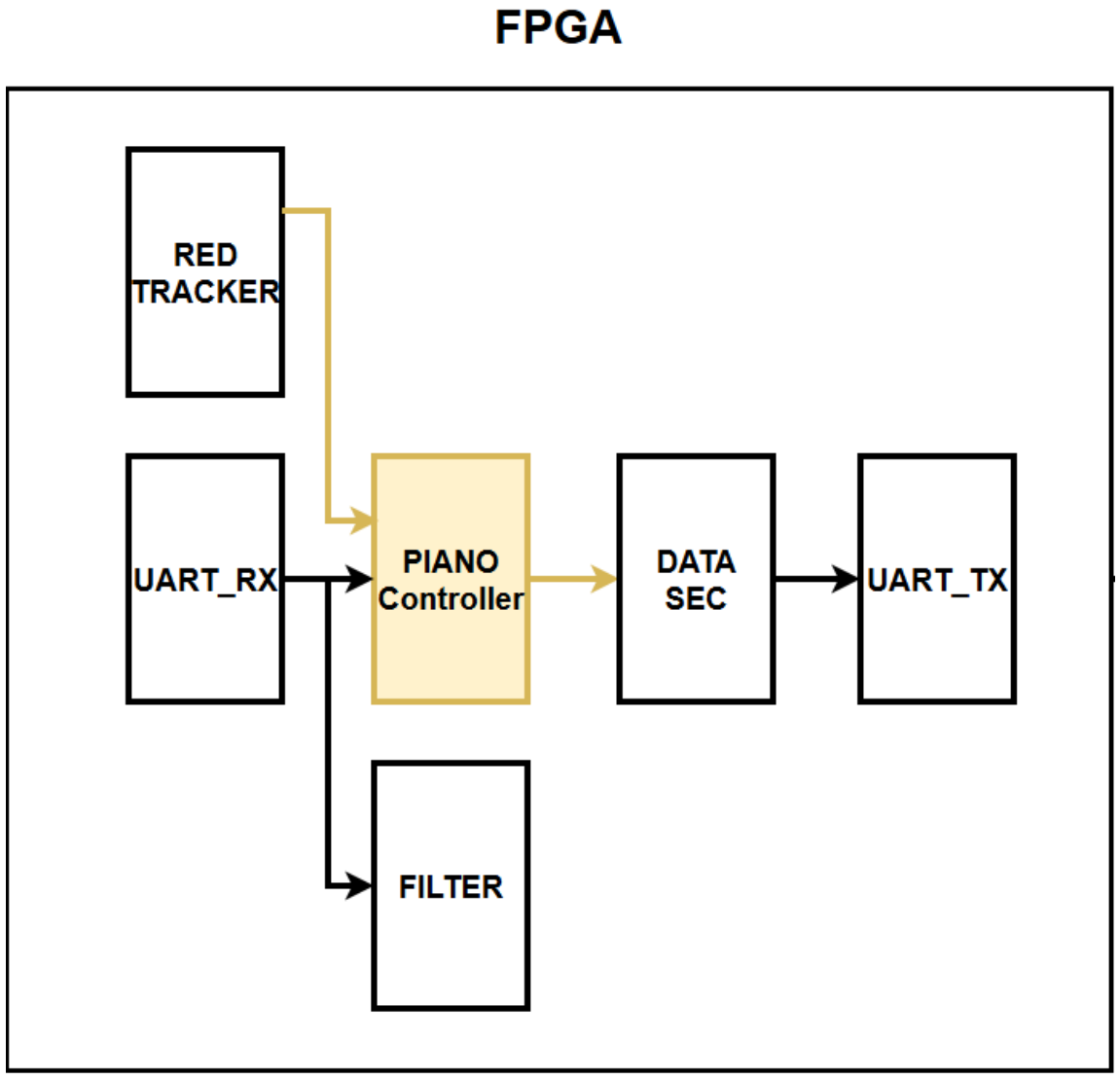
→ 3. Purity : G, B가 일정 수준(Tmax) 이하

- **RED Tracker**

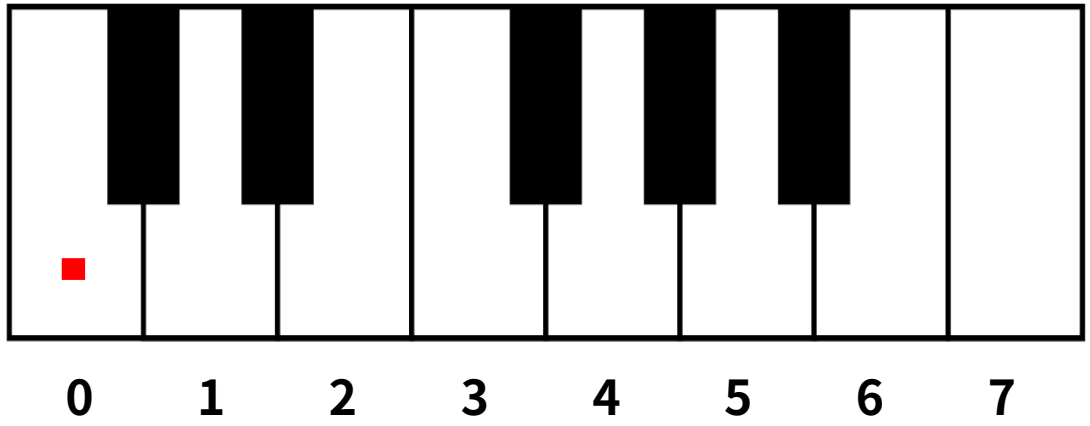
: 작은 점/홀/외곽 잡음을 제거하여 정확한 Red 영역만 인식. Shift Register로 구현



PIANO Controller



PIANO Controller



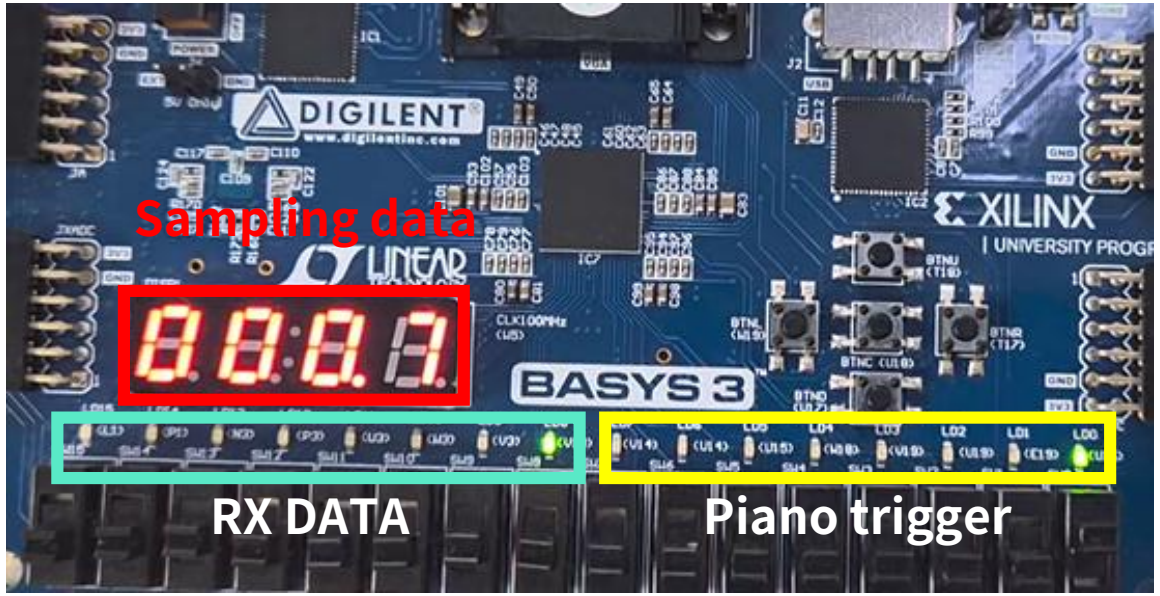
RED Tracker 모듈에서 받은 좌표

```
if (center_y >= 320 && center_y <= 479) begin
    if (center_x >= 40 && center_x <= 109) note_next = 0;
    else if (center_x >= 110 && center_x <= 179) note_next = 1;
    else if (center_x >= 180 && center_x <= 249) note_next = 2;
    else if (center_x >= 250 && center_x <= 319) note_next = 3;
    else if (center_x >= 320 && center_x <= 389) note_next = 4;
    else if (center_x >= 390 && center_x <= 459) note_next = 5;
    else if (center_x >= 460 && center_x <= 529) note_next = 6;
    else if (center_x >= 530 && center_x <= 599) note_next = 7;
    else note_next = 8'hFF;
end else note_next = 8'hFF;
```

UART로의 전송 신호

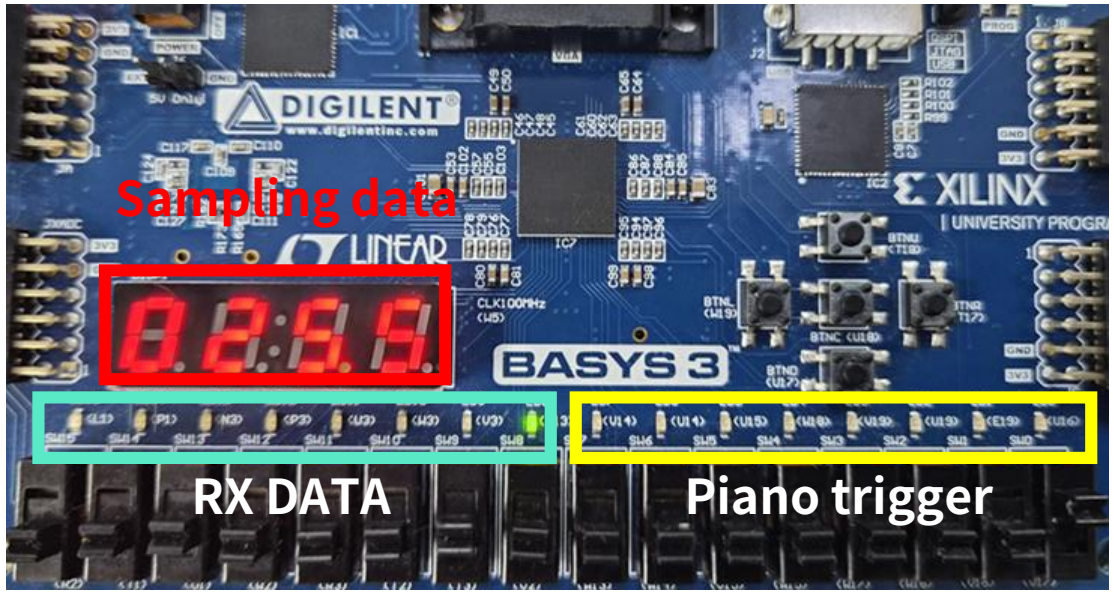
UART & FND

- Right DATA



낮은 도 : 1 , 레 : 2 … 높은 도 : 7 출력

- Fail DATA



‘255’ 출력

Filter

FILTER

[눈 내리는 필터]

- **눈 좌표 생성**

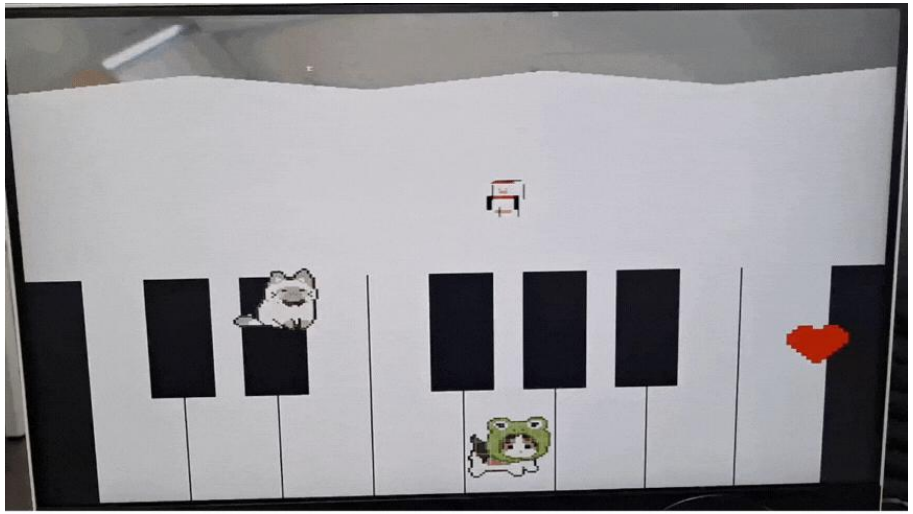
```
assign rand_big = ((x >> 2) * 73) + (((y - snow_timer) >> 2) * 97);
assign falling_snow = (rand_big[10:0] == 0);  // 속도

assign rand_small = (x * 123) ^ (y * 91) ^ (snow_timer * 57);
assign small_snow = (rand_small[9:0] == 10'h001);  //속도
```

화면 좌표와 타이머 기반 난수를 조합해 눈송이 생성

- **눈 쌓이는 로직**

```
always_comb wave_height = (x[7] ? ~x[6:2] : x[6:2]) >> 1;
assign piled_snow = (y < 250) && (y >= (250 - pile_base - wave_height));
```

wave 기반으로 자연스럽게 눈이 쌓이도록 구현



[벚꽃 필터]

- **꽃잎 좌표 생성**

```
assign rand_big_next = (({3'b0, x_scaled} + varied_timer) * 73)
                     + ((y_scaled_full - varied_timer) * 97);
```

좌표 스케일링과 난수 기반으로 꽃잎을 흩날리게 생성

- **꽃잎 기본 속도 생성**

```
logic falling_petal = (rand_big_q[6:0] == 0);
```

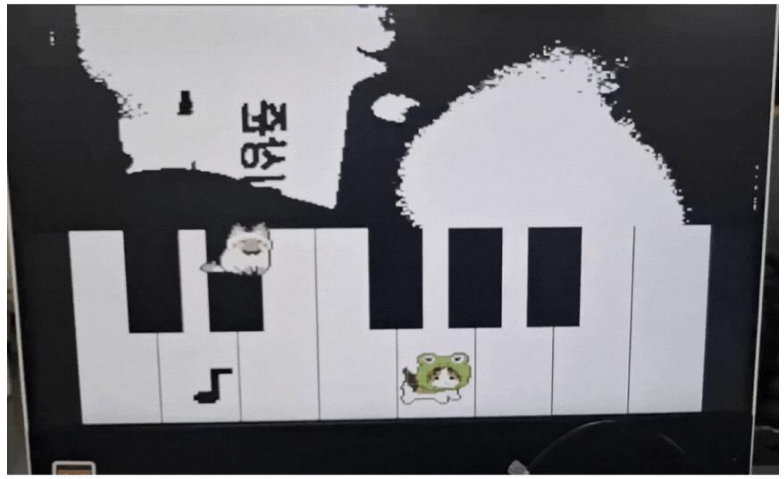
바닥에 점진적으로 쌓이는 로직을 적용

FILTER

[흑백 필터]

```
localparam THRESHOLD = 4'd8;

logic [5:0] sum_rgb;
logic [3:0] avg_luminance;

assign sum_rgb = i_r + i_g + i_b;
assign avg_luminance = sum_rgb / 3;

logic [3:0] binary_val;
assign binary_val = (avg_luminance > THRESHOLD) ? 4'hF : 4'h0;

assign o_r = binary_val;
assign o_g = binary_val;
assign o b = binary val;
```

(r+g+b)/3 후, 그 값보다 밝으면 흰색, 어두우면 검정색 출력



[번개 필터]

```
always_comb begin
    dark_r = cam_r >> 1;
    dark_g = cam_g >> 1;
    dark_b = (cam_b >> 1) + (cam_b >> 2);
    if (in_bolt || in_branch1 || in_branch2 || in_bolt2) begin
        out_r = 4'hF;
        out_g = 4'hF;
        out_b = 4'hC;
    end else if (pika_px != 12'h000) begin
        out_r = pika_px[11:8];
        out_g = pika_px[7:4];
        out_b = pika_px[3:0];
    end else if (is_flash) begin
        sum_r = cam_r + 3;
        sum_g = cam_g + 3;
        sum_b = cam_b + 3;
        out_r = (sum_r > 15) ? 4'd15 : sum_r[3:0];
        out_g = (sum_g > 15) ? 4'd15 : sum_g[3:0];
        out_b = (sum_b > 15) ? 4'd15 : sum_b[3:0];
    end else begin
        out_r = dark_r;
        out_g = dark_g;
        out_b = dark_b;
```

배경은 어둡게 처리하고,
랜덤 flash, bolt, branch 효과를 조건 기반으로 출력

동작 영상

Trouble Shooting

Trouble Shooting(1)

문제 상황 : 소리 연속 출력

‘A’ 키 입력은 정상적으로 인식.
But, TX 전송 과정에서 동일 데이터가 연속적으로 송신되어 **소리가 지속적으로 출력**되는 현상 발생.

해결 방안 : Delay module 설계

- TX 제어 모듈 수정 : 0.5초 주기로 TX송신 신호 카운터로 출력.
- FIFO 구조 추가 : 연속 입력 발생 시, 데이터를 순차적으로 송신하여 안정적 전송 가능.

Trouble Shooting(2)

문제 상황 : 건반 이동 문제

특정 음정으로 넘어갈 때, 스위치를 통해 연속적 입력을 받음
필터 외 구간을 거치거나 다음 음을 경유하는 방식 사용

해결 방안 : PC키보드 연결

PC키보드를 연결하여 특정 키(“A”)를 눌렀을 때 소리를 출력하는 트리거 방식 사용

고 찰

느낀점 & 배운점

이서영

좌표와 신호 타이밍을 정확히 맞추지 않으면 필터가 원치 않는 위치에 그려지는 어려움을 경험했습니다. 특히 여러 레이어를 겹칠 때는 우선순위 처리와 조건 범위 설정이 꼬이면 화면 전체가 덮이는 문제가 자주 발생해 디버깅 역량이 크게 향상됐습니다. 또한 팀원들과 기능을 연동하면서 모듈 인터페이스를 맞추는 과정에서 정확한 신호 정의와 소통의 중요성을 다시 한번 느낄 수 있었습니다.

박찬호

전에 개인 프로젝트를 할 때는 문제가 생기면 해결이 안 될 때가 많았습니다.
하지만 이번에는 문제가 생길 때 조원과 모여서 토론을 진행하며, 혼자서는 미처 생각하지 못했던 접근 방식을 팀원들을 통해 발견할 수 있었습니다.
기술적인 성장도 컸지만, 협업의 중요성을 깨닫는 경험이 되었습니다.

느낀점 & 배운점

김호준

PC-FPGA 간 인터페이스에서 실시간 디버깅이 어려워 데이터 검증과 타이밍 조정에
많은 시간이 필요했지만,문제 원인을 차근히 분석하고 해결하며 연동을 성공적으로 수행했습니다.
이를 통해 임베디드 시스템 연동과정에서의 디버깅 중요성을 체감하고 한 단계 성장할 수 있었습니다.

이승후

필터 설계와 구현 과정에 대한 이해를 높일 수 있었고, 실제 적용 결과를 확인하며 이론이 어떻게 활용되는지
알 수 있었습니다. 더하여 필터 관련 기술을 더욱 깊이 탐구하는 계기가 되었습니다.
팀 프로젝트를 진행하면서 협업을 통한 소통과 역할 분담의 중요성을 다시 한 번 느끼게 되었고,
앞으로의 프로젝트에서 더 나은 팀워크를 발휘하기 위한 방향을 생각해보는 기회가 되었습니다.

Thank you