VGA_OV7670을 이용한 FRUIT NINJA 게임 구현하기

닌자 거북이 (7조) 

이윤지, 김태원, 석경현, 한규혁

Contents

Development Environment

SystemVerilog

Language

Board

VIVADO

Tools

Camera

VGA_port

Introduction

SOFTWARE

- 개발과 업데이트가 빠르고 유연한 대신, 타이밍과 리소스가 os·엔진에 의존한다.

HARDWARE

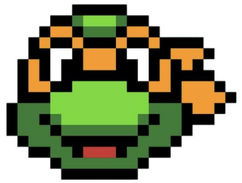
- 병렬성과 타이밍 제어가 탁월하지만, 설계가 어렵고 복잡한 게임에는 비현실적이다.

Project Overview

프로젝트 주제 | OV7670 카메라와 VGA를 이용하여 Fruit Ninja 게임 구현

프로젝트 목표 | 카메라 영상 처리를 이용한 카메라 기반 객체 인식을 실제 게임으로 구현하고자 함

팀원 소개



김태원

게임 UI 구현

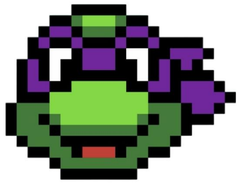
설계

- 물리엔진
- 골든 레퍼런스
- SCCB

한규혁

설계

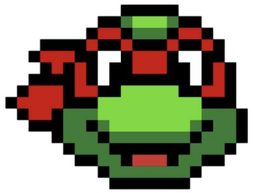
- cursor tracking
- rendering
- collision detect



석경현

설계

- line buffer
- 3x3 red filter
- timing control

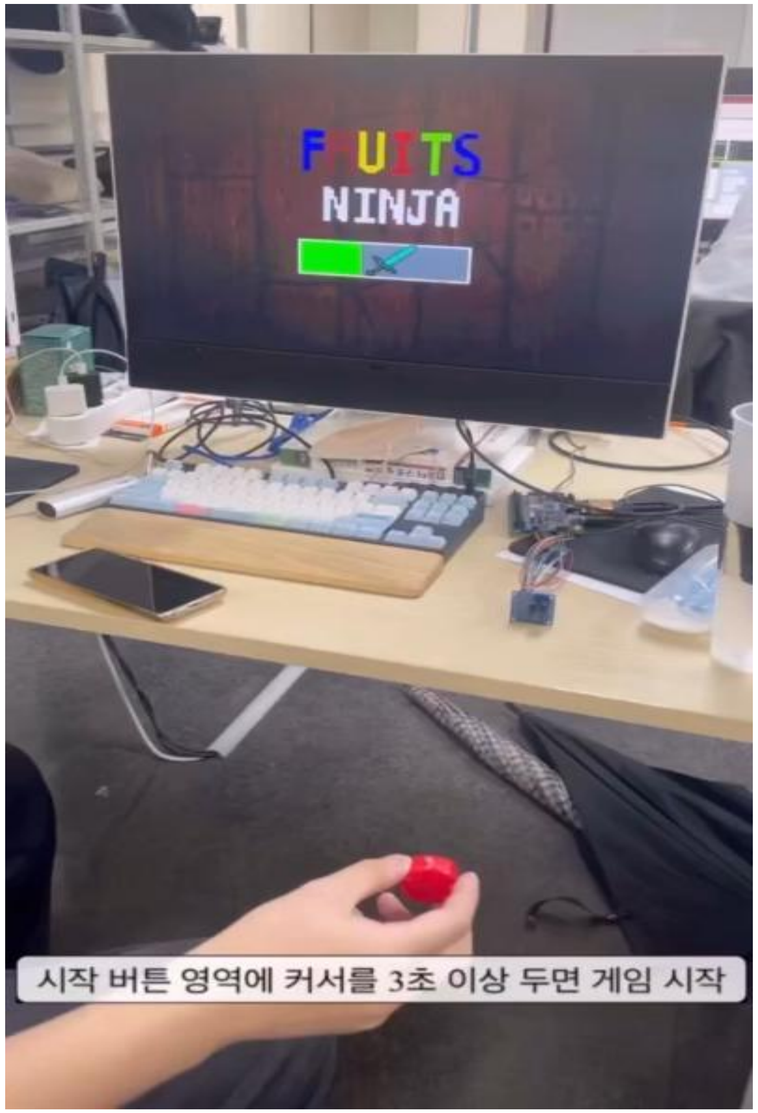


이윤지

게임 UI 구현

설계

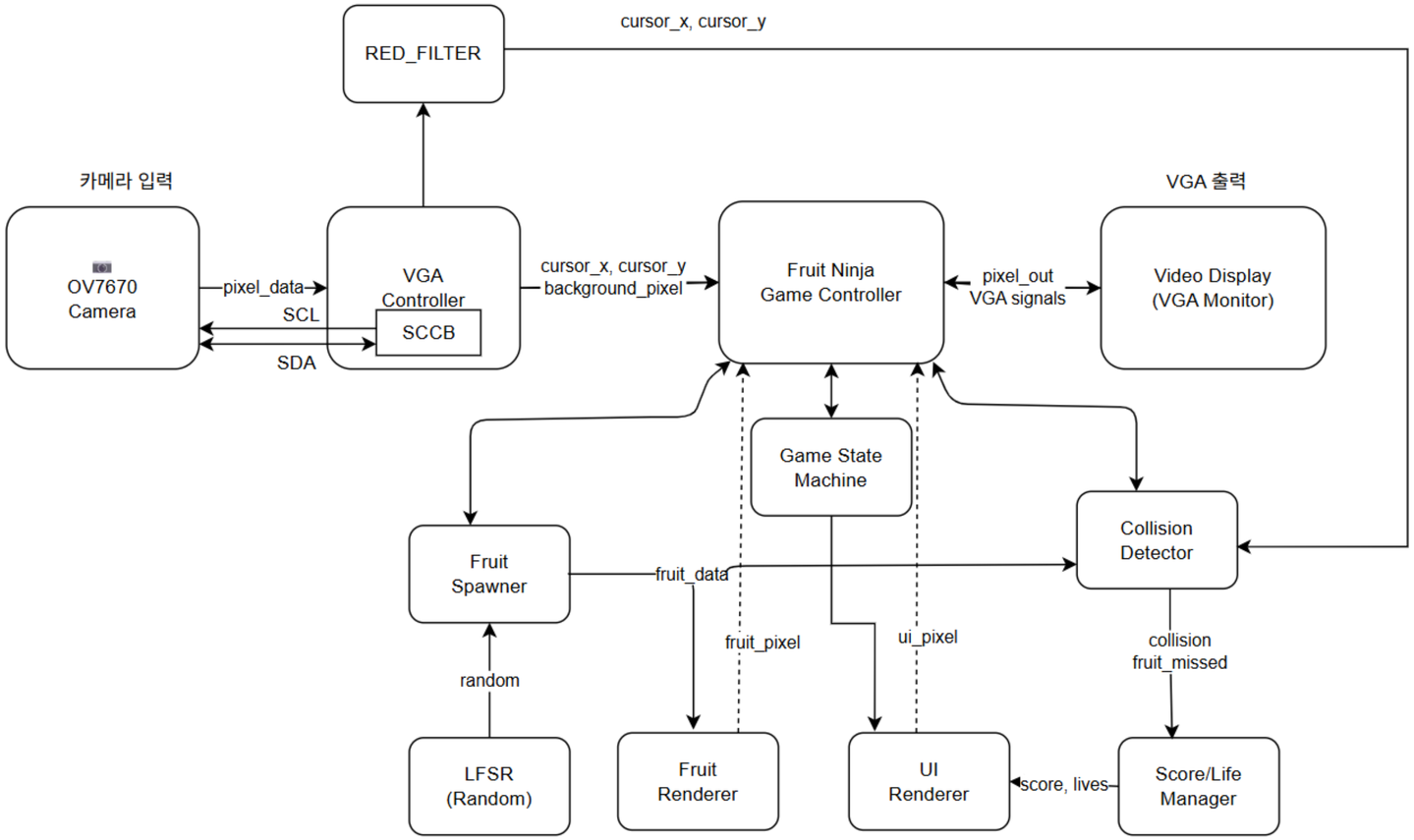
- rendering
- Score/life mange
- Gaussian filter


FRUITS
NINJA
시작 버튼 영역에 커서를 3초 이상 두면 게임 시작

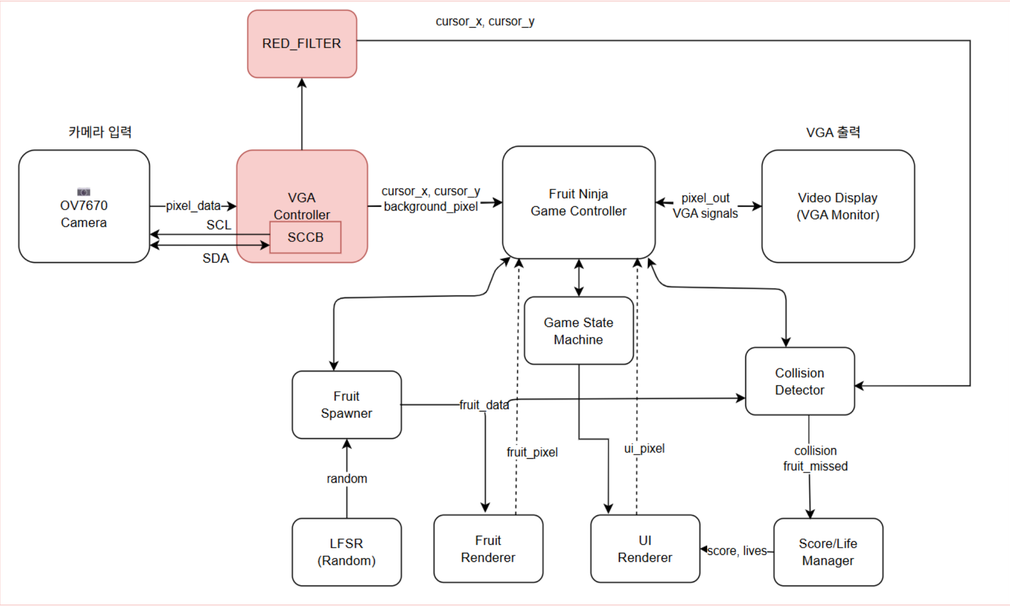
게임 규칙


FRUIT
NINJA
SCORE 01260
LIVES
SLASH!
FRUITS
CURSOR
SCORE RULES
X10
X20
X30
X40
X50
X100
LOSE A LIFE

Block Diagram



VGA Controller



SCCB

- 역할 : FPGA가 마스터가 되어 카메라의 레지스터를 순차적으로 초기 설정
- SIO_D 라인은 IOBUF 버퍼를 사용하여 송신 / 수신
- 통신 프로토콜 : ID 전송 -> 레지스터 주소 -> 데이터 값
- 각 바이트 전송 후 카메라의 ACK 신호 확인
- 내장 ROM에 저장된 75개의 설정 값을 FSM이 순차적으로 로드하여 전송함.

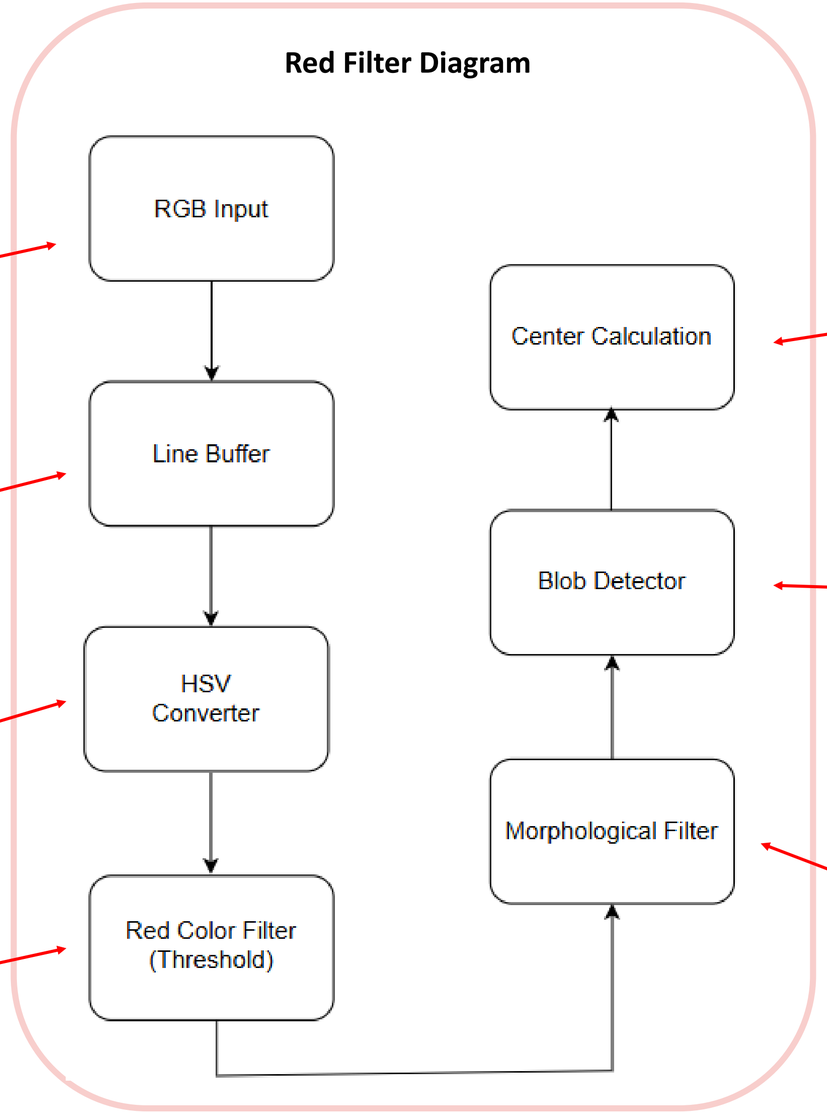
Red Filter

RGB 입력
- OV7670 카메라에 RGB444 영상 수신

3x3 윈도우 생성
- 9픽셀로 묶어서 라인버퍼 구성
- 공간 필터링, 색 검출 안정성 확보

HSV 색 공간 변환
- Hue 값 중심의 색 분리 수행

Red 색상 Thresholding

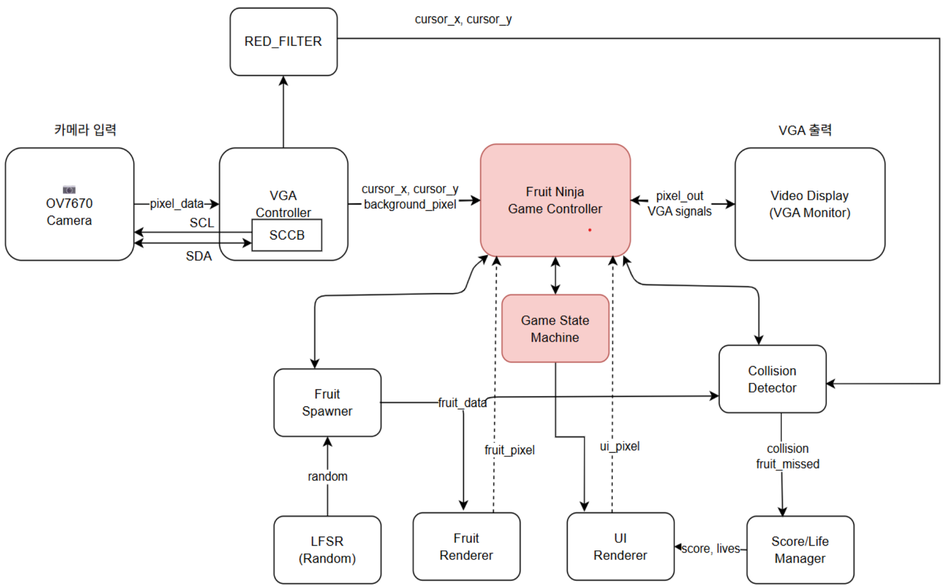


중심점 계산
- 검출된 blob의 중심 좌표 계산

Blob Detection
- 이미지 전체를 스캔해 red blob의 최소 x, y 위치 계산

Morphological Filter
- 아주 작은 노이즈 제거
- 끊긴 빨간 영역을 연결
- Blob 내부의 빈 공간 메꾸기

Game Controller

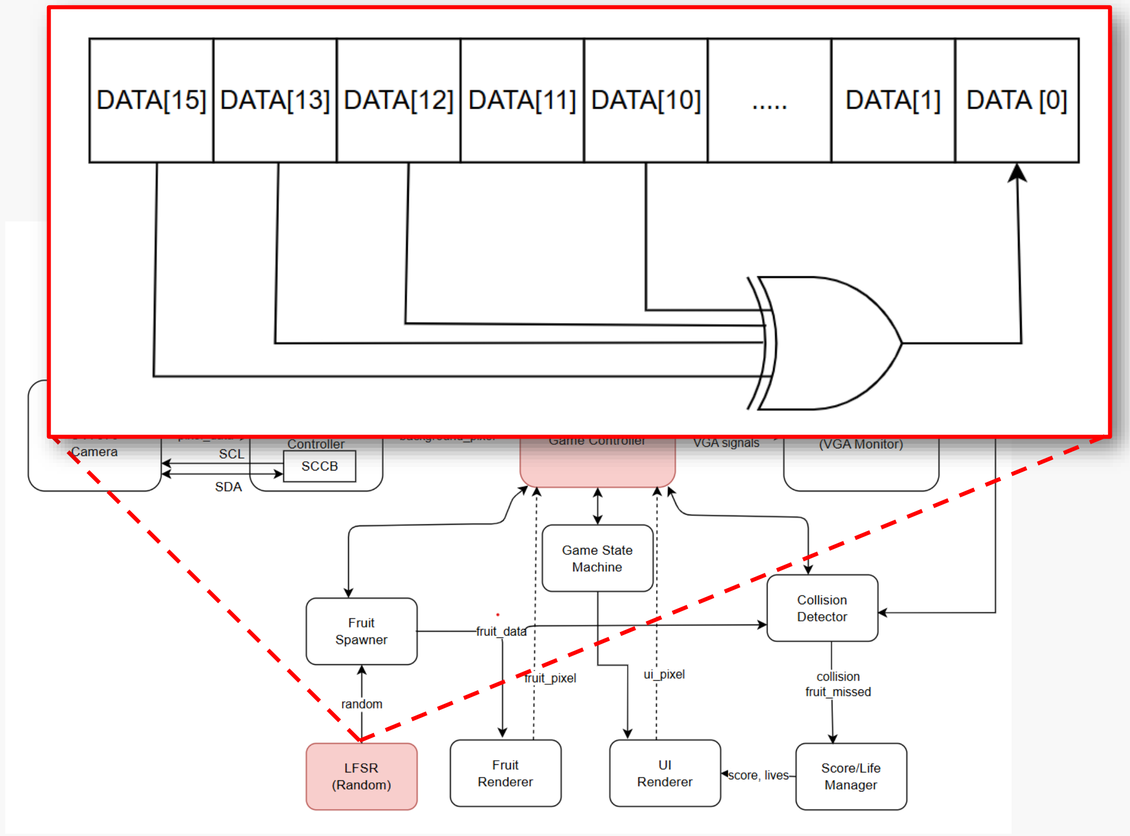


Game State

```
ST_TITLE: begin
    if (on_button) begin
        if (hover_count >= HOVER_TIME) begin
            state_reg <= ST_PLAYING;
            hover_count <= 8'd0;
            game_start_pulse <= 1'b1;  // 게임 시작 펄스
        end else begin
            hover_count <= hover_count + 1'b1;
        end
    end else begin
        hover_count <= 8'd0;
    end
end

ST_PLAYING: begin
    if (lives == 2'd0) begin
        state_reg <= ST_GAMEOVER;
        hover_count <= 8'd0;
    end
end

ST_GAMEOVER: begin
    if (on_button) begin
        if (hover_count >= HOVER_TIME) begin
            state_reg <= ST_TITLE;
            hover_count <= 8'd0;
        end else begin
            hover_count <= hover_count + 1'b1;
        end
    end else begin
        hover_count <= 8'd0;
    end
end
```

- 게임 상태는 TITLE → PLAYING → GAMEOVER → TITLE 순서로 순환함.
- TITLE에서 Start 버튼 위에 커서가 3초 머무르면 start_pulse가 발생해 PLAYING으로 진입.
- PLAYING에서 목숨이 0이 되는 순간 GAMEOVER로 전환.
- GAMEOVER에서 Menu 버튼 위에 커서가 3초 머무르면 TITLE로 복귀.

Game Controller



LFSR

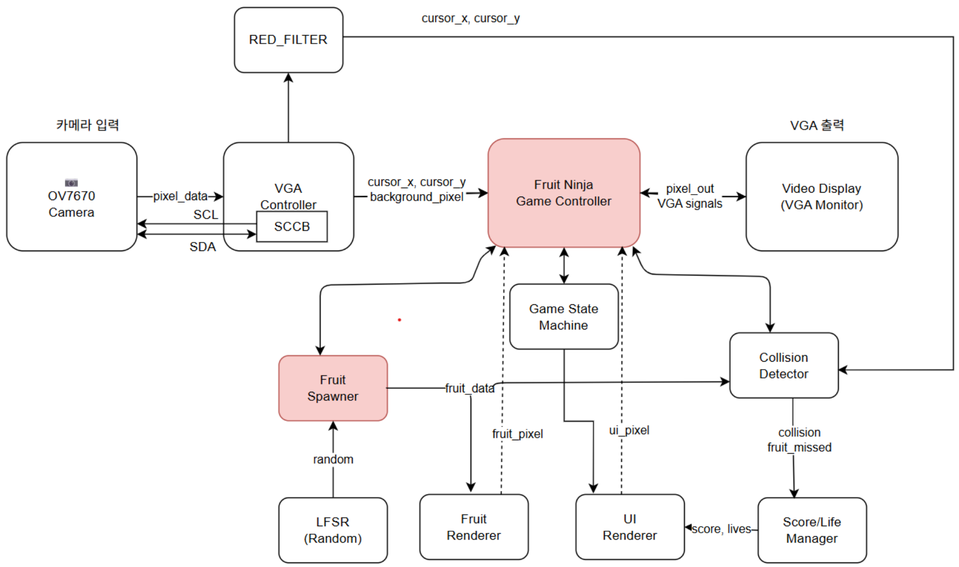
$$b_{\text{new}} = v_{15} \oplus v_{13} \oplus v_{12} \oplus v_{10}$$

$$\text{LFSR}_{\text{next}} = (v \ll 1) + b_{\text{new}}$$

- $\oplus$: XOR 연산
- $\ll 1$: 1비트 왼쪽 시프트

- 15·13·12·10번 비트를 XOR한 값을 LFSR의 LSB에 넣고, 레지스터를 1bit Left Shift.
- 하드웨어에서 구현이 쉬운 선형 귀환 시프트 레지스터 (LFSR)를 사용해, 각 과일마다 다른 시작 위치와 속도를 만든다.

Game Controller



Fruit Spawner

- **포물선 초기 값 수식**

초기 위치 (x_0 , y_0)

$$x_0 = \begin{cases} 60 + r_x, & \text{sel} = 0 \\ 200 + r_x, & \text{sel} = 1 \\ 340 + r_x, & \text{sel} = 2 \\ 480 + r_x, & \text{sel} = 3 \end{cases}$$

$$y_0 = 520$$

x시작 위치는 4개의 베이스 위치 + 랜덤 오프셋

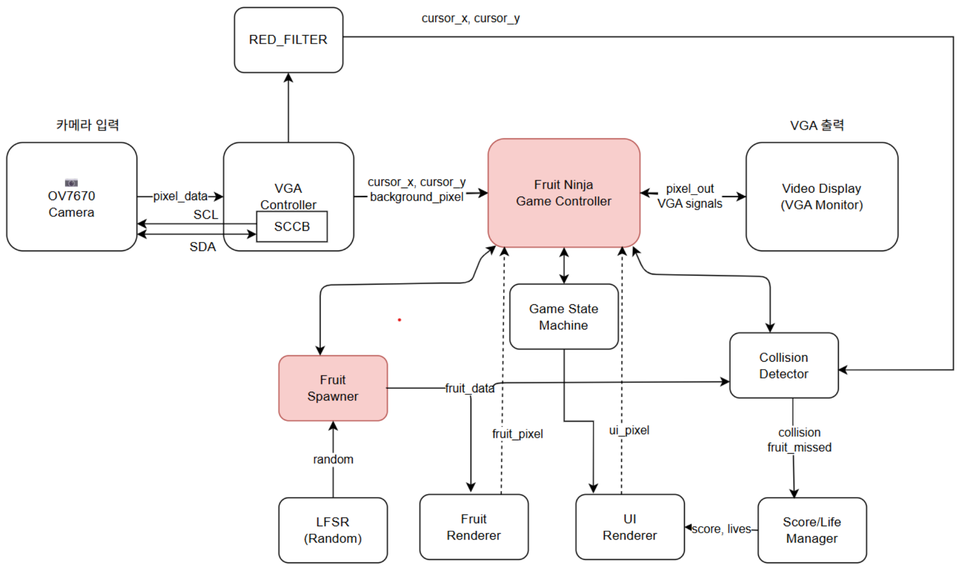
- BASE : 60, 200, 340, 480
- r_x: 0 ~ 63
- r_v: 0 ~ 63
- r_y: 0 ~ 128

초기 속도 $(v_{x,0} , v_{y,0})$

$$v_{x,0} = \begin{cases} +16 + r_v, & \text{sel} \in \{0, 1\} \quad (\text{오른쪽으로}) \\ -(16 + r_v), & \text{sel} \in \{2, 3\} \quad (\text{왼쪽으로}) \end{cases}$$

$$v_{y,0} = -160 - r_y \quad (\text{위쪽 방향})$$

Game Controller



Fruit Spawner

- **포물선 운동 구현**

위치 증가량

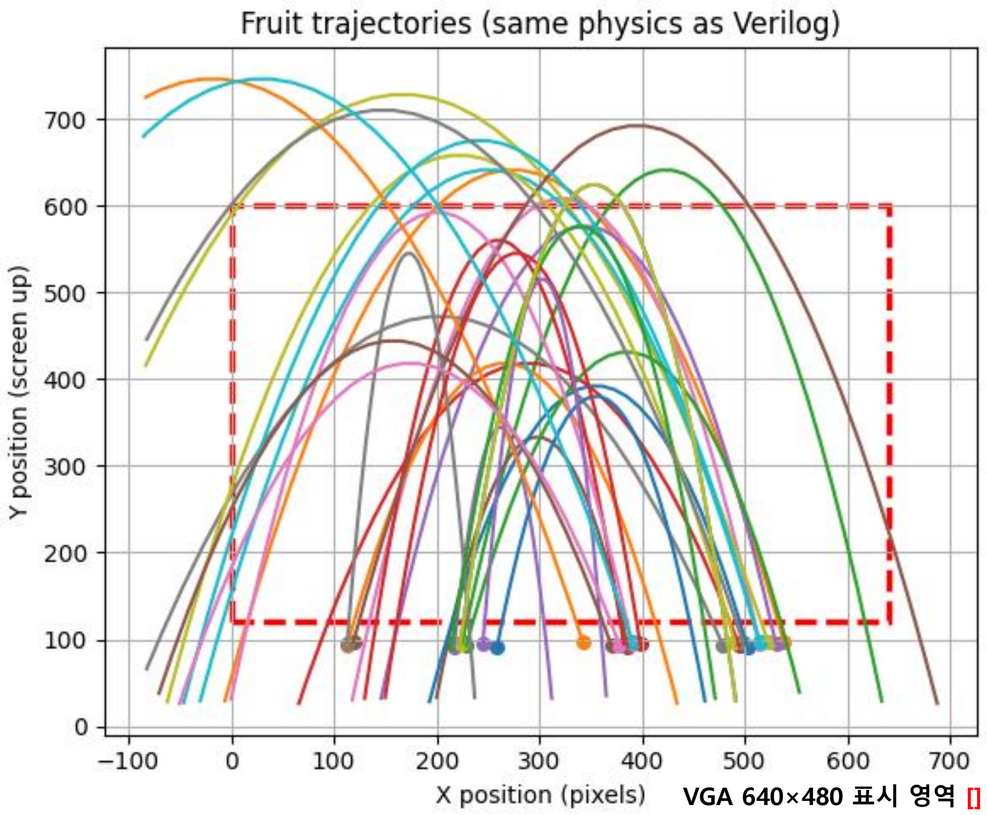
$$x_{n+1} = x_n + \frac{v_{x,n}}{16}, \quad y_{n+1} = y_n + \frac{v_{y,n}}{16}$$

- 매 프레임마다 위와 같은 포물선 방정식을 적용하여 y > 560 || x < -80 || x >720 이면 과일을 비활성화

중력에 의한 y 속도 변화

$$v_{y,n+1} = v_{y,n} + 4$$

궤적 분포와 초기 속도



Python으로 시뮬레이션한 궤적 그래프

=== Initial velocities (Verilog physics 기준) ===

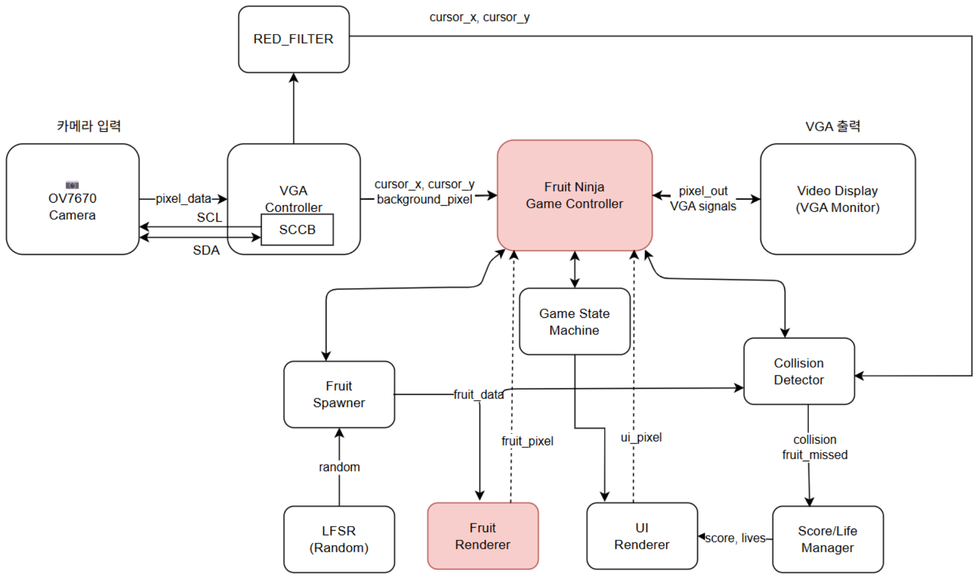
idx	x0	y0	vx[pix/f]	vy[pix/f]	speed
0	221	520	2.81	-15.81	16.06
1	542	520	-3.19	-16.19	16.50
2	222	520	3.06	-16.06	16.35
3	498	520	-3.44	-12.44	12.90
4	533	520	-2.19	-15.19	15.34
5	388	520	-1.62	-10.62	10.75
6	517	520	-2.69	-15.69	15.92
7	483	520	-4.44	-13.44	14.15
8	527	520	-4.44	-17.44	17.99
9	518	520	-3.19	-16.19	16.50
10	505	520	-2.94	-11.94	12.29
11	110	520	3.50	-12.50	12.98
12	226	520	3.56	-12.56	13.06
13	384	520	-1.88	-14.88	14.99
13	384	520	-1.88	-14.88	14.99
14	216	520	1.81	-10.81	10.96
15	115	520	4.00	-17.00	17.46
16	393	520	-2.38	-15.38	15.56
17	112	520	1.75	-14.75	14.85
18	494	520	-3.44	-16.44	16.79
19	518	520	-3.69	-16.69	17.09
20	257	520	2.56	-11.56	11.84
21	347	520	-4.62	-17.62	18.22
22	213	520	2.06	-15.06	15.20
23	400	520	-1.62	-14.62	14.72
24	244	520	1.06	-14.06	14.10
25	374	520	-3.88	-12.88	13.45
26	382	520	-3.38	-12.38	12.83
27	503	520	-4.19	-17.19	17.69
28	221	520	2.81	-15.81	16.06
29	395	520	-4.62	-17.62	18.22

- 각 궤적에 대응하는 초기 위치·속도 ($v_{x,0}$, $v_{y,0}$) 및 속도 크기 | LFSR로 생성된 물리 파라미터를 수치적으로 확인

결과 확인

- 이 그래프와 테이블을 통해 과일이 항상 화면 안에서 자연스러운 포물선을 그리도록 물리 파라미터를 조정했음을 확인

Game Controller



Fruit Render

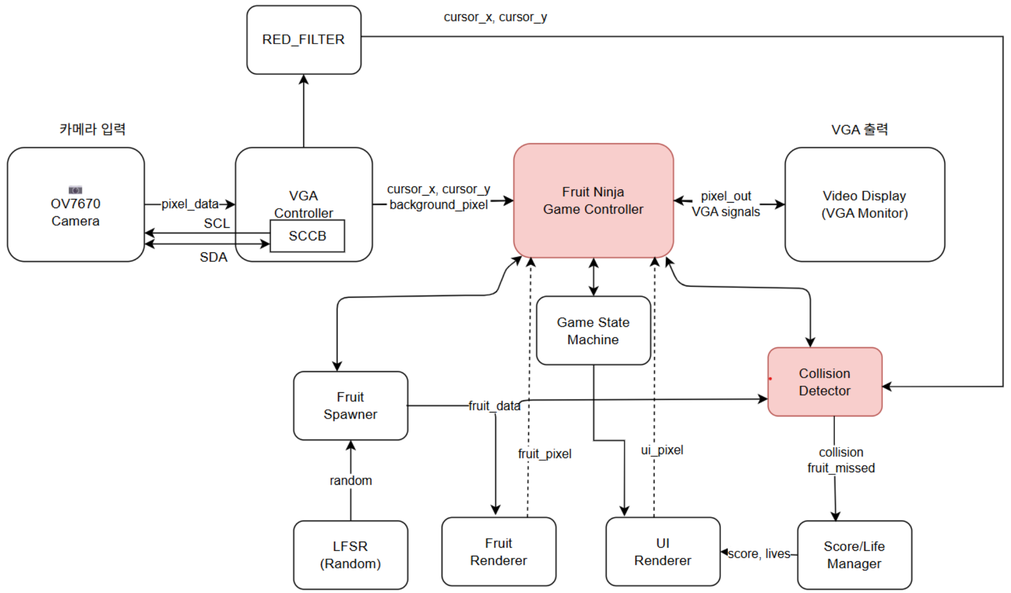
- 과일영역 rel_x = x_pixel – fruit_x + HALF_SIZE; 과일 크기의 -40~+40까지

```
for (int j = MAX_FRUITS - 1; j >= 0; j--) begin
    if (fruit_active[j] && !fruit_sliced[j] && !hit_found) begin
        logic signed [11:0] rel_x, rel_y;
        rel_x = $signed({2'b0, x_pixel}) - $signed({2'b0, fruit_x[j]}) + HALF_SIZE;
        rel_y = $signed({2'b0, y_pixel}) - $signed({2'b0, fruit_y[j]}) + HALF_SIZE;

        if (rel_x >= 0 && rel_x < DISPLAY_SIZE && rel_y >= 0 && rel_y < DISPLAY_SIZE) begin
            hit_found = 1'b1;
            // 80/16 = 5 → rel / 5
            // 근사: (rel * 13) >> 6 ≈ rel / 5
            sprite_x = (rel_x * 13) >> 6;
            sprite_y = (rel_y * 13) >> 6;
            if (sprite_x > 4'd15) sprite_x = 4'd15;
            if (sprite_y > 4'd15) sprite_y = 4'd15;
            hit_type = fruit_type[j];
        end
    end
end
```

- 현재 VGA 픽셀 좌표가 각 과일의 80×80 히트박스 안에 들어왔는지 먼저 검사함.

- 박스 안에 들어온 경우, 그 좌표를 16×16 스프라이트 좌표로 크기를 변환해 ROM에서 해당 과일 픽셀 색을 읽어 옴.

Game Controller



Collision Detector

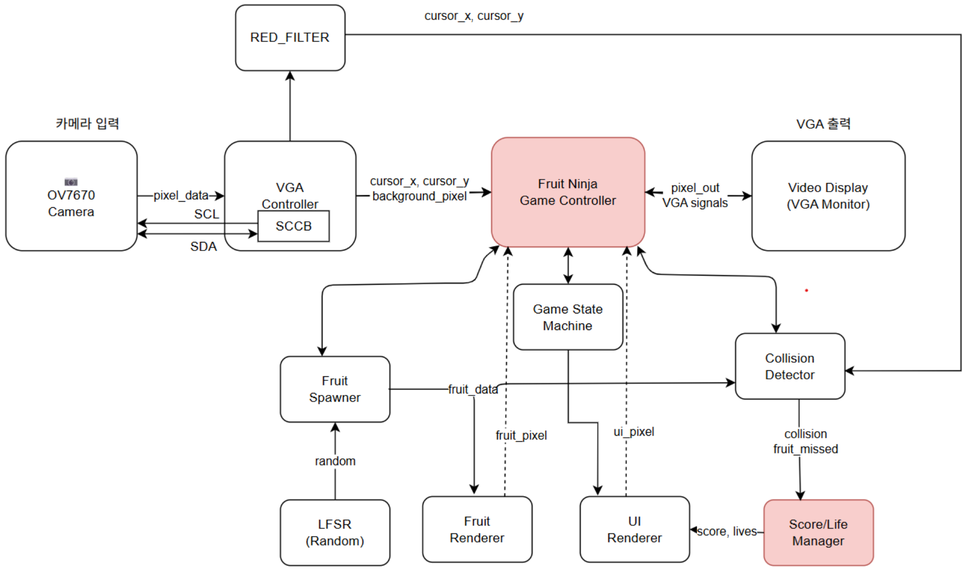
```
if (game_state == ST_PLAYING) begin
    for (i = 0; i < MAX_FRUITS; i++) begin
        // 비활성화된 과일은 already_missed 플래그 초기화
        if (!fruit_active[i]) begin
            already_missed[i] <= 1'b0;
            already_scored[i] <= 1'b0;
        end

        if (fruit_active[i] && !fruit_sliced[i]) begin
            // 충돌 체크
            logic signed [11:0] dx, dy;
            logic [10:0] abs_dx, abs_dy;

            dx = $signed({2'b0, cursor_x}) - $signed({2'b0, fruit_x[i]});
            dy = $signed({2'b0, cursor_y}) - $signed({2'b0, fruit_y[i]});
            abs_dx = dx[11] ? (-dx[10:0]) : dx[10:0];
            abs_dy = dy[11] ? (-dy[10:0]) : dy[10:0];

            if (cursor_active && abs_dx < HITBOX_SIZE && abs_dy < HITBOX_SIZE && !already_scored) begin
                collision_detected[i] <= 1'b1;
                already_scored[i] <= 1'b1;
            end

            // 놓침 체크: 아직 놓침 판정 안 받은 과일만
            if (fruit_entered[i] && fruit_vy[i] > 12'sd0 && fruit_y[i] > 10'd500 && !already_missed[i]) begin
                fruit_missed[i] <= 1'b1;
                already_missed[i] <= 1'b1;  // 이 과일은 이미 놓침 처리됨
            end
        end
    end
end else begin
    // 게임 중이 아니면 초기화
    already_missed <= '0;
    already_scored <= '0;
end
```

- 커서가 과일 중심에서 ±50px 이내이고 cursor_active가 1이면 베기 성공으로 판정.

- 과일이 아래로 떨어져 $y > 500$이 되면 놓친 과일로 처리하고, 이미 베었거나 놓친 과일은 다시 계산하지 않음.

- 게임 상태가 PLAYING이 아닐 때는 모든 충돌·놓침 판정을 초기화함.

Game Controller



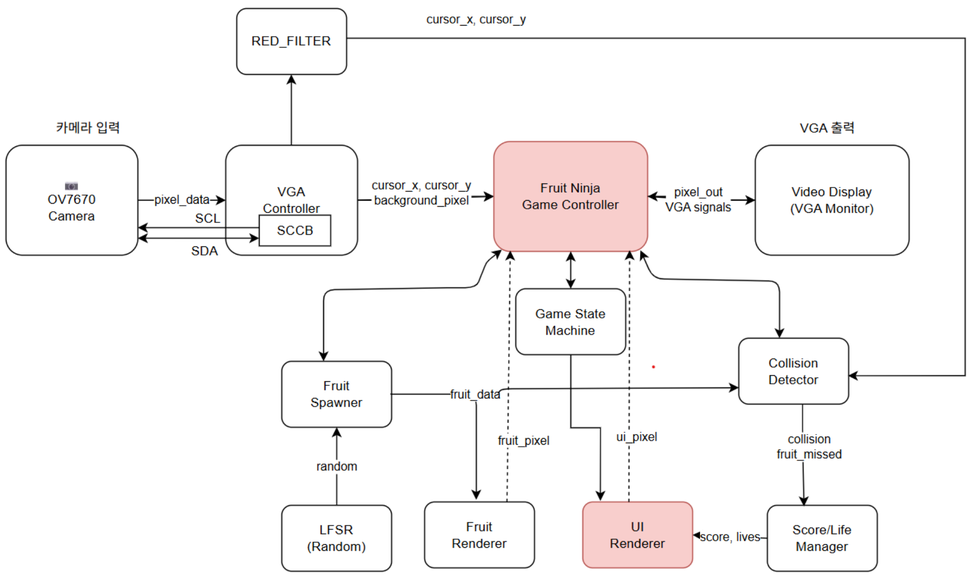
Score & life Manager

```
for (i = 0; i < MAX_FRUITS; i++) begin
    if (collision_detected[i]) begin
        if (fruit_type[i] == 3'd6) begin
            life_lost = 1'b1;
        end else begin
            score_add = score_add + {8'd0, get_fruit_score(fruit_type[i])};
        end
    end
    if (fruit_missed[i] && fruit_type[i] != 3'd6) begin
        life_lost = 1'b1;
    end
end
```

- 충돌이 발생한 과일의 타입이 3'd6(폭탄)이면 점수 대신 목숨을 1개 잃음.

- 폭탄이 아닌 과일은 타입 별 점수를 더해 총 점수에 반영하고, 폭탄을 제외한 과일을 놓치면 목숨을 1개 잃도록 처리함.

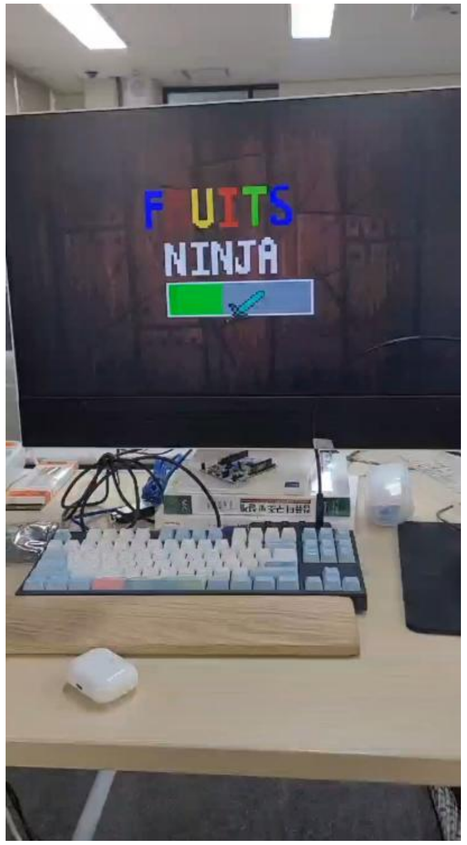
Game Controller



UI Render

- ST_TITLE에서는 타이틀 이미지와 Start 버튼 UI를 그려줌.
- ST_PLAYING에서는 점수 박스, 숫자 폰트, 라이프 하트 등 게임 HUD를 렌더링함.
- ST_GAMEOVER에서는 GAME OVER 텍스트, 최종 점수 박스, 메뉴 버튼 UI를 표시함.

Trouble Shooting



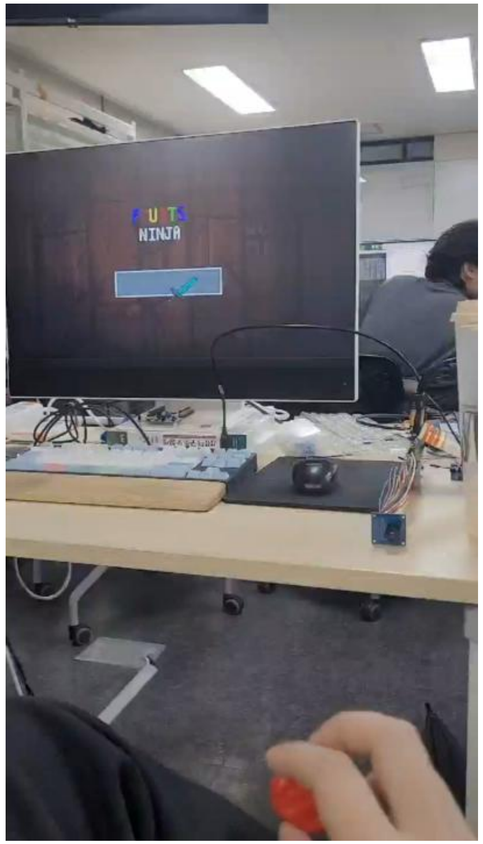
Trouble 1) life 가 2씩 감소하는 버그 발생

```
always_ff @(posedge clk) begin
    if (reset) begin
        collision_detected <= '0;
        fruit_missed <= '0;
        already_missed <= '0;
        already_scored <= '0;
    end else if (vsync_rising) begin
        collision_detected <= '0;
        fruit_missed <= '0;

        if (game_state == ST_PLAYING) begin
            for (i = 0; i < MAX_FRUITS; i++) begin
                // 비활성화된 과일은 already_missed 플래그 초기화
                if (!fruit_active[i]) begin
                    already_missed[i] <= 1'b0;
                    already_scored[i] <= 1'b0;
                end

                if (fruit_active[i] && !fruit_sliced[i]) begin
                    // 충돌 체크
                    logic signed [11:0] dx, dy;
                    logic [10:0] abs_dx, abs_dy;

                    dx = $signed({2'b0, cursor_x}) - $signed({2'b0, fruit_x[i]});
                    dy = $signed({2'b0, cursor_y}) - $signed({2'b0, fruit_y[i]});
                    abs_dx = dx[11] ? (-dx[10:0]) : dx[10:0];
                    abs_dy = dy[11] ? (-dy[10:0]) : dy[10:0];

                    if (cursor_active && abs_dx < HITBOX_SIZE && abs_dy < HITBOX_SIZE && !already_scored) begin
                        collision_detected[i] <= 1'b1;
                        already_scored[i] <= 1'b1;
                    end

                    // 놓침 체크: 아직 놓침 판정 안 받은 과일만
                    if (fruit_entered[i] && fruit_vy[i] > 12'sd0 && fruit_y[i] > 10'd500 && !already_missed[i]) begin
                        fruit_missed[i] <= 1'b1;
                        already_missed[i] <= 1'b1;  // 이 과일은 이미 놓침 처리됨
                    end
                end
            end
        end else begin
            // 게임 중이 아니면 초기화
            already_missed <= '0
            already_scored <= '0
```

- already_missed와 already_scored 플래그를 사용해 한 번 처리된 과일은 다시 카운트하지 않도록 함.

- 예를 들어 Frame 101에서 miss/score가 이미 기록되면, Frame 102에서는 신호가 나와도 무시되어 라이프와 점수가 중복으로 변하지 않음.

Trouble Shooting



Trouble 2) 상하 반전 모드에서 트래커/커서는 실제 손 위치와 반대로 움직이는 문제 발생.

```
logic [9:0] blob_x;
logic [9:0] blob_y;
logic blob_valid;
logic [11:0] red_debug;
logic [9:0] blob_y_flip;

assign blob_y_flip = 10'd479 - blob_y;
ISP_Red_Detection U_ISP_Red_Detection(
    .clk(clk),
    .reset(reset),
    .vsync(vsync),
    .de(de),
    .x_pixel(x_pixel),
    .y_pixel(y_pixel),
    .rgb444_in(rgb444_in),
    .rgb444_out(red_debug),
    .blob_x(blob_x),
    .blob_y(blob_y),
    .blob_valid(blob_valid)
);
```

```
Cursor_Controller U_Cursor_Controller(
    .clk(clk),
    .reset(reset),
    .target_x(blob_x),
    .target_y(blob_y_flip),
    .target_valid(blob_valid),
    .vsync(vsync),
    .cursor_x(cursor_x),
    .cursor_y(cursor_y),
    .cursor_angle(cursor_angle),
    .cursor_active(cursor_active)
);
```

blob_y_flip = 479 – blob_y로 좌표계를 다시 맞춰서 커서 인스턴스에 연결

Trouble Shooting

- **BEFORE**

- **AFTER**

Trouble 3) 빨간색 감지의 범위가 너무 넓고, 작은 빨간색에도 반응해서 커서가 그 중간 값으로 계속 움직임.

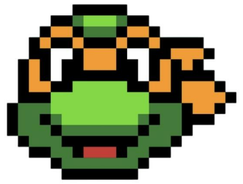
```
always_ff @(posedge clk or posedge reset) begin
    if (reset) begin
        blob_x <= 10'd320;
        blob_y <= 10'd240;
        blob_valid <= 1'b0;
    end else if (vsync_falling) begin
        // 최소 크기: 200 → 50 픽셀 (작은 영역도 검출)
        if (largest_count > 19'd20) begin
            blob_x <= largest_sum_x / largest_count;
            blob_y <= largest_sum_y / largest_count;
            blob_valid <= 1'b1;
        end else begin
            blob_valid <= 1'b0;
        end
    end
end
```

- 한 프레임 동안 여러 빨간 덩어리 중에서 가장 픽셀 수가 많은 blob을 largest에 저장
- 저장된 blob의 픽셀 수 largest_count가 20픽셀보다 클 때만 blob_valid=1로 내보내서 작은 빨간 덩어리는 전부 무시함.

```
always_ff @(posedge clk or posedge reset) begin
    if (reset) begin
        red_detected <= 1'b0;
        valid_out <= 1'b0;
    end else begin
        valid_out <= valid_in;

        if (valid_in &&
            sat >= SAT_MIN && sat <= SAT_MAX &&
            val >= VAL_MIN && val <= VAL_MAX) begin

            if ((hue >= HUE_LOW_MIN && hue <= HUE_LOW_MAX) ||
                (hue >= HUE_HIGH_MIN && hue <= HUE_HIGH_MAX)) begin
                red_detected <= 1'b1;
            end else begin
                red_detected <= 1'b0;
            end
        end else begin
            red_detected <= 1'b0;
        end
    end
```

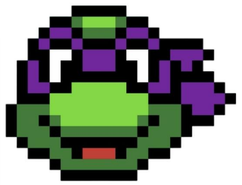
- SAT_MIN~SAT_MAX: 채도가 낮으면 빨간색으로 인식 안 함
- VAL_MIN~VAL_MAX: 명도가 너무 낮으면 빨간색으로 인식 안 함
- Hue 범위를 만족하는 경우라도 SAT, VAL 조건을 동시에 만족해야 “red_detected=1”

Conclusion



김태원

해당 포물선들의 궤적이 VGA 640*480 화면 안에 등장해야했기에 이에 대한 Constraints들 설정하는데 어려웠고, 이를 파이썬을 통해 확인하며 해결했습니다. 이 과정에서 하드웨어 구현 전 알고리즘이 논리적으로 맞는지 확인하는 골든 레퍼런스의 중요성을 깨달았습니다.

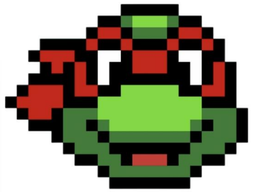


석경현

Image 신호를 처리하는 과정에서 3x3 Filter 연산과 리소스 절약을 위한 Line Buffer와 Pipelining을 통해 Delay를 보상하여 픽셀 좌표와 컨트롤 신호를 동기화 시키는 과정에서 Timing Control의 중요성을 깨닫게 되었습니다.

한규혁

빨간색 필터를 따라가는 커서를 만드는 과정에서 빠르게 움직일 때 그 움직임을 잡는 방법이나 부드럽게 움직이게 하는 것이 어렵다고 느꼈지만, 이 프로젝트를 계기로 영상 신호가 가진 프레임,노이즈,색상 변동성을 이해하고 이를 보정하는 필터 설계 능력을 키울 수 있었습니다.



이윤지

제목부터 과일 배경까지 여러 그래픽이 겹치지 않게 게임 환경을 구축한다는 것이 훨씬 까다롭다는 것을 배웠으며 과일이나 커서의 움직임 하나하나를 물리 법칙에 맞도록 조율하는 것이 얼마나 중요한지 깨달을 수 있었습니다.