FPGA 내 VGA를 활용한 장애물 피하기 게임 프로젝트

정윤철

본 디지털 하드웨어 시스템 설계 최종프로젝트는 강의 시간을 통해 배운 FPGA 내 부착된 스위치, 세그먼트, 버튼, VGA, 등 주변장치를 활용하는 방법을 응용하여 프로젝트를 설계합니다. RTL 상에서 코드를 통해 레지스터를 구현하고 구현된 모듈 간 연결을 통한 디지털 조합회로를 구성하고 시스템 입력에 따른 출력을 결정할 수 있는 로직 회로를 구성합니다.

본 프로젝트를 통해 디스플레이 동작 방식을 이해하고 메모리 구조를 응용하여 메모리 생성 방법과 관리 방법에 대해 학습하였습니다. VGA graphic Interface 펌웨어를 만들면서 디스플레이가 요구하는 동작 방식에 대해 학습하고 12비트 RGB 값 할당에 따른 디스플레이 출력 변화를 확인하여 디스플레이 내 원하는 형태의 graphic 출력을 할 수 있게 되었습니다. FPGA 내 HDL 코드를 통해 ROM을 구현하여 메모리 관리 방법에 대해 학습하여 본 프로젝트에 필요한 데이터를 SW로 구현된 ROM에 저장하고 사용할 수 있게 되었습니다.

본 프로젝트 수행 기간 중 초기에 프로젝트 주제로 게임을 기획하고 진행하였으나, VGA 부분만 완성되고 나머지 게임 관련 부분은 프로젝트 기간 중 코드로 작성된 모듈 형태로만 존재하고 시간 관계상 실제 동작 여부는 확인하지 못하였습니다.

프로젝트 블록 다이어그램:

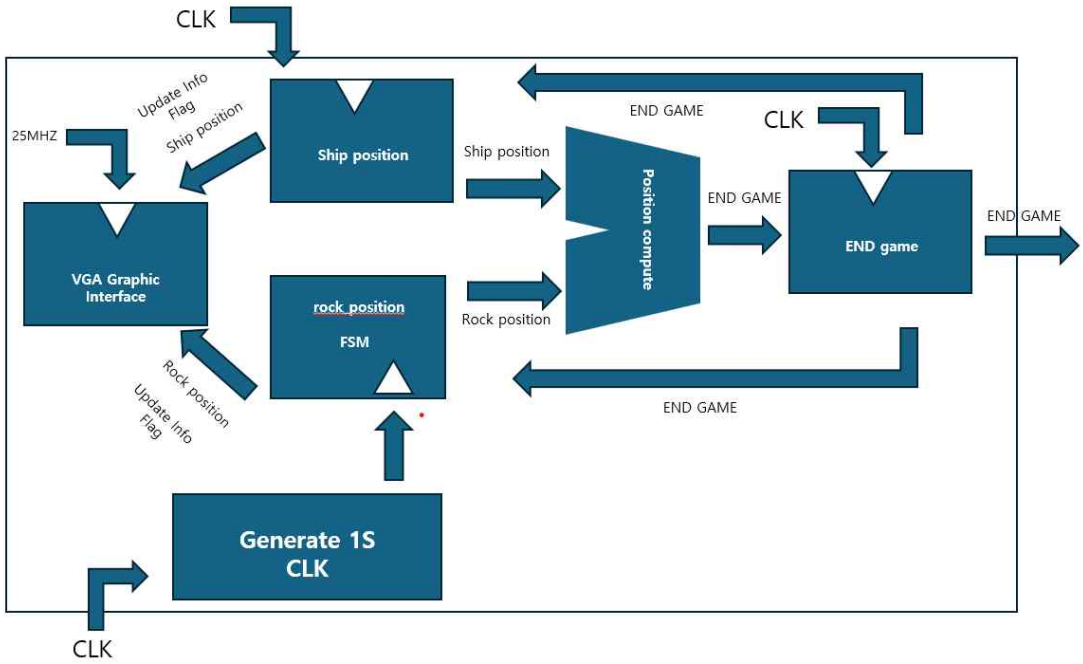


그림 1. 프로젝트 블록 다이어그램

본 프로젝트는 FPGA의 주변 장치로 연결된 키보드로부터 입력 받은 신호를 이용하여 비행체를 움직여 비행체에 다가오는 장애물을 피하는 게임입니다. FPGA 내 FSM을 이용하여 시간에 따른 상태 변화를 바탕으로 VGA를 통해 디스플레이에 화면을 출력하는 프로젝트입니다. 기존 제공된 FPGA VGA 출력 드라이버를 변형하여 원하는 위치에 원하는 출력을 나오도록 코드를 변형시켰고, SW로 데이터를 저장할 수 있는 롬을 만들어 디스플레이에 원하는 그래픽이 나오도록 코드를 작성하였습니다.

비행기의 위치와 장애물의 위치를 FSM으로 만들어 상태에 따른 디스플레이상에 물체를 표시할 수 있도록 설계하였습니다. Ship_position module과 Rock_position으로부터 얻은 비행기 위치와 장애물 위치를 position compute 모듈로 보냅니다. position compute에서는 비행기 위치와 장애물 위치가 같으면 게임이 끝나고, 위치가 같지 않는다면 게임이 끝나지 않고 지속해서 게임을 이어서 진행 하도록 합니다. Ship position과 Rock position 모듈로부터 얻어지는 비행기 위치와 장애물 위치는 VGA를 통해 디스플레이에 출력하기 위해서 현재 state 값을 직접 작성한 VGA Graphic interface 모듈로 보냅니다.

VGA Graphic interface에서 입력 받은 각각의 물체의 state 값에 따라 사전에 정의한 좌표로 변환됩니다. 변환된 좌표를 기반으로 사전에 정의한 물체의 그래픽 픽셀 크기를 바탕으로 디스플레이를 업데이트 합니다. 이때 업데이트하는 주기는 25MHz로 업데이트는 HSYNC와 VSYNC로 FPGA로부터 디스플레이에 출력하여 FPGA와 연결된 화면을 업데이트합니다. 25Mhz에서 생성되는 클럭 상승 엣지마다 입력하고자 하는 픽셀의 색상을 FPGA의 출력으로 생성해 디스플레이에 전달합니다. 이때 픽셀당 출력 가능한 최대 색상은 12비트입니다.

프로젝트 FSM state 구조의 형태:

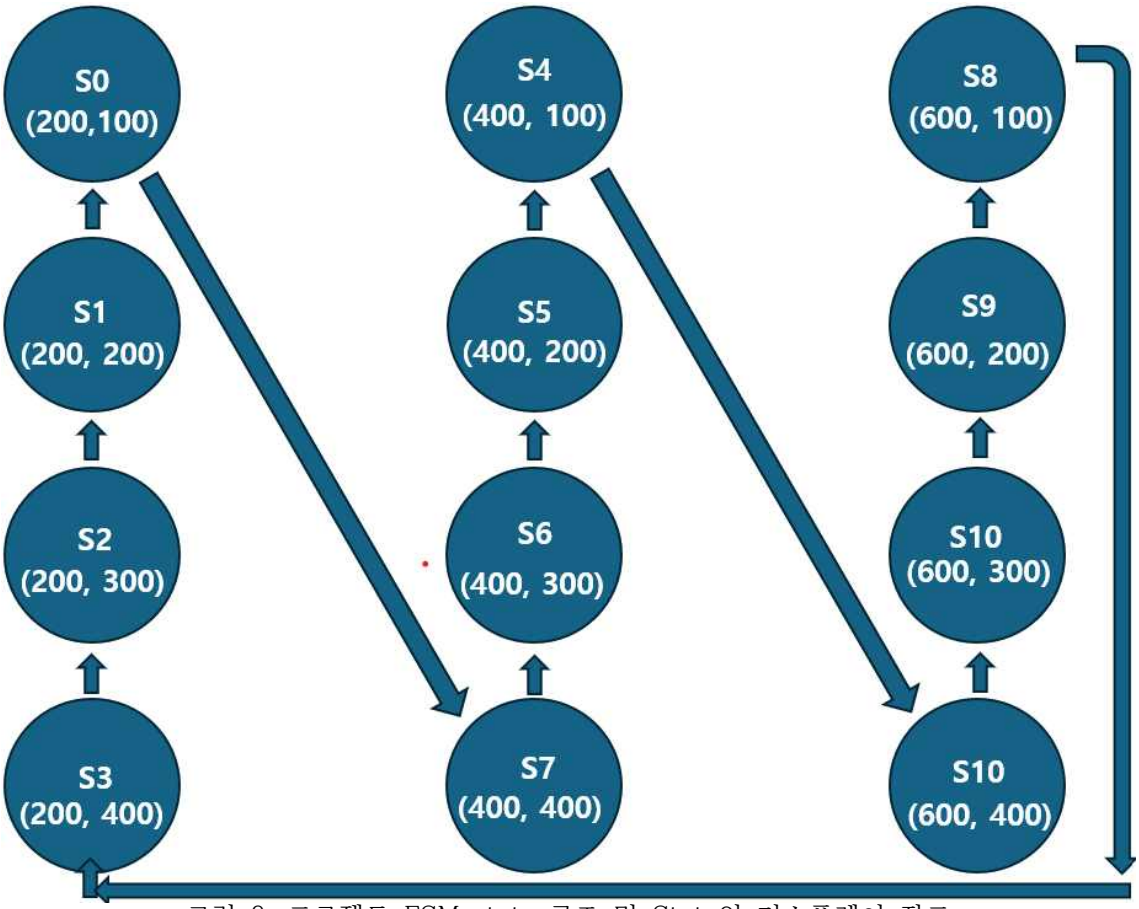


그림 2. 프로젝트 FSM state 구조 및 State의 디스플레이 좌표

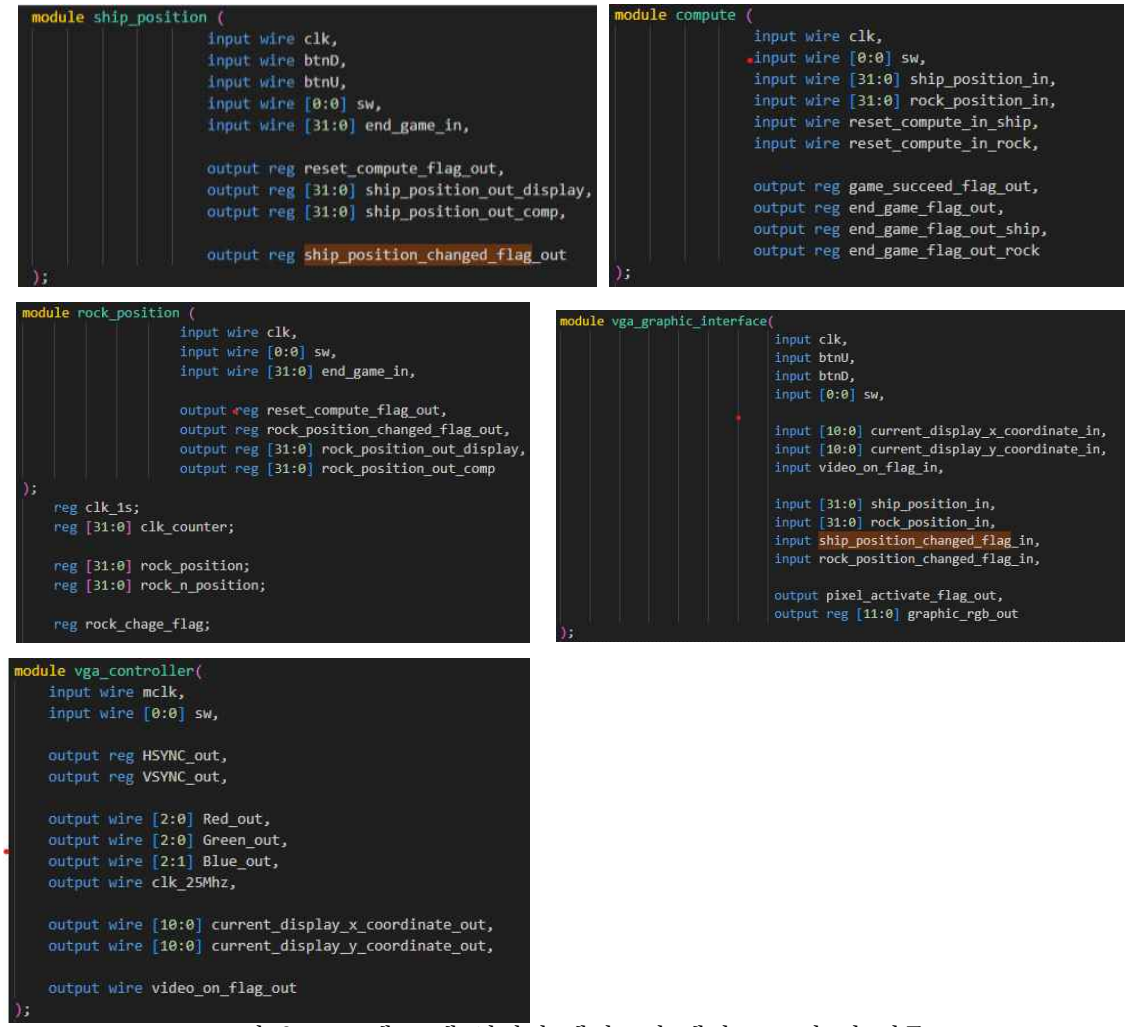
각각의 FSM state는 VGA를 통해 출력되는 디스플레이상 위치를 나타냅니다. FSM은 본 프로젝트의 비행기와 장애물체의 위치를 디스플레이에 나타내기 위해서 기준점이 되는 좌표가 됩니다. state를 기준으로 디스플레이의 좌표를 다음과 같이 만들었습니다. 그림 2에서 State의 방향 순서는 다음과 같습니다.

S2 -> S1 -> S0 -> S7 -> S6 -> S5 -> S4 -> S11 -> S10 -> S9 -> S8 -> S3 -> S2

그림 2 에서 State0의 디스플레이상 좌표 (X, Y) = (200, 100) 가 됩니다.

프로젝트 레지스터 데이터 크기 설정:

본 프로젝트는 실제 사용되는 시스템에서 32 비트 프로세서와 유사하게 구현하기 위해 레지스터의 크기를 32 비트로 하여 작성하였습니다.



```
module ship_position (
                    input wire clk,
                    input wire btnD,
                    input wire btnU,
                    input wire [0:0] sw,
                    input wire [31:0] end_game_in,

                    output reg reset_compute_flag_out,
                    output reg [31:0] ship_position_out_display,
                    output reg [31:0] ship_position_out_comp,

                    output reg ship_position_changed_flag_out
);
```

```
module compute (
                    input wire clk,
                    input wire [0:0] sw,
                    input wire [31:0] ship_position_in,
                    input wire [31:0] rock_position_in,
                    input wire reset_compute_in_ship,
                    input wire reset_compute_in_rock,

                    output reg game_succeed_flag_out,
                    output reg end_game_flag_out,
                    output reg end_game_flag_out_ship,
                    output reg end_game_flag_out_rock
);
```

```
module rock_position (
                    input wire clk,
                    input wire [0:0] sw,
                    input wire [31:0] end_game_in,

                    output reg reset_compute_flag_out,
                    output reg rock_position_changed_flag_out,
                    output reg [31:0] rock_position_out_display,
                    output reg [31:0] rock_position_out_comp
);
    reg clk_1s;
    reg [31:0] clk_counter;

    reg [31:0] rock_position;
    reg [31:0] rock_n_position;

    reg rock_chage_flag;
```

```
module vga_graphic_interface(
                    input clk,
                    input btnU,
                    input btnD,
                    input [0:0] sw,

                    input [10:0] current_display_x_coordinate_in,
                    input [10:0] current_display_y_coordinate_in,
                    input video_on_flag_in,

                    input [31:0] ship_position_in,
                    input [31:0] rock_position_in,
                    input ship_position_changed_flag_in,
                    input rock_position_changed_flag_in,

                    output pixel_activate_flag_out,
                    output reg [11:0] graphic_rgb_out
);
```

```
module vga_controller(
    input wire mclk,
    input wire [0:0] sw,

    output reg HSYNC_out,
    output reg VSYNC_out,

    output wire [2:0] Red_out,
    output wire [2:0] Green_out,
    output wire [2:1] Blue_out,
    output wire clk_25Mhz,

    output wire [10:0] current_display_x_coordinate_out,
    output wire [10:0] current_display_y_coordinate_out,

    output wire video_on_flag_out
);
```

그림 3. 프로젝트 내 선언된 레지스터 데이트 크기 및 이름

Flag 라고 적혀 있는 부분은 해당 데이터의 상태 정보를 나타내는 용도로 데이터의 크기를 1 비트로 하였습니다. ship_position_changed_flag의 값이 1인 경우 비행기의 위치 정보가 변경되었음을 알립니다.

프로젝트 데이터 이름 설정:

각각의 모듈에 모듈 내부로 들어오는 신호와 모듈 외부로 출력되는 신호를 직관적으로 코드를 통해 확인하기 위해 들어오는 신호에는 이름 뒤에 ship_position_changed_flag_in 과 같이 in 이라는 키워드를 붙여주었고, 출력 신호에는 pixel_activate_flag_out 과 같이 out 이라는 키워드를 붙였습니다.

프로젝트 주석 처리:

프로젝트 코드 디버깅 시 효율적인 분석을 위해 코드를 Context 별로 나누어서 코드를 덩이리 형태로 주석을 처리하였습니다.

비행기 생성 모듈 :

```
ship_position genergate_ship_position (
                                        .clk(clk),
                                        .btnD(btn_keyboard[1]),
                                        .btnU(btn_keyboard[0]),
                                        .sw(sw),
                                        .end_game_in(end_game_flag_endToship),

                                        .reset_compute_flag_out(reset_game_flag_shipToComp),
                                        .ship_position_out_display(ship_position_display),
                                        .ship_position_out_comp(ship_position_comp),

                                        .ship_position_changed_flag_out(ship_position_changed_flag)
);
```

```
    always@ (posedge clk) begin
        if(~sw ) begin
            ship_position <= 1'b0;
            reset_compute_flag_out <= 1'b1;
            ship_position_changed_flag_out <= 1'b0;
        end else begin
            reset_compute_flag_out <= 1'b0;
            ship_position_changed_flag_out <= 1'b0;

            if(btnR) begin
                if(ship_position < 5) begin
                    ship_position <= ship_position + 32'd4;
                    ship_position_changed_flag_out <= 1'b1;
                end else begin
                    ship_position <= ship_position;
                    ship_position_changed_flag_out <= 1'b0;
                end
            end else if(btnL) begin
                if(ship_position > 3) begin
                    ship_position <= ship_position - 32'd4;
                    ship_position_changed_flag_out <= 1'b1;
                end else begin
                    ship_position <= ship_position;
                    ship_position_changed_flag_out <= 1'b0;
                end
            end
        end
    end

    always @(ship_position) begin
        ship_position_out_comp <= ship_position;
        ship_position_out_display <= ship_position;
    end
endmodule
```

그림 4. 비행기 위치 생성 모듈

FPGA 외부 장치로부터 입력을 받아 비행기의 위치를 바꾸는 모듈입니다. 비행기가 총 세 번의 움직임으로 제한하기 위해서 외부 장치로부터 btnR인 오른쪽 입력을 받을 때, 기존에 계획했던 state_table에 의해 4씩 증가하다가 5보다 작은 경우까지 증가할 수 있도록 설정하였습니다. 이렇게 하면, ship_position은 state 0에서 시작하여 state 4를 거쳐 state 8에 도달할 수 있습니다. 또한, ship position이 state 8에 도달하면 더 이상 증가하지 않도록 하였습니다.

FPGA의 외부 장치로부터 btnL인 왼쪽 입력을 받을 때, 4씩 감소하도록 하였습니다. 이렇게 설정하면 ship_position은 state 8에서 state 4를 거쳐 state 0에 도달할 수 있습니다. 또한, ship position이 state 0에 도달하면 더 이상 감소하지 않도록 하였습니다.

ship_position 모듈의 출력은 VGA_Graphic_interface, compute 모듈과 연결되어 있어 비행기 위치 데이터와 비행기 위치 업데이트 여부를 확인할 수 있는 Flag 비트를 송신합니다.

그림 5. 비행기 state 변화

비행기 생성 코드:

```
module ship_position(
                  input wireclk,
                  input wirebtnD,
                  input wirebtnU,
                  input wire[0:0] sw,
                  input wire[31:0] end_game_in,
                  output regreset_compute_flag_out,
                  output reg[31:0] ship_position_out_display,
                  output reg[31:0] ship_position_out_comp,
                  output regship_position_changed_flag_out
);
    reg[31:0] ship_position;


    //*****************************************//
    //          CREATE ship position           //
    //*****************************************//


    always@ (posedgeclk) begin
        if(~sw ) begin
              ship_position <=1'b0;
              reset_compute_flag_out <=1'b1;
              ship_position_changed_flag_out <=1'b0;
        end else begin
              reset_compute_flag_out <=1'b0;
              ship_position_changed_flag_out <=1'b0;
            if(btnR) begin
                if(ship_position <5) begin
                      ship_position <=ship_position +32'd4;
                      ship_position_changed_flag_out <=1'b1;
                end else begin
                      ship_position <=ship_position;
                      ship_position_changed_flag_out <=1'b0;
                end
            end else if(btnL) begin
                if(ship_position >3) begin
                      ship_position <=ship_position -32'd4;
                      ship_position_changed_flag_out <=1'b1;
                end else begin
                      ship_position <=ship_position;
                      ship_position_changed_flag_out <=1'b0;
                end
            end
        end
    end
    always@(ship_position) begin
       ship_position_out_comp <=ship_position;
       ship_position_out_display <=ship_position;
    end
endmodule
```

장애물 위치 생성 모듈:

```
rock_position generate_rock_position (
                                    .clk(clk),
                                    .sw(sw),
                                    .end_game_in(end_game_flag_endToRock),

                                    .reset_compute_flag_out(reset_game_flag_rockToComp),
                                    .rock_position_out_comp(rock_position_comp),
                                    .rock_position_out_display(rock_position_display),

                                    .rock_position_changed_flag_out(rock_position_changed_flag)
);
```

```
always @(posedge clk) begin
    if(~sw) begin
        rock_position <= 1'b0;
        rock_n_position <= 2'b10;
        clk_count <= 32'd0;
        change <= 1'b0;
        // reset_compute_flag_out <= 1'b1;
        // rock_position_changed_flag_out <= 1'b0;
    end else begin
    // reset_compute_flag_out <= 1'b0;
    // rock_position_changed_flag_out <= 1'b0;

        rock_position <= rock_n_position;
        if (clk_count == 32'd100000000) begin
            clk_count <= 32'b0;
            change <= 1'b1;
        end else begin
            clk_count <= clk_count + 1'b1;
            change <= 1'b0;
        end

        if(change == 1'b1 ) begin
        case(rock_position)
            32'd0: rock_n_position = 32'd7;
            32'd1: rock_n_position = 32'd0;
            32'd2: rock_n_position = 32'd1;
            32'd3: rock_n_position = 32'd2;
            32'd4: rock_n_position = 32'd11;
            32'd5: rock_n_position = 32'd4;
            32'd6: rock_n_position = 32'd5;
            32'd7: rock_n_position = 32'd6;
            32'd8: rock_n_position = 32'd3;
            32'd9: rock_n_position = 32'd8;
            32'd10: rock_n_position = 32'd9;
            32'd11: rock_n_position = 32'd10;
            default: rock_n_position = 32'd7;
        endcase
    end else begin
        rock_n_position <= rock_position;   //state isn't changed
    end
end
end
```

그림 5. 장애물 위치 정보 생성 모듈

1초를 클락 주기로 하여 change 값을 변경하고 change 값이 1이면, 장애물의 state를 업데이트 하는 코드를 작성하였습니다. FSM 구조로 만들어 다음 state의 장애물 위치를 입력하고 클락에 따라 현재 장애물 위치의 state를 업데이트하도록 작성하였습니다.

본 코드에서 장애물의 위치는 다음과 같은 순서로 state가 업데이트 됩니다.

S0 -> S2 -> S1 -> S0 -> S7 -> S6 -> S5 -> S4 -> S11 -> S10 -> S9 -> S8 -> S3 -> S2 로 반복됩니다.

본 코드의 출력은 reset_compute, rock_position, rock_position_changed_flag 로 장애물이 업데이트 되었는지 여부를 다른 모듈로 전송하고, 업데이트된 장애물의 위치를 출력으로 합니다. reset_compute는 compute 모듈을 초기화하기 위함입니다.

rock_position 모듈의 출력은 VGA_Graphic_interface 와 compute 모듈의 입력과 연결되어 있어 장애물의 위치 정보와 장애물의 상태 정보 업데이트 여부를 알리는 Flag 비트를 송신합니다. rock_postion 모듈의 입력은 FPGA가 물리적으로 생성하는 100MHz 클락과 모듈 내 선언된 레지스터 값을 초기화할 수 있는 0번 스위치와 eng_game 모듈로부터 게임의 상태 정보를 확인할 수 있는 Flag 비트를 수신받습니다.

장애물 위치 생성 코드:

```
module rock_position(
                    input wire clk,
                    input wire [0:0] sw,
                    input wire [31:0] end_game_in,

                    output reg reset_compute_flag_out,
                    output reg rock_position_changed_flag_out,
                    output reg [31:0] rock_position_out_display,
                    output reg [31:0] rock_position_out_comp
);
    reg clk_1s;
    reg [31:0] clk_counter;
    reg [31:0] rock_position;
    reg [31:0] rock_n_position;
    reg rock_chage_flag;

    //*********************************************//
    //              CREAT ROCK POSITION            //
    //*********************************************//

    reg change;
    reg [31:0] clk_count;
    always@(posedge clk) begin
        if(~sw) begin
              rock_position <=1'b0;
              rock_n_position <=2'b10;
              clk_count <=32'd0;
              change <=1'b0;
           // reset_compute_flag_out <= 1'b1;
           // rock_position_changed_flag_out <= 1'b0;
        end else begin
        // reset_compute_flag_out <= 1'b0;
        // rock_position_changed_flag_out <= 1'b0;

              rock_position <=rock_n_position;
              if(clk_count ==32'd100000000) begin
                    clk_count <=32'b0;
                    change <=1'b1;
              end else begin
                    clk_count <=clk_count +1'b1;
                    change <=1'b0;
            end

            if(change ==1'b1begin
            case(rock_position)
                32'd0: rock_n_position =32'd7;
                32'd1: rock_n_position =32'd0;
                32'd2: rock_n_position =32'd1;
                32'd3: rock_n_position =32'd2;
                32'd4: rock_n_position =32'd11;
                32'd5: rock_n_position =32'd4;
```

```
                32'd6: rock_n_position =32'd5;
                32'd7: rock_n_position =32'd6;
                32'd8: rock_n_position =32'd3;
                32'd9: rock_n_position =32'd8;
                32'd10: rock_n_position =32'd9;
                32'd11: rock_n_position =32'd10;
                default: rock_n_position =32'd7;
            endcase
        end else begin
               rock_n_position <=rock_position;   //state isn't changed
        end
    end
end



    always@(posedge clk) begin
       rock_position_out_comp <=rock_position;
       rock_position_out_display <=rock_position;
    end
endmodule
```

비행기 및 장애물 위치 계산 모듈:

```
compute generate_compute (
                        .clk(clk),
                        .sw(sw),
                        .ship_position_in(ship_position_comp),
                        .rock_position_in(rock_position_comp),
                        .reset_compute_in_ship(reset_game_flag_shipToComp),
                        .reset_compute_in_rock(reset_game_flag_rockToComp),

                        .game_succeed_flag_out(game_succeed_flag),
                        .end_game_flag_out(end_game_flag),
                        .end_game_flag_out_ship(end_game_flag_endToship),
                        .end_game_flag_out_rock(end_game_flag_endToRock)
);
```

```
always @(posedge clk) begin
    if(~sw | reset_compute_in_ship | reset_compute_in_rock) begin
        end_game_flag_out <= 1'b0;
        end_game_flag_out_ship <= 1'b0;
        end_game_flag_out_rock <= 1'b0;
        game_succeed_flag_out <= 1'b0;
    end else begin
        if(ship_position_in == rock_position_in) begin
            // game failed
            end_game_flag_out <= 1'b1;
            end_game_flag_out_ship <= 1'b1;
            end_game_flag_out_rock <=1'b1;
            game_succeed_flag_out <= 1'b0;
        end else if (rock_position_in == 32'd0 | rock_position_in == 32'd4 | rock_position_in | 32'd8) begin
            // game succeed
            game_succeed_flag_out <= 1'b1;
            end_game_flag_out <= 1'b0;
            end_game_flag_out_ship <= 1'b0;
            end_game_flag_out_rock <=1'b0;
        end else begin
            // game in progress
            game_succeed_flag_out <= 1'b0;
            end_game_flag_out <= 1'b0;
            end_game_flag_out_ship <= 1'b0;
            end_game_flag_out_rock <=1'b0;
        end
    end
end
```

그림 6. 비행기 및 장애물 위치 계산 모듈

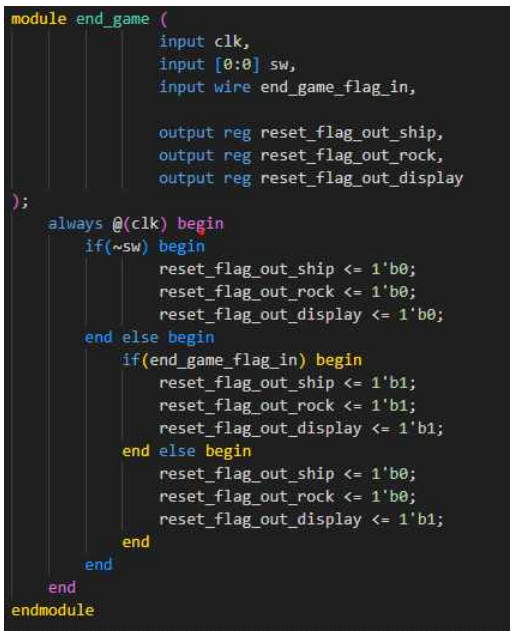
비행기의 위치 정보를 생성하는 ship_position 모듈과 장애물의 위치 정보를 생성하는 rock_position 모듈에서 생성되는 각각의 물체 위치 정보를 수신받아 비행기의 위치 정보와 장애물의 정보 위치가 같은 경우 게임 실패라고 정의하였고, 비행기의 위치와 장애물 위치가 같지 않은 경우 비행기가 물체에 충돌하지 않았다고 가정하여 게임 성공이라고 정의하였습니다. 위 두가지 경우가 아닌 경우에는 게임은 실패도 성공도 아닌 진행중에 있다고 정의하였습니다.

게임 실패 시 end_game_flag를 생성하여 모듈 외부로 게임 종료를 알립니다.

비행기 및 장애물 위치 계산 모듈 코드:

```
module compute(
                input wire clk,
                input wire [0:0] sw,
                input wire [31:0] ship_position_in,
                input wire [31:0] rock_position_in,
                input wire reset_compute_in_ship,
                input wire reset_compute_in_rock,

                output reg game_succeed_flag_out,
                output reg end_game_flag_out,
                output reg end_game_flag_out_ship,
                output reg end_game_flag_out_rock
);
    always@(posedge clk) begin
        if(~sw |reset_compute_in_ship |reset_compute_in_rock) begin
               end_game_flag_out <=1'b0;
               end_game_flag_out_ship <=1'b0;
               end_game_flag_out_rock <=1'b0;
               game_succeed_flag_out <=1'b0;
        end else begin
            if(ship_position_in ==rock_position_in) begin
                // game failed
               end_game_flag_out <=1'b1;
               end_game_flag_out_ship <=1'b1;
               end_game_flag_out_rock <=1'b1;
               game_succeed_flag_out <=1'b0;
            end else if(rock_position_in ==32'd0 |rock_position_in ==32'd4
|rock_position_in |32'd8) begin
                // game succeed
               game_succeed_flag_out <=1'b1;
               end_game_flag_out <=1'b0;
               end_game_flag_out_ship <=1'b0;
               end_game_flag_out_rock <=1'b0;
            end else begin
                // game in progress
               game_succeed_flag_out <=1'b0;
               end_game_flag_out <=1'b0;
               end_game_flag_out_ship <=1'b0;
               end_game_flag_out_rock <=1'b0;
            end
        end
    end
endmodule
```

게임 종료 모듈:



```
module end_game (
                input clk,
                input [0:0] sw,
                input wire end_game_flag_in,

                output reg reset_flag_out_ship,
                output reg reset_flag_out_rock,
                output reg reset_flag_out_display
);
    always @(clk) begin
        if(~sw) begin
                reset_flag_out_ship <= 1'b0;
                reset_flag_out_rock <= 1'b0;
                reset_flag_out_display <= 1'b0;
        end else begin
            if(end_game_flag_in) begin
                reset_flag_out_ship <= 1'b1;
                reset_flag_out_rock <= 1'b1;
                reset_flag_out_display <= 1'b1;
            end else begin
                reset_flag_out_ship <= 1'b0;
                reset_flag_out_rock <= 1'b0;
                reset_flag_out_display <= 1'b1;
            end
        end
    end
endmodule
```

그림 7. 게임 종료 모듈

비행기 및 장애물 위치 계산 모듈로부터 end_game_flag 값을 받으면 각각의 모듈 ship_position, rock_position, 디스플레이 관련 모듈에 게임 종료로 인한 모듈 초기화를 위한 reset flag 신호를 송신합니다.

eng game 이라는 모듈을 별도로 작성한 이유는 모듈의 사용 용도를 분리하기 위함입니다. end_game_flag 신호를 수신하는 compute 모듈에서 본 모듈의 기능을 구현할 수 있지만, compute 모듈은 단순한 계산만을 위해 존재하는 모듈로 보존하고 싶어 end_game 모듈을 별도로 작성하였습니다.

게임 종료 모듈 코드:

```
module end_game(
                input clk,
                input [0:0] sw,
                input wire end_game_flag_in,
                output reg reset_flag_out_ship,
                output reg reset_flag_out_rock,
                output reg reset_flag_out_display
);
    always@(posedge clk) begin
        if(~sw) begin
              reset_flag_out_ship <=1'b0;
              reset_flag_out_rock <=1'b0;
              reset_flag_out_display <=1'b0;
        end else begin
            if(end_game_flag_in) begin
              reset_flag_out_ship <=1'b1;
              reset_flag_out_rock <=1'b1;
              reset_flag_out_display <=1'b1;
            end else begin
              reset_flag_out_ship <=1'b0;
              reset_flag_out_rock <=1'b0;
              reset_flag_out_display <=1'b1;
            end
        end
    end
endmodule
```

VGA Controller:

```
vga_controller generate_vga(
                            .mclk(clk),
                            .sw(sw),

                            .HSYNC_out(HSYNC),
                            .VSYNC_out(VSYNC),
                            .Red_out(),
                            .Green_out(),
                            .Blue_out(),
                            .clk_25Mhz(clk_25Mhz),

                            .current_display_x_coordinate_out(display_x),
                            .current_display_y_coordinate_out(display_y),

                            .video_on_flag_out(video_on_flag)
);
```

VGA는 25MHz의 클럭을 기본으로 하여 클럭마다 1씩 숫자가 올라가는 카운터기를 중심으로 제어가 됩니다.

```
//****************************************//
//              Create FSM counter        //
//****************************************//

reg [10:0] H_count;
reg [10:0] V_count;

reg [10:0] new_H_count;
reg [10:0] new_V_count;

reg [10:0] H_count_next;
reg [10:0] V_count_next;


always @(posedge mclk) begin
    if(~sw) begin
        H_count <= 1'b0;
        V_count <= 1'b0;

        new_H_count <= 1'b0;
        new_V_count <= 1'b0;
    end else begin
        H_count <= H_count_next;
        V_count <= V_count_next;

        new_H_count <= H_count_next;
        new_V_count <= V_count_next;
    end

end
```

```
//****************************************//
//         CREATE Horizontal PIXEL        //
//****************************************//
/*
// if end of the monitor pixel reached, send image.

reg [10:0] H_image_count;
reg H_image_flag;

// create horizontal count
always @(posedge clk_25Mhz) begin
    if(~sw) begin
        H_image_count <= 11'b0;
    end else begin
        if (H_count < H_SYNC_CYC + H_SYNC_BACK) begin
            H_image_count <= 11'b0;
        end else if (H_count >= H_SYNC_CYC + H_SYNC_BACK + H_SYNC_ACT) begin
            H_image_count <= 11'b0;
        end else begin
            H_image_count <= H_image_count + 1'b1;
        end
    end
end

// create horizontal send bit
always @(posedge clk_25Mhz) begin
    if(~sw) begin
        H_image_flag <= 1'b0;
    end else begin
        if( H_count < H_SYNC_CYC + H_SYNC_BACK) begin
            H_image_flag <= 1'b0;
        end else if ( H_count >= H_SYNC_CYC + H_SYNC_BACK + H_SYNC_ACT ) begin
            H_image_flag <= 1'b0;
        end else begin
            H_image_flag <= 1'b1;
        end
    end
end

*/
```

그림 8. VGA Controller

VGA Controller 모듈은 디스플레이 사용을 위한 초기 설정을 하는 용도로 모듈을 사용하기 위해 제어식은 본 모듈에서 정의하지 않았습니다. 디스플레이 제어는 다른 모듈에서 할 수 있도록 하기 위해서 카운터기를 외부로 출력하도록 설계하였습니다. 본 모듈의 25MHz 클록으로 값이 변경되는 H_count 에 여러 개의 wire를 연결하게 되면 multiple drivers 문제가 발생하여 new_H_count와 new_V_count를 별도로 선언하여 외부 출력으로 만들었습니다.

본 VGA Controller 모듈 코드는 김현진 교수님의 VGA 동작 방식 코드를 응용한 코드로서 주석 처리된 Horiontal Pixel을 생성하는 코드 방식을 분석하여 H_Count에 따른 데이터 변환 과정을 분석하여 Count 값에 따른 RGB 출력 방식으로 디스플레이에 표시되는 것을 확인하였습니다.

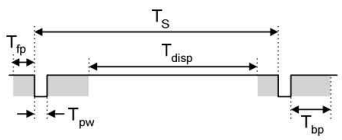
```
//*************************************//
//          CREATE H_SYNC CLK          //
//*************************************//


always @(posedge clk_25Mhz) begin
    if(~sw) begin
        H_count_next <= 11'b0;
        HSYNC_out <= 1'b1;
    end else begin
        if (H_count < H_SYNC_TOTAL) begin
            H_count_next <= H_count + 1'b1;
        end else begin
            H_count_next <= 1'b0;
        end
    end

    if(H_count < H_SYNC_CYC) begin
        HSYNC_out <= 1'b0;
    end else begin
        HSYNC_out <= 1'b1;
    end
end
```

```
//*************************************//
//          CREATE V_SYNC CLK          //
//*************************************//

always @(posedge clk_25Mhz) begin
    if(~sw) begin
        V_count_next <= 11'b0;
        VSYNC_out <= 1'b1;
    end else begin
        if(H_count == 11'b0) begin
            if(V_count < V_SYNC_TOTAL) begin
                V_count_next <= V_count + 1'b1;
            end else begin
                V_count_next <= 11'b0;
            end

            if(V_count < V_SYNC_CYC) begin
                VSYNC_out <= 1'b0;
            end else begin
                VSYNC_out <= 1'b1;
            end
        end
    end
end
```

그림 9. VGA Controller

정의된 H_SYNC_TOTAL 은 H_SYNC_CYC, H_SYNC_BACK, H_SYNC_ACT, H_SYNC_FRONT, 의 합으로 디지털 영상 신호에서 한 수평 주기의 총 길의를 나타냅니다. 여기서 H_SYCNC_CYC는 수평 동기 신호의 길이이며, H_SYNC_BACK은 수평 동기 신호가 끝난 후에 활성 데이터가 시작되기 전까지의 Back porch 구간입니다. H_SYSNC_ACT는 실제 화면에 표시되는 활성 데이터 구간입니다. H_SYNCT_FRONT는 활성 데이터가 끝난 후 다음 수평 동기 신호가 시작되기 전까지의 Front porch 구간을 의미합니다.

수평값 H_SYNC를 만들어준 것 같이 V_SYNC도 같은 방식으로 작성하여 사용하였습니다.



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

그림 10. VGA Control Reference Manual

Basys3 reference manual에 적혀 있는 값은 H_SYNC_CYC = 96, H_SYNC_ACT = 640, H_SYNC_BACK = 48, H_SYNC_FRONT = 16을 유의해서 사용하였습니다.

```
assign video_on_flag_out = (H_count < 640) && (V_count < 480); // 0-639 and 0-479 respectively

assign current_display_x_coordinate_out = new_H_count;
assign current_display_y_coordinate_out = new_V_count;
```

디스플레이에 입력할 데이터 위치는 외부 모듈에서 사용되도록 출력 wire 로 설계하였습니다.

VGA Graphic Interface:

```
// create ship and rock position coordinate
always @(posedge clk) begin
    if(~sw) begin
        x_ship_position_coordinate <= 1'b0;
        y_ship_position_coordinate <= 1'b0;
    end else begin
            case(ship_position_in)
            32'd0 : x_ship_position_coordinate = 32'd200;  // (200, 100)
            32'd4 : x_ship_position_coordinate = 32'd400;  // (400, 100)
            32'd8 : x_ship_position_coordinate = 32'd600;  // (600, 100)
            endcase

            y_ship_position_coordinate <= 32'd100;

            case(rock_position_in)
            32'd3 : begin x_rock_position_coordinate = 32'd200; y_rock_position_coordinate = 32'd400; end   // (200, 400)
            32'd2 : begin x_rock_position_coordinate = 32'd200; y_rock_position_coordinate = 32'd300; end   // (200, 300)
            32'd1 : begin x_rock_position_coordinate = 32'd200; y_rock_position_coordinate = 32'd200; end   // (200, 200)
            32'd0 : begin x_rock_position_coordinate = 32'd200; y_rock_position_coordinate = 32'd100; end   // (200, 100)
            32'd7 : begin x_rock_position_coordinate = 32'd400; y_rock_position_coordinate = 32'd400; end   // (400, 400)
            32'd6 : begin x_rock_position_coordinate = 32'd400; y_rock_position_coordinate = 32'd300; end   // (400, 300)
            32'd5 : begin x_rock_position_coordinate = 32'd400; y_rock_position_coordinate = 32'd200; end   // (400, 200)
            32'd4 : begin x_rock_position_coordinate = 32'd400; y_rock_position_coordinate = 32'd100; end   // (400, 100)
            32'd11 : begin x_rock_position_coordinate = 32'd600; y_rock_position_coordinate = 32'd400; end  // (600, 400)
            32'd10 : begin x_rock_position_coordinate = 32'd600; y_rock_position_coordinate = 32'd300; end  // (600, 300)
            32'd9 : begin x_rock_position_coordinate = 32'd600; y_rock_position_coordinate = 32'd200; end   // (600, 200)
            32'd8 : begin x_rock_position_coordinate = 32'd600; y_rock_position_coordinate = 32'd100; end   // (600, 100)
            endcase
        end
    end
```

그림 11. VGA Graphic Interface

VGA Graphic Interface 모듈은 ship_position 모듈과 rock_position 모듈로부터 나온 비행기의 state 값과 장애물의 state 값을 이용하여 디스플레이에 표시되도록 좌표를 할당하였습니다. X 최대 크기 640에서 사이드에 여유를 만들기 위해서 40을 빼고 3분할 하여 각각 200, 400, 600을 X 좌표로 할당하였습니다. Y 최대 크기 480에서 사이드에 여유를 만들기 위해서 80을 빼고 4분할 하여 각각 100, 200, 300, 400을 Y 좌표로 할당하였습니다. 다음 그림은 state에 따른 디스플레이 위치입니다.

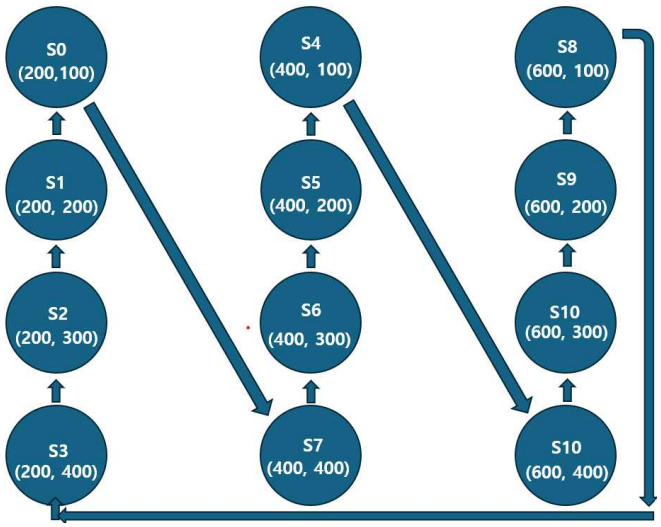


그림 12. 장애물 FSM state 에 따른 디스플레이 위치

비행기의 state는 0, 4, 8 로만 할당이 가능하여 Y의 값은 고정하고 X 좌표만 S0일 때 200, S4일 때 400, S8일 때 600으로 할당하였습니다. 장애물의 경우에는 S3에서 시간에 따라 순차적으로 일직선으로 비행기에 도달하도록 하였습니다.

```
//*****************************************//
//          CREATE SHIP Graphic            //
//*****************************************//

wire [9:0] x_ship_left_coordinate;
wire [9:0] x_ship_right_coordinate;
wire [9:0] y_ship_top_coordinate;
wire [9:0] y_ship_bottom_coordinate;

wire current_display_in_ship_coordinate;

parameter MAX_SHIP_SIZE = 8;

// referenced position top left edge
assign x_ship_left_coordinate = x_ship_position_coordinate[9:0];
assign y_ship_top_coordinate = y_ship_position_coordinate[9:0];
assign x_ship_right_coordinate = x_ship_left_coordinate + MAX_SHIP_SIZE - 1;
assign y_ship_bottom_coordinate = y_ship_top_coordinate + MAX_SHIP_SIZE - 1;

assign current_display_in_ship_coordinate = (x_ship_left_coordinate <= x) &&
                                            (x <= x_ship_right_coordinate) &&
                                            (y_ship_top_coordinate <= y) &&
                                            (y <= y_ship_bottom_coordinate);
```

그림 13. VGA Graphic Interface

비행기의 좌측 상단을 중심으로 하는 좌표를 이용하여 사전에 정의한 비행기의 그래픽 디자인을 출력하도록 하는 코드입니다. 좌측 상단에는 VGA Graphic Interface에 들어온 비행기 state를 좌표로 변환하여 저장합니다. 아래 그림과 같이 비행기의 기준 좌표가 현재 디스플레이가 출력해야 하는 픽셀 범위 안에 있으면 current_display_in_ship_coordinate 값은 1이 됩니다.

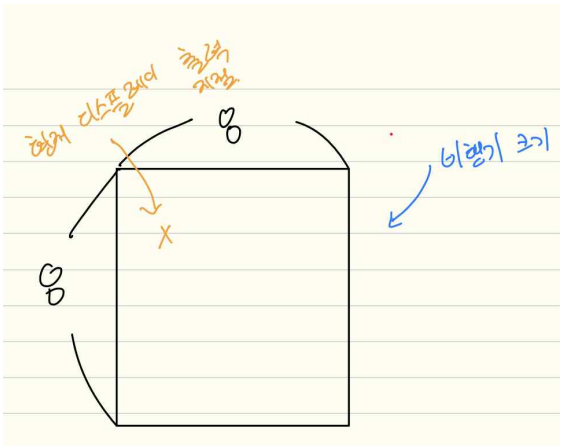
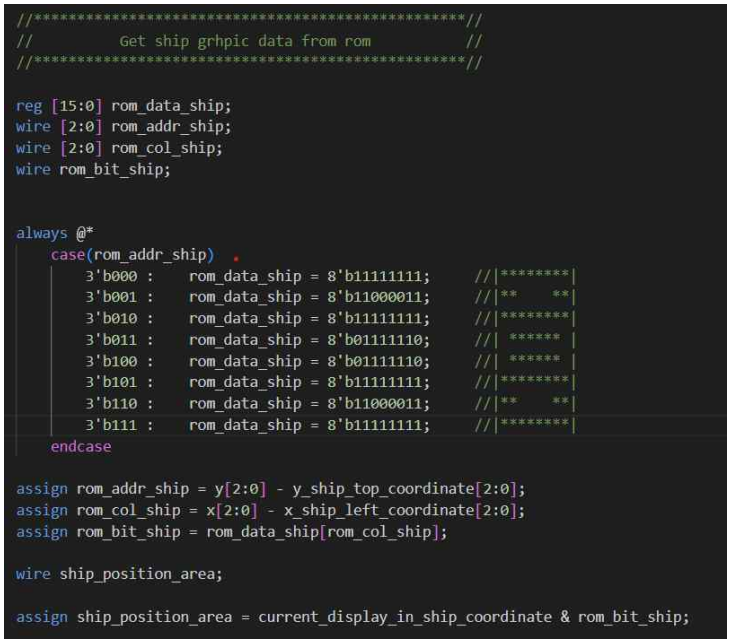


그림 14. 비행기 좌표에 따른 디스플레이 인식

현재 외부로 출력해야 하는 X 좌표 위치가 비행기 크기 영역에 들어가 있으면 해당 위치는 비행기 그래픽으로 출력되어야 합니다.



```
//*********************************************//
//         Get ship grhpic data from rom       //
//*********************************************//

reg [15:0] rom_data_ship;
wire [2:0] rom_addr_ship;
wire [2:0] rom_col_ship;
wire rom_bit_ship;


always @*
    case(rom_addr_ship)
        3'b000 :    rom_data_ship = 8'b11111111;     //|********|
        3'b001 :    rom_data_ship = 8'b11000011;     //|**    **|
        3'b010 :    rom_data_ship = 8'b11111111;     //|********|
        3'b011 :    rom_data_ship = 8'b01111110;     //| ****** |
        3'b100 :    rom_data_ship = 8'b01111110;     //| ****** |
        3'b101 :    rom_data_ship = 8'b11111111;     //|********|
        3'b110 :    rom_data_ship = 8'b11000011;     //|**    **|
        3'b111 :    rom_data_ship = 8'b11111111;     //|********|
    endcase

assign rom_addr_ship = y[2:0] - y_ship_top_coordinate[2:0];
assign rom_col_ship = x[2:0] - x_ship_left_coordinate[2:0];
assign rom_bit_ship = rom_data_ship[rom_col_ship];

wire ship_position_area;

assign ship_position_area = current_display_in_ship_coordinate & rom_bit_ship;
```

그림 15. 비행기 그래픽 이미지

Rom을 이용하여 비행기의 좌표가 저장되어 해당 위치에 X 값이 도달하면 출력되도록 코드를 작성하였습니다. 해당 코드는 직접 작성한 것이 아니라 오픈소스를 활용하여 메모리 구조를 작성한 방식에 대해 학습하고 디스플레이에 원하는 형태의 그래픽 디자인이 출력되도록 변형한 코드입니다. 해당 픽셀은 8x8 픽셀로 화면상에 출력하는 경우 화면 비율에 비해 작아 개선이 필요합니다. 이를 개선하기 위해 기존 8비트 크기에서 16비트를 크기를 스케일링하여 코드 재작성이 필요합니다.

해당 Rom은 행이 메모리의 주소가 되고, 열이 메모리의 데이터의 비트 위치로 하여 데이터를 사용합니다. 현재 디스플레이에 출력하고자 하는 위치의 y 값에서 비행기의 y 값 좌표를 빼주면 Rom에 저장된 데이터의 위치를 가져올 수 있습니다. 예를 들어 현재 X, Y의 좌표가 (12, 7)이고 배의 상단 좌표가 (10, 5) 일 때, rom_addr_ship의 값은 2가 됩니다. 이를 기반으로 Rom에 저장되어 있는 2번째 행 데이터를 반환받게 됩니다.

```
wire ship_position_area;

assign ship_position_area = current_display_in_ship_coordinate & rom_bit_ship;
```

그림 16. 디스플레이에서 비행기 위치 인식

현재 출력하는 좌표가 비행기 크기 안에 있으면서 Rom 내 존재하는 데이터값 중 0이라고 적혀 있는 부분에는 Background 색이 출력되도록 만들기 위한 코드입니다. and 연산자를 통해 두 개의 비트가 모두 1인 경우에만 ship_position_area 값이 1이 되어 비행기의 색이 출력됩니다.

비행기의 디스플레이 출력 여부를 계산한 것과 같이 장애물의 좌표도 구해서 다음과 같이 rock_position_area 값을 구합니다.

```
wire rock_position_area;

assign rock_position_area = current_display_in_rock_coordinate & rom_bit_rock;
```

그림 17. 디스플레이에서 장애물 위치 인식

```
// choose color of pixel of diplay
always @(posedge clk) begin
    if(~video_on_flag_in) begin
        graphic_rgb_out = 12'h000;
    end else begin
        if(ship_position_area) begin
            graphic_rgb_out = ship_rgb_color;
        end else if (rock_position_area) begin
            graphic_rgb_out <= rock_rgb_color;
        end else begin
            graphic_rgb_out = background_rgb_color;
        end
    end
end
```

그림 18. 디스플레이에서 현재 픽셀 위치에 따른 색상 정보 결정

구해진 rock_position_area flag 값과 ship_position_area flag 값을 이용해서 현재 출력하고자 하는 위치가 비행기가 출력되어야 하는 위치라고 하면, 비행기의 색상 정보를 출력하고 장애물이 출력되어야 하는 위치라고 하면, 장애물의 색상 정보를 출력합니다. 만약 해당 위치가 비행기와 장애물 모두 아니라고 하면, 배경 화면 색상을 출력하도록 합니다.

```
wire [11:0] ship_rgb_color = 12'hFFF;       // White
wire [11:0] rock_rgb_color = 12'hFFF;       // White
wire [11:0] background_rgb_color = 12'h000; // Black
```

그림 19. 배경과 물체의 색상 정보

색상 정보는 12비트 RGB 형태로 RGB 모두 F로 할당하면 하얀색이 출력되고, 모두 0으로 하면 검정색이 출력됩니다. 12'hFFF에서 각각의 위치는 RGB로 0부터 F까지 빛의 세기를 나타냅니다. 12'h0FF으로 할당되는 경우에는 해당 픽셀의 빨간색은 꺼버리고, 초록색과 파란색만 최대 밝기로 하여 픽셀을 나타냅니다.

XDC 파일 설정시, 12 비트 RGB 값으로 선언하여 12'hFFF 라고 하였을 때 R값이 F이고, G 값이 F 이고, B 값이 F가 되도록 하였습니다.

```
set_property PACKAGE_PIN G19 [get_ports {display_rgb[11]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[11]}]
set_property PACKAGE_PIN H19 [get_ports {display_rgb[10]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[10]}]
set_property PACKAGE_PIN J19 [get_ports {display_rgb[9]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[9]}]
set_property PACKAGE_PIN N19 [get_ports {display_rgb[8]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[8]}]
set_property PACKAGE_PIN N18 [get_ports {display_rgb[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[7]}]
set_property PACKAGE_PIN L18 [get_ports {display_rgb[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[6]}]
set_property PACKAGE_PIN K18 [get_ports {display_rgb[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[5]}]
set_property PACKAGE_PIN J18 [get_ports {display_rgb[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[4]}]
set_property PACKAGE_PIN J17 [get_ports {display_rgb[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[3]}]
set_property PACKAGE_PIN H17 [get_ports {display_rgb[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[2]}]
set_property PACKAGE_PIN G17 [get_ports {display_rgb[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[1]}]
set_property PACKAGE_PIN D17 [get_ports {display_rgb[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {display_rgb[0]}]
set_property PACKAGE_PIN P19 [get_ports HSYNC]
    set_property IOSTANDARD LVCMOS33 [get_ports HSYNC]
set_property PACKAGE_PIN R19 [get_ports VSYNC]
    set_property IOSTANDARD LVCMOS33 [get_ports VSYNC]
```

그림 20. XDC VGA 포트 선언 방식

디스플레이는 VGA Controller에서 25Mhz 클럭에 맞춰 디스플레이 왼쪽 상단부부터 오른쪽으로 순차적으로 하나씩 움직이면서 픽셀의 색상을 결정할 수 있도록 합니다.

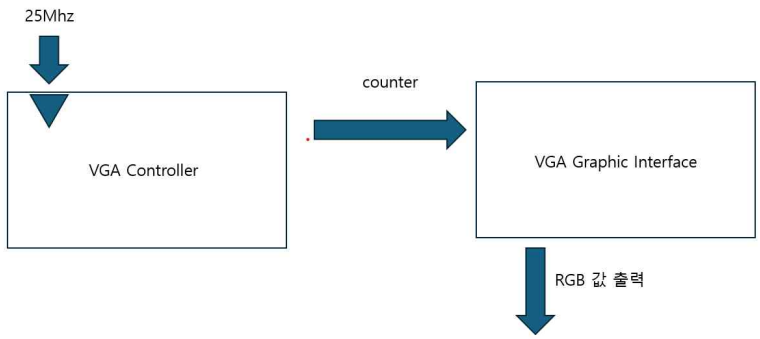


그림 21. VGA 연결 구성도

VGA Controller로부터 생성된 counter를 입력으로 하여 픽셀 위치를 확인하고 RGB 값을 출력합니다.

```
//*****************************************//
//                 FINAL VIDEO             //
//*****************************************//

module generate_video(
                    input wire clk,
                    input wire [0:0]sw,
                    input wire [11:0] video_data_in,
                    input wire clk_25Mhz,

                    output reg [11:0] rgb_out
);
    reg [11:0] rgb_next_state;

    always @(video_data_in) begin
        rgb_next_state <= video_data_in;
    end


    always @(posedge clk) begin
        if(~sw) begin
            rgb_out <= 1'b0;
        end else begin
            if (clk_25Mhz) begin
                rgb_out <= rgb_next_state;
            end
        end
    end


endmodule
```

그림 22. VGA 최종 출력 모듈

VGA Graphic Interface 모듈로부터 생성된 RGB 데이터 정보를 video_data로 입력 받아 FSM을 통해 다음 주기에 25MHz에 맞춰 RGB 값을 업데이트하도록 설계하였습니다.