Screaming Bird

Yiran Hu, Yuesheng Ma, Chenyang Zhou,Yang Li

Flappy Bird Mechanics

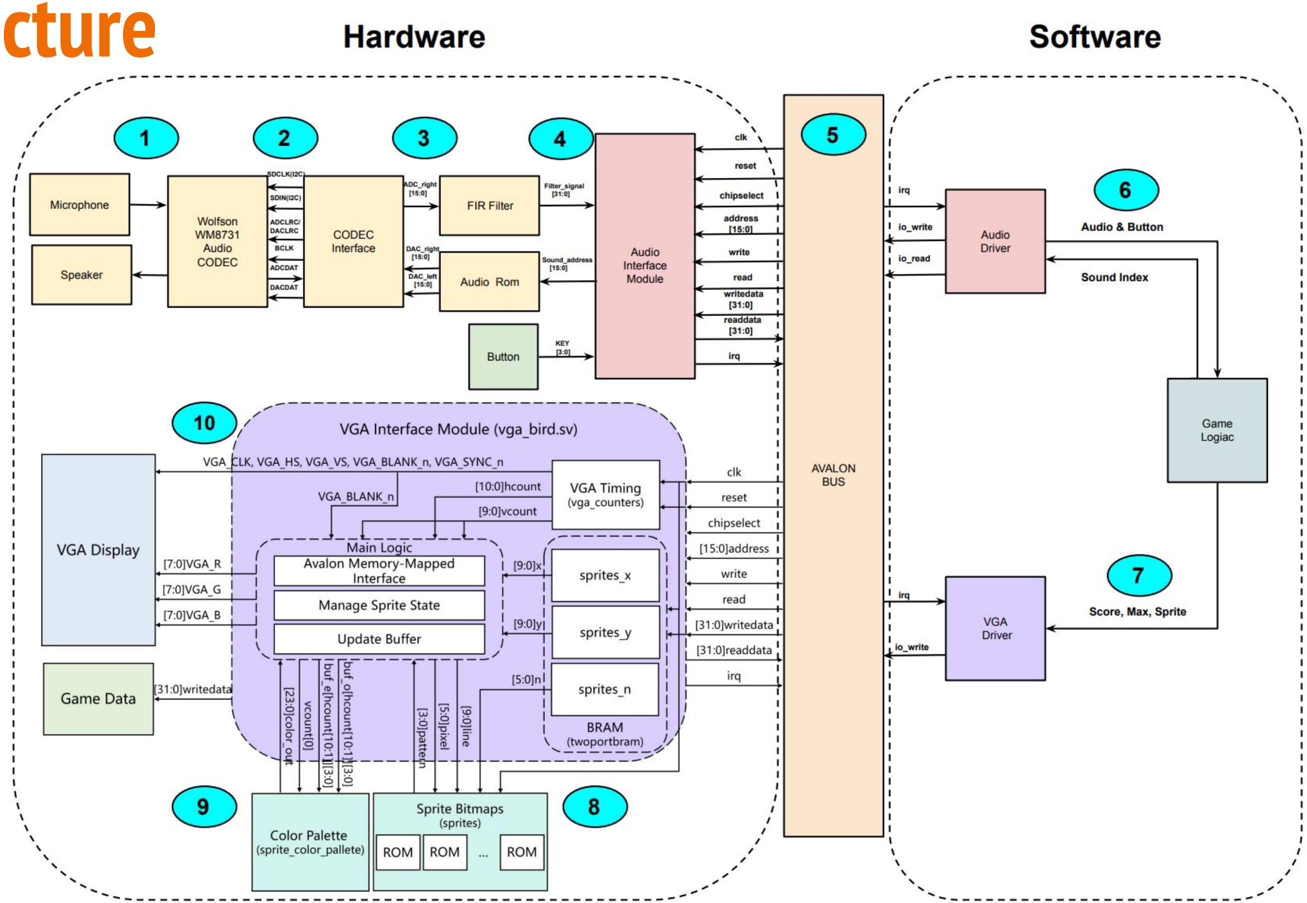
Game Mechanics:

- User: Button, Microphone
- Hardware: VGA controller and audio controller

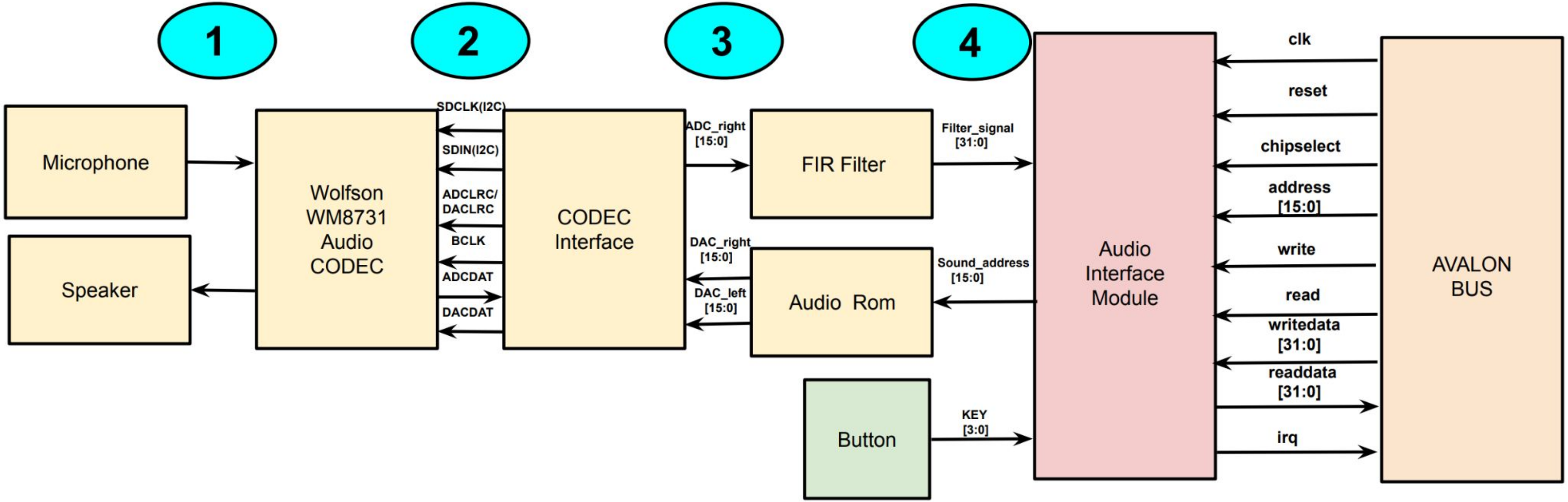
Game Logic:

- Press the key to Start Game.
- Press the key to set the difficulty level of the game: EASY, MEDIUM, HARD.
- Use the Microphone to flap the bird. If no audio is detected, the bird will keep falling due to gravity.
- Control the bird through pipes with random heights.
- Game Over detection occurs if the bird reaches the ground or collides with a pipe.
- Fly as far as possible to get the highest score.

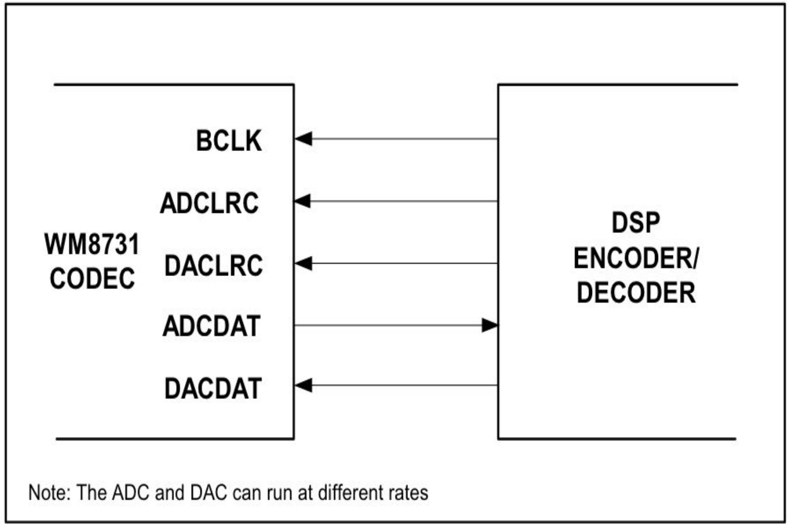
Architecture

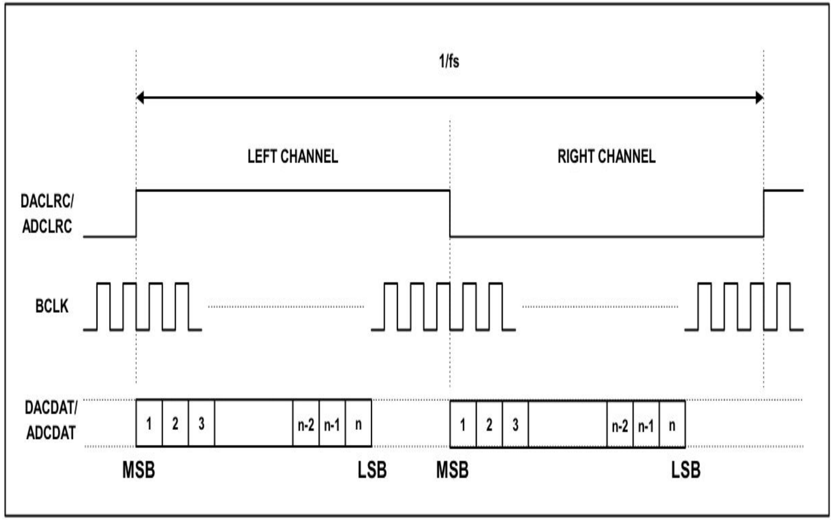


Audio Module

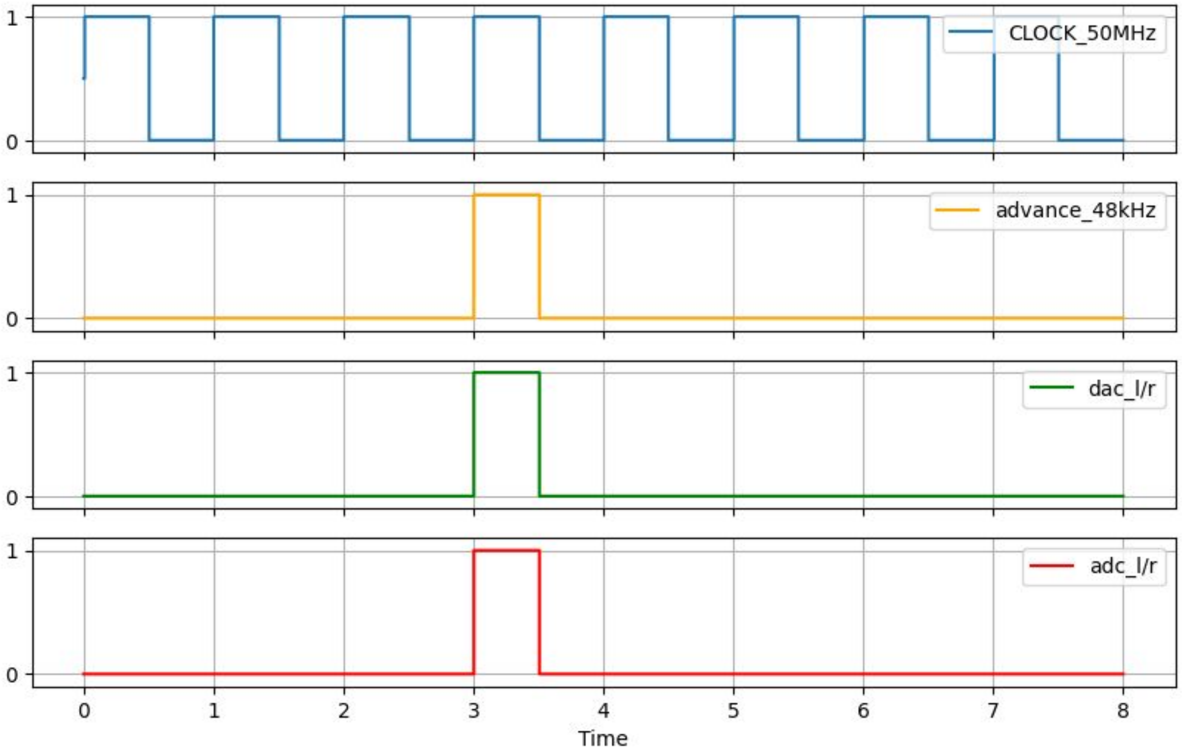


CODEC Interface IP





CODEC Interface IP

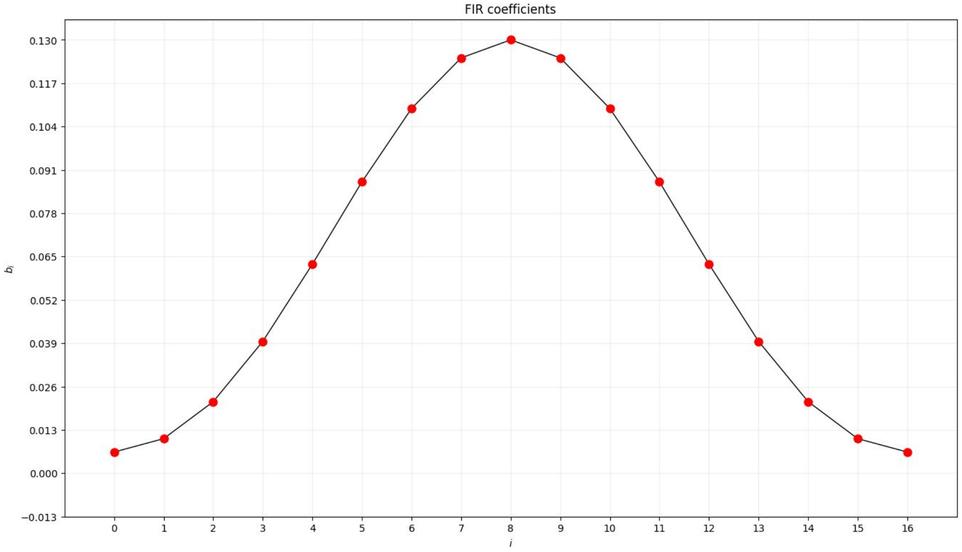


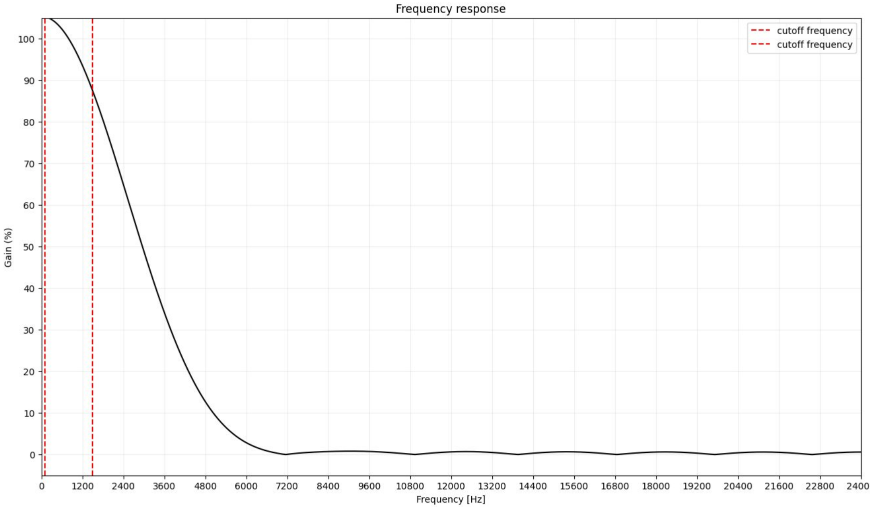
FIR Filter

- A hardcode 17-tap FIR Filter (N=17)
- Sample Rate = 48kHz
- Bandpass filter 100~1500 Hz

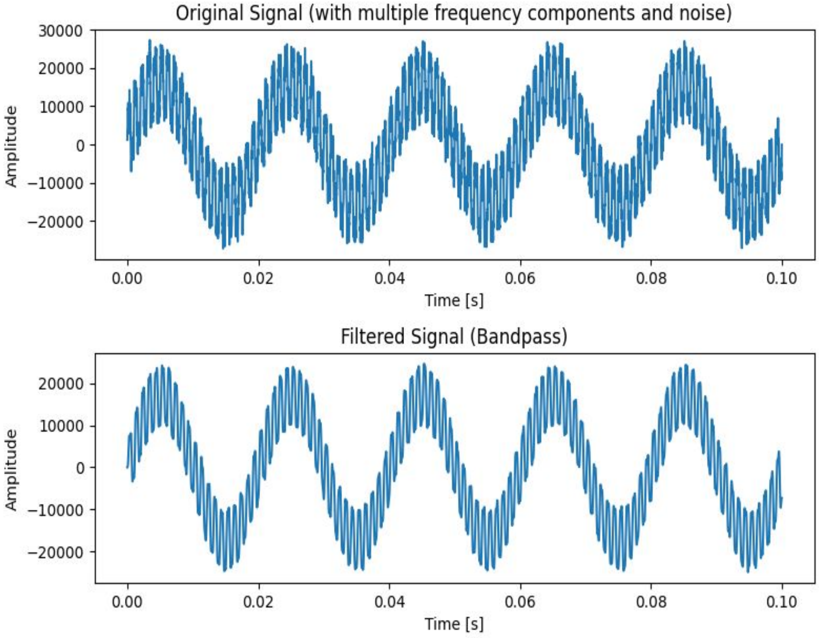
$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k]$$

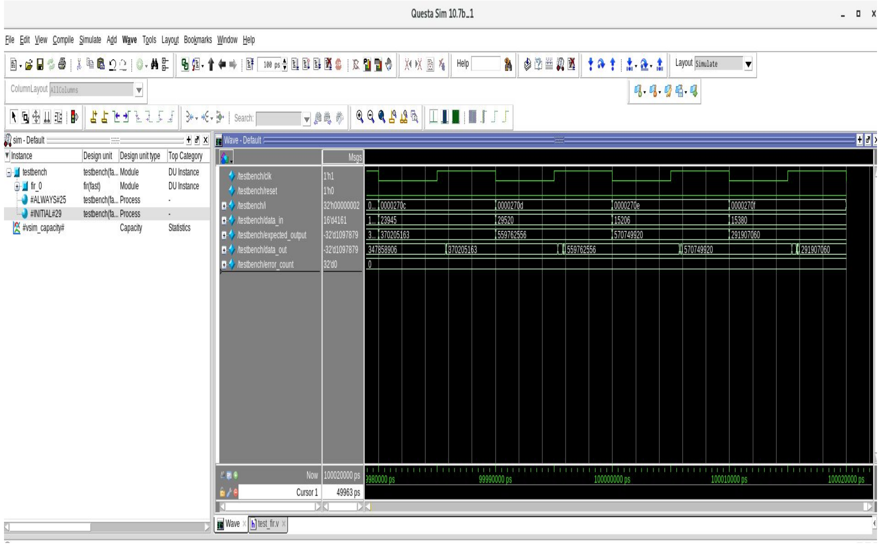
FIR Filter-Design





FIR Filter-Test





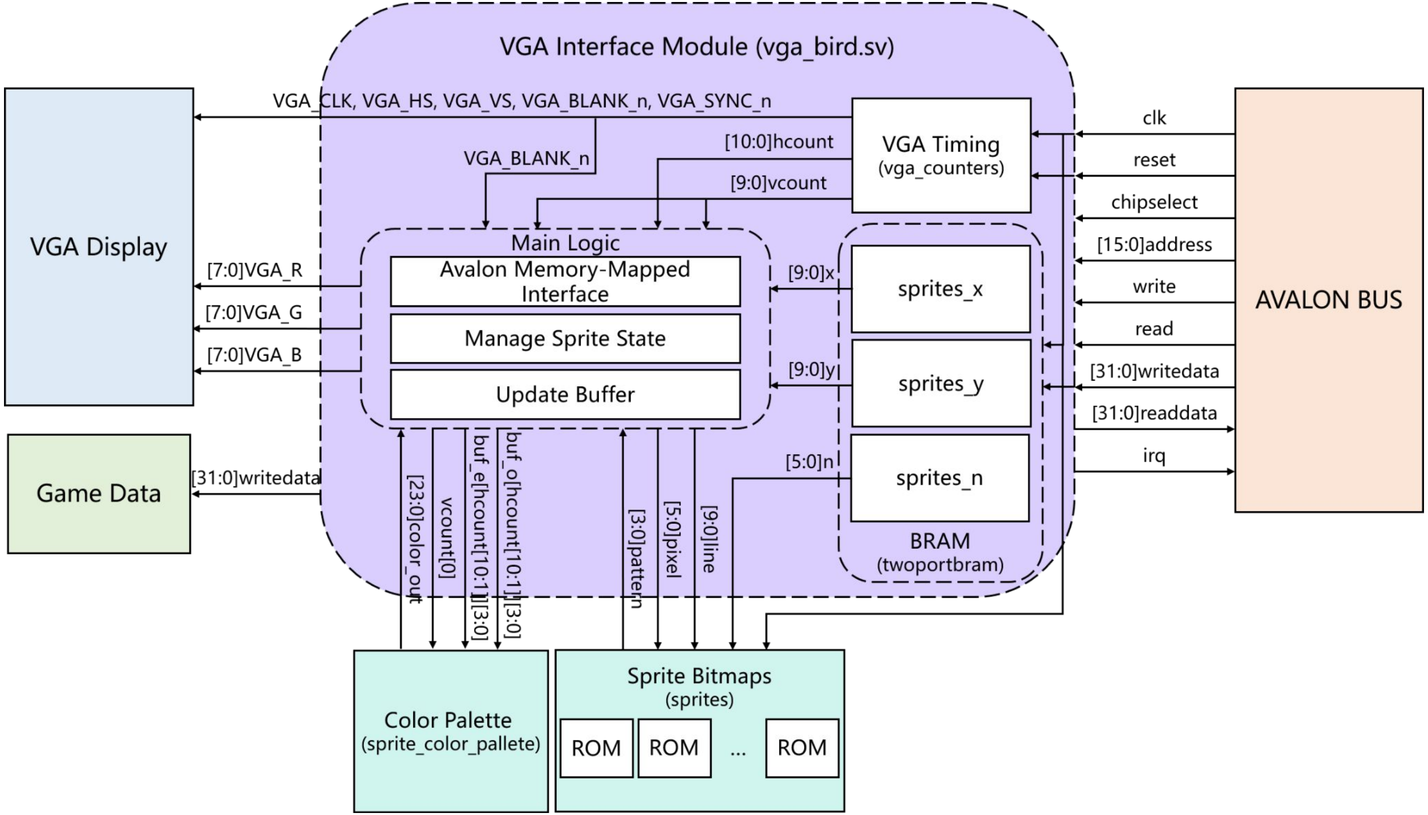
Instance	Design unit	Design unit type	Top Category
testbench	testbench(fa...	Module	DU Instance
fir_0	fir(fast)	Module	DU Instance
#ALWAYS#25	testbench(fa...	Process	-
#INITIAL#29	testbench(fa...	Process	-
#vsim_capacity#		Capacity	Statistics

	Msgs					
/testbench/clk	1'h1					
/testbench/reset	1'h0					
/testbench/i	32'h00000002	0...	0000270c	0000270d	0000270e	0000270f
/testbench/data_in	16'd4161	1...	23945	29520	15206	15380
/testbench/expected_output	-32'd1097879	3...	370205163	559762556	570749920	291907060
/testbench/data_out	-32'd1097879	347858906	370205163	559762556	570749920	291907060
/testbench/error_count	32'd0	0				

Now 100020000 ps

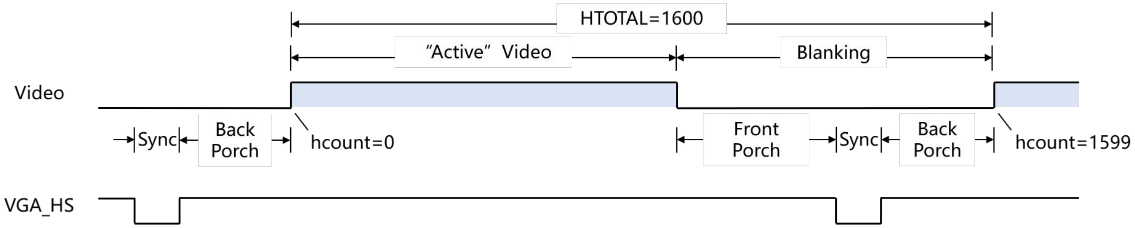
Cursor 1 49963 ps

VGA Display Module

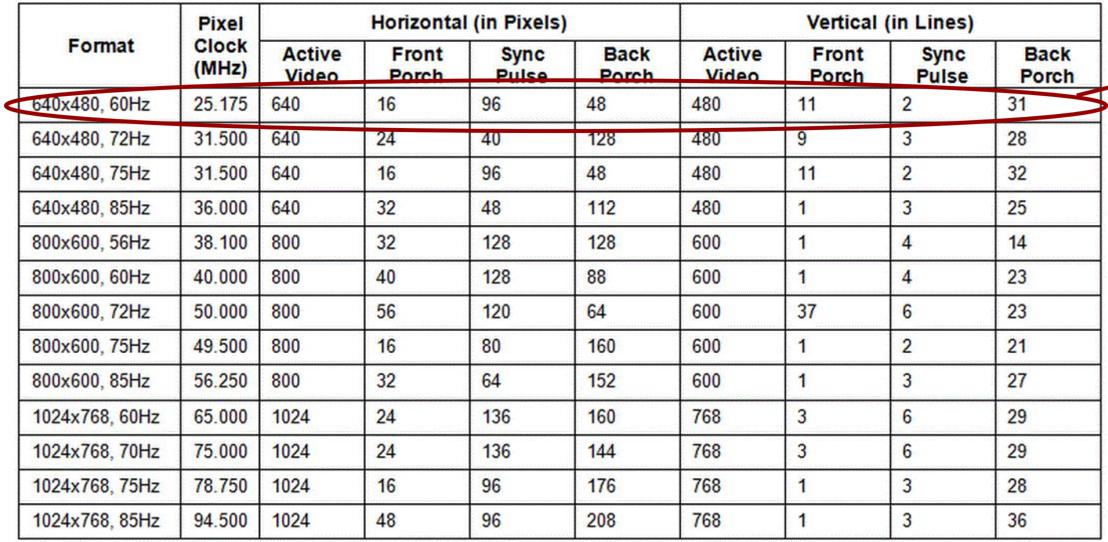


VGA Timing

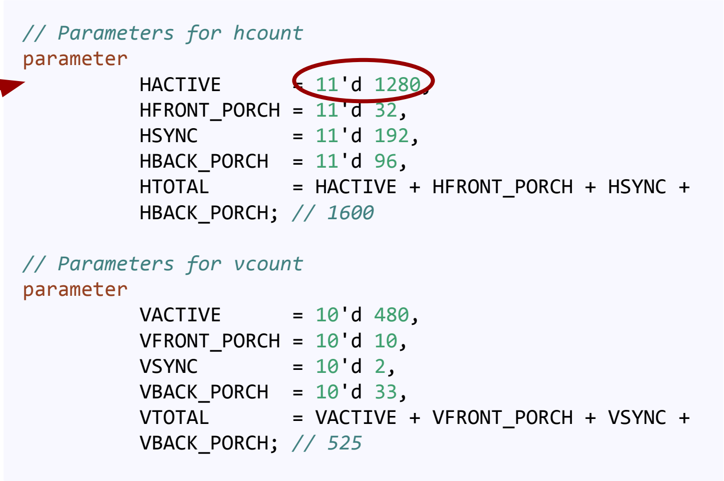
vga_counter module is used for generating VGA timing signals based on a 50 MHz clock.



Active low



Format	Pixel Clock (MHz)	Horizontal (in Pixels)				Vertical (in Lines)			
		Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640x480, 60Hz	25.175	640	16	96	48	480	11	2	31
640x480, 72Hz	31.500	640	24	40	128	480	9	3	28
640x480, 75Hz	31.500	640	16	96	48	480	11	2	32
640x480, 85Hz	36.000	640	32	48	112	480	1	3	25
800x600, 56Hz	38.100	800	32	128	128	600	1	4	14
800x600, 60Hz	40.000	800	40	128	88	600	1	4	23
800x600, 72Hz	50.000	800	56	120	64	600	37	6	23
800x600, 75Hz	49.500	800	16	80	160	600	1	2	21
800x600, 85Hz	56.250	800	32	64	152	600	1	3	27
1024x768, 60Hz	65.000	1024	24	136	160	768	3	6	29
1024x768, 70Hz	75.000	1024	24	136	144	768	3	6	29
1024x768, 75Hz	78.750	1024	16	96	176	768	1	3	28
1024x768, 85Hz	94.500	1024	48	96	208	768	1	3	36



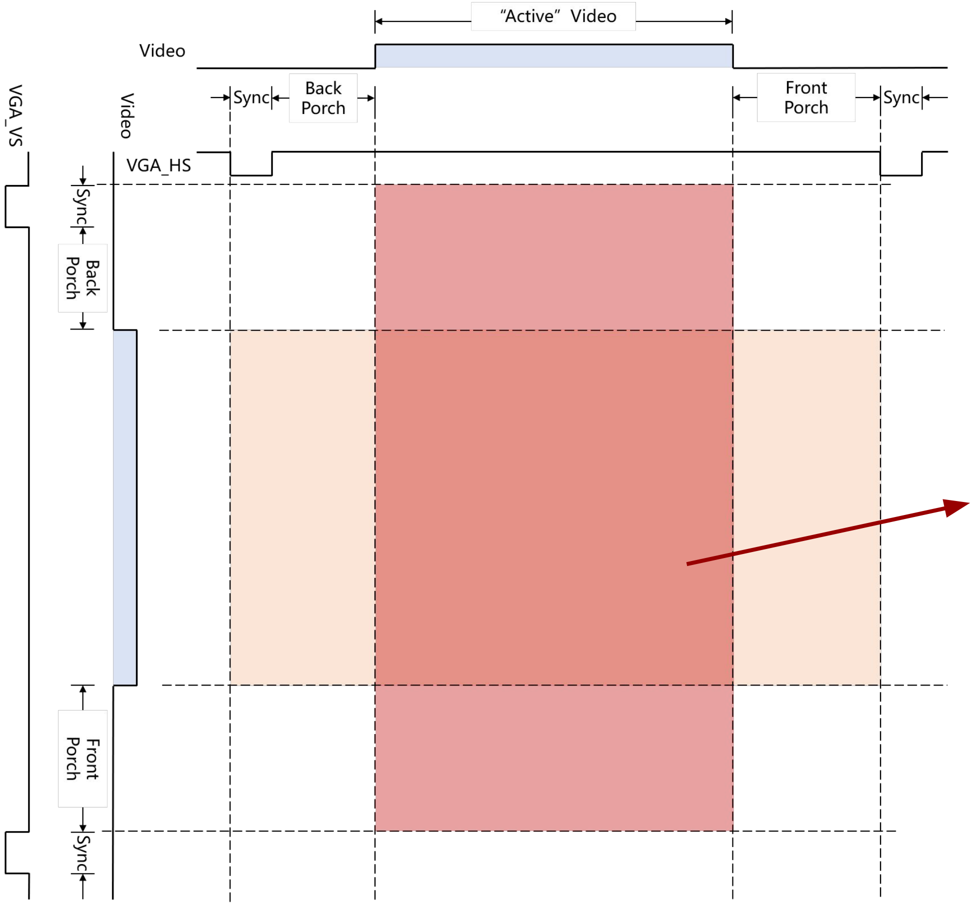
```
// Parameters for hcount
parameter
          HACTIVE      = 11'd 1280,
          HFRONT_PORCH = 11'd 32,
          HSYNC        = 11'd 192,
          HBACK_PORCH  = 11'd 96,
          HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
          HBACK_PORCH; // 1600

// Parameters for vcount
parameter
          VACTIVE      = 10'd 480,
          VFRONT_PORCH = 10'd 10,
          VSYNC        = 10'd 2,
          VBACK_PORCH  = 10'd 33,
          VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
          VBACK_PORCH; // 525
```

Horizontal Counter : [10:0] hcount

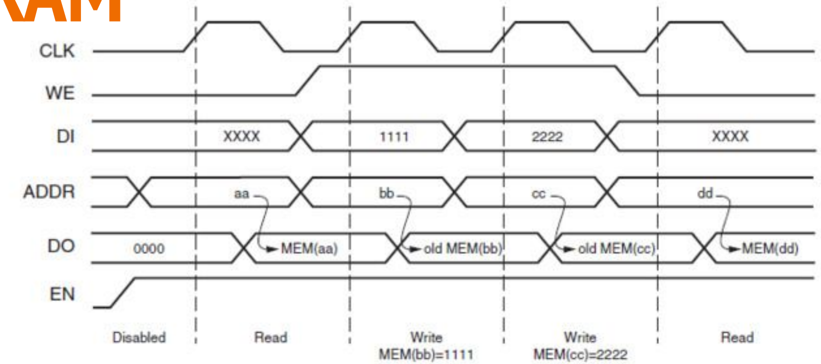
Vertical Counter: [9:0] vcount

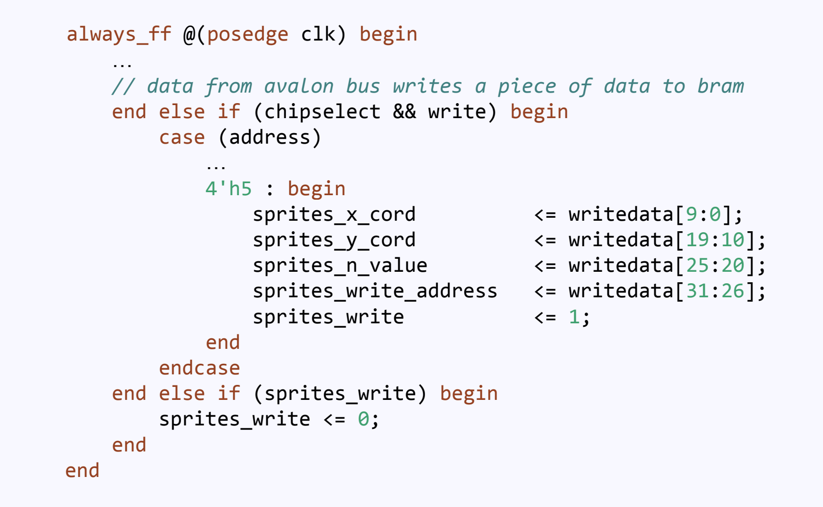
VGA Timing



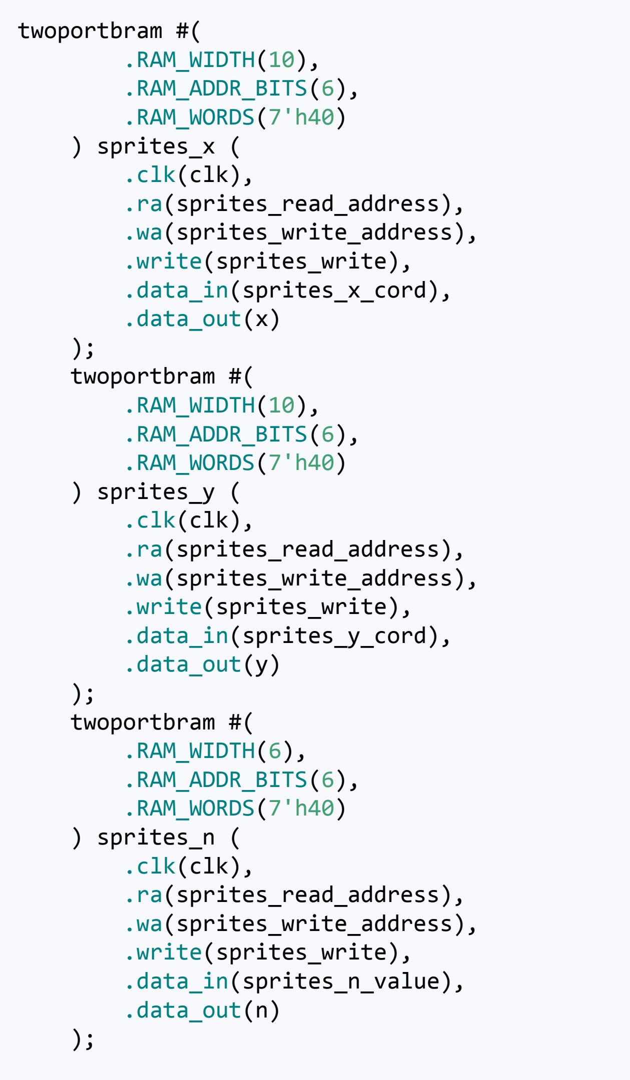
Final display area

BRAM



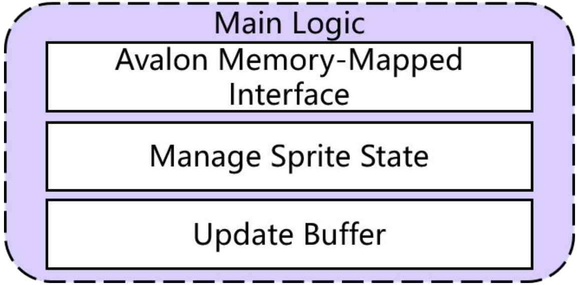


```
always_ff @(posedge clk) begin
    …
    // data from avalon bus writes a piece of data to bram
    end else if (chipselect && write) begin
        case (address)
            …
            4'h5 : begin
                sprites_x_cord          <= writedata[9:0];
                sprites_y_cord          <= writedata[19:10];
                sprites_n_value         <= writedata[25:20];
                sprites_write_address   <= writedata[31:26];
                sprites_write           <= 1;
            end
        endcase
    end else if (sprites_write) begin
        sprites_write <= 0;
    end
end
```



```
twoportbram #(
        .RAM_WIDTH(10),
        .RAM_ADDR_BITS(6),
        .RAM_WORDS(7'h40)
    ) sprites_x (
        .clk(clk),
        .ra(sprites_read_address),
        .wa(sprites_write_address),
        .write(sprites_write),
        .data_in(sprites_x_cord),
        .data_out(x)
    );
    twoportbram #(
        .RAM_WIDTH(10),
        .RAM_ADDR_BITS(6),
        .RAM_WORDS(7'h40)
    ) sprites_y (
        .clk(clk),
        .ra(sprites_read_address),
        .wa(sprites_write_address),
        .write(sprites_write),
        .data_in(sprites_y_cord),
        .data_out(y)
    );
    twoportbram #(
        .RAM_WIDTH(6),
        .RAM_ADDR_BITS(6),
        .RAM_WORDS(7'h40)
    ) sprites_n (
        .clk(clk),
        .ra(sprites_read_address),
        .wa(sprites_write_address),
        .write(sprites_write),
        .data_in(sprites_n_value),
        .data_out(n)
    );
```

VGA Controller (Main Logic)



Avalon Memory-Mapped Interface Handler

Address	Function
4'h3	Updates the score registers.
4'h5	Interprets writedata as a sprite packet, extracting the x-coordinate, y-coordinate, sprite index, and sprite entry, and sets these into the respective registers.

Sprite Rendering State Machine

State	Function
0	Prepares for reading sprite data.
1	Checks if the sprite is active and within the current line.
2	Updates the buffer with the sprite's pixel data if it is active.

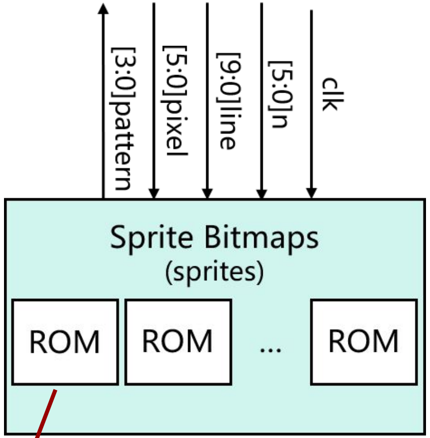
Buffer Update for Sprite Pixels

buf_e and buf_o: Manage even and odd frames or lines (640,4) → **Smooth display update**

Sprite Bitmaps and ROM

```
always_comb begin
    case (n_sprite)
        6'd1  : color_code = spr_rom_data[6'd1];  // Sprite 1
        6'd2  : color_code = spr_rom_data[6'd2];  // Sprite 2
        ...
        default : color_code = 4'h0; // Default color code
    endcase
end
```

```
assign spr_rom_addr = (line<<5) + pixel;
```



```
initial begin
    if (INIT_F != 0) begin
        $display("Creating rom_sync from init file '%s'.", INIT_F);
        $readmemh(INIT_F, memory);
    end
end
```

Memory Initialization

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 2 3 3 3 3 3 3 3 3 3 3 2 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 2 2 2 2 1 1 1 1 1 3 3 1 1 1 2 2 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 3 3 3 2 1 1 1 1 1 3 3 1 1 1 2 3 3 0 0 0 0 0 0 0
0 0 0 0 0 0 2 3 1 1 1 1 4 4 4 5 3 1 1 1 1 1 1 1 2 3 0 0 0 0 0 0
0 0 0 0 0 2 2 3 1 1 1 1 4 4 4 5 3 1 1 1 1 1 1 1 2 3 3 0 0 0 0 0
0 0 0 0 2 3 1 1 4 4 4 4 4 4 4 5 3 1 1 1 1 1 2 3 1 1 3 3 0 0 0 0
0 0 0 0 2 5 1 1 4 4 4 4 4 4 4 5 3 1 1 1 1 1 2 3 1 1 3 3 0 0 0 0
0 0 0 3 5 4 4 4 4 4 4 4 4 4 4 5 3 1 1 1 1 1 2 3 1 1 3 3 0 0 0 0
0 0 0 3 5 5 5 5 5 5 5 6 4 4 4 6 5 2 2 1 1 1 1 2 1 1 3 3 0 0 0 0
0 0 2 3 3 3 3 3 3 3 3 5 4 4 4 4 4 3 3 1 1 1 1 1 1 1 3 3 0 0 0 0
0 2 2 1 1 1 1 1 1 1 1 2 5 6 4 4 4 6 6 2 2 2 2 2 2 2 3 3 2 2 0 0
0 3 2 1 1 1 1 1 1 1 1 1 3 5 4 4 4 4 6 3 3 3 3 3 3 3 3 3 3 2 0 0
0 3 2 1 1 1 1 1 1 1 1 1 3 5 4 4 4 5 3 7 7 7 7 7 7 7 7 7 7 5 3 2
0 3 2 1 1 1 1 1 1 1 1 2 3 5 4 4 6 3 3 7 7 7 7 7 7 7 7 7 7 5 3 2
0 0 2 3 3 3 3 3 3 3 3 5 6 6 6 5 3 7 7 3 3 3 3 3 3 3 3 3 3 2 0 0
0 0 0 2 3 3 5 5 5 5 5 5 6 6 6 5 5 5 5 5 5 5 5 5 5 5 5 3 3 2 0 0
0 0 0 0 2 3 6 6 6 6 6 6 6 6 6 6 6 3 3 7 7 7 7 7 7 7 7 7 3 2 0 0
0 0 0 0 0 2 5 5 5 5 6 6 6 6 6 6 6 5 5 5 5 5 5 5 5 5 5 2 0 0 0
0 0 0 0 0 0 2 3 3 3 6 6 6 6 6 6 6 6 6 3 3 3 3 3 3 3 3 3 0 0 0 0
0 0 0 0 0 0 0 0 0 2 5 5 5 5 5 5 5 5 5 2 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

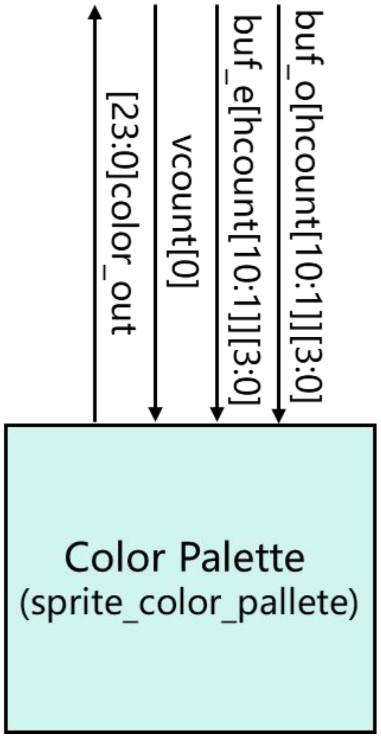
```
always_ff @(posedge clk) begin
    data <= memory[addr];
end
```

Data Output

Color Palette

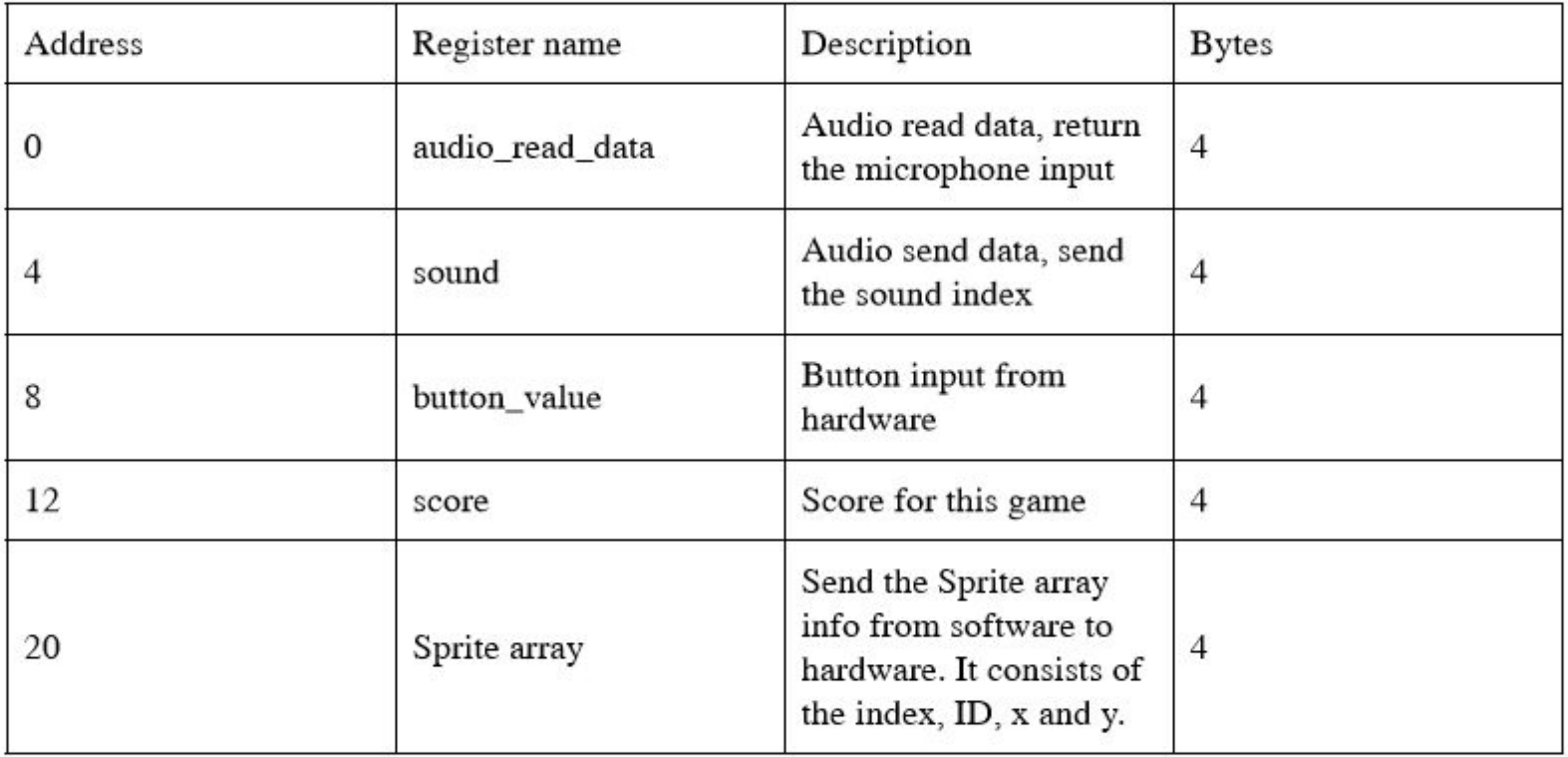
```
always_comb begin
    case(color_code)
        4'h0 : color = 24'hFFFFFF;
        4'h1 : color = 24'hFFFFFF;
        4'h2 : color = 24'h646361;
        ...
        default : color = 24'h000000;
    endcase
end
```

```
assign color_code = (select) ? color_code_o : color_code_e;
```



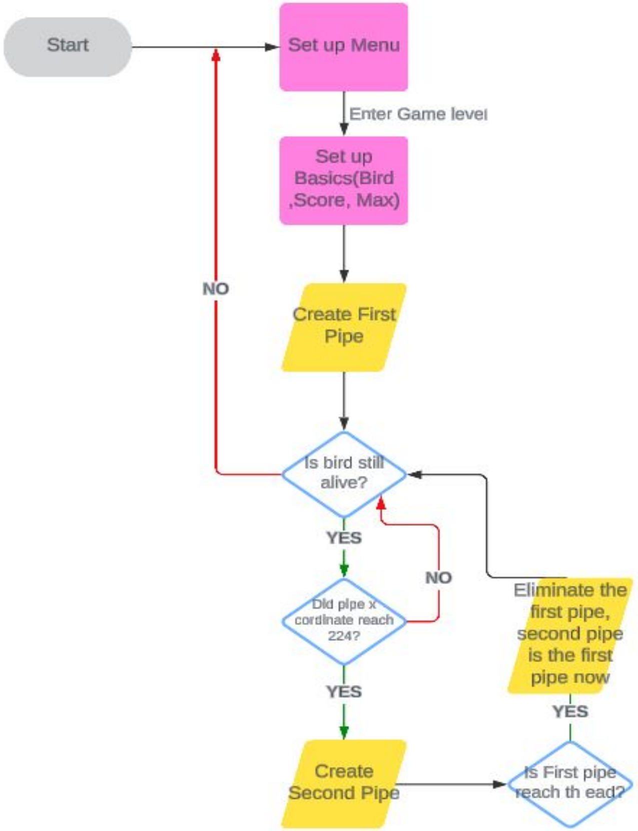
Color Palette
(sprite_color_pallete)

Register



Address	Register name	Description	Bytes
0	audio_read_data	Audio read data, return the microphone input	4
4	sound	Audio send data, send the sound index	4
8	button_value	Button input from hardware	4
12	score	Score for this game	4
20	Sprite array	Send the Sprite array info from software to hardware. It consists of the index, ID, x and y.	4

Game Logic diagram



Start

Set up Menu

Enter Game level

Set up Basics(Bird ,Score, Max)

Create First Pipe

Is bird still alive?

NO

YES

Did pipe x cordinate reach 224?

NO

YES

Create Second Pipe

Is First pipe reach th ead?

YES

Eliminate the first pipe, second pipe is the first pipe now

Software

The menu_setup(sprites) and scorecombosetup(sprites) functions are crucial for initializing the game's menu and basic configuration. The left initialization in sprite.sv sets up all objects with color. The code in the right is an example of how to use it in software:

Struct Definition

```
typedef struct {
      int x, y, dx, dy, id, index, hit;
} sprite;
```

Sprite Data Initialization in sprite.sv

```
6'd1  - 6'd10 = spr_rom_data[6'd1-10];  // 1-10
6'd11 - 6'd17 = spr_rom_data[6'd11-17]; // B-R
6'd18 - 6'd25 = spr_rom_data[6'd18-25]; // E_w-N_w
6'd26 - 6'd27 = spr_rom_data[6'd26-27]; // A-X
6'd28 - 6'd38 = spr_rom_data[6'd28-38]; // BIRD-O_w
```

Example Usage in Software for setting “MENU”

```
sprites[1].id = 20; // M
sprites[2].id = 18; // E
sprites[3].id = 25; // N
sprites[4].id = 19; // U

for (int i = 1; i < 5; i++) {
    sprites[i].x = 108 + 32*(i-1); // Position in
corresponding x and y pixels
    sprites[i].y = 120;
    sprites[i].dx = 0; // Object doesn't move horizontally
    sprites[i].dy = 0; // Object doesn't move vertically
    sprites[i].index = i; // Setting the index for further
use
}
```

Software

The check_bird_position(sprite *sprites, vga_pipe_position_t *pipe_info_first, vga_pipe_position_t *pipe_info_second) and create_pipe(sprite *sprites, vga_pipe_position_t *pipe_info, int pipe_index_start, int difficulty_level) functions are essential for gameplay mechanics. They perform the following tasks:

1. check_bird_position():
   - Checks if the bird collides with any pipes.
   - Return 1 if collides, otherwise game continue
2. create_pipe():
   - Creates a new pipe if the first pipe has reached 224 pixels and the bird is still alive.
   - Uses the pipe_index_start and difficulty_level parameters to determine the pipe's speed and length

Software/Hardware Interaction

- **AUDIO:**

int check_receive_audio(int counter, float sum_audio_data, int aud_fd, aud_mem_t amt):

- Receives audio data from the hardware.
- If the received audio value exceeds 500,000,000, the bird flaps its wings.
- If the value is less than or equal to 500,000,000, the bird continues to fall.

- **Sound:**

send_sound(&c, aud_fd):

- Sends a sound value to the hardware.
- Sends value `0` for stop
- Sends value `1` when the bird flap.
- Sends value `2` when the bird collides.
- The hardware plays the corresponding sound effect based on the value sent.

Lesson Learned

- VGA Display
- Sprite Implementation
- Hardware and software Collaboration
- Sound output
- Microphone input

DEMO