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

Note: The ADC and DAC can run at different rates
CODEC Interface IP

Graphs showing the waveforms for different signals:
- CLOCK_50MHz
- advance_48kHz
- dac_l/r
- adc_l/r

Time scale from 0 to 8.
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

Original Signal (with multiple frequency components and noise)

Filtered Signal (Bandpass)
VGA Timing

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

### VGA Timing Parameters

- **Horizontal Counter:** [10:0] hcount
- **Vertical Counter:** [9:0] vcount

### VGA_Hs Timing Table

<table>
<thead>
<tr>
<th>Format</th>
<th>Pixel Clock (MHz)</th>
<th>Active Video</th>
<th>Front Porch</th>
<th>Sync Pulse</th>
<th>Back Porch</th>
</tr>
</thead>
<tbody>
<tr>
<td>640x480, 60Hz</td>
<td>25.175</td>
<td>640</td>
<td>16</td>
<td>96</td>
<td>48</td>
</tr>
<tr>
<td>640x480, 72Hz</td>
<td>31.500</td>
<td>640</td>
<td>24</td>
<td>40</td>
<td>128</td>
</tr>
<tr>
<td>640x480, 75Hz</td>
<td>31.500</td>
<td>640</td>
<td>16</td>
<td>96</td>
<td>48</td>
</tr>
<tr>
<td>640x480, 85Hz</td>
<td>36.000</td>
<td>640</td>
<td>32</td>
<td>48</td>
<td>112</td>
</tr>
<tr>
<td>800x600, 56Hz</td>
<td>38.100</td>
<td>800</td>
<td>32</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>800x600, 60Hz</td>
<td>40.000</td>
<td>800</td>
<td>40</td>
<td>128</td>
<td>88</td>
</tr>
<tr>
<td>800x600, 72Hz</td>
<td>50.000</td>
<td>800</td>
<td>56</td>
<td>120</td>
<td>64</td>
</tr>
<tr>
<td>800x600, 75Hz</td>
<td>49.500</td>
<td>800</td>
<td>16</td>
<td>80</td>
<td>160</td>
</tr>
<tr>
<td>800x600, 85Hz</td>
<td>56.250</td>
<td>800</td>
<td>32</td>
<td>64</td>
<td>152</td>
</tr>
<tr>
<td>1024x768, 60Hz</td>
<td>65.000</td>
<td>1024</td>
<td>24</td>
<td>136</td>
<td>160</td>
</tr>
<tr>
<td>1024x768, 70Hz</td>
<td>75.000</td>
<td>1024</td>
<td>24</td>
<td>136</td>
<td>144</td>
</tr>
<tr>
<td>1024x768, 75Hz</td>
<td>78.750</td>
<td>1024</td>
<td>16</td>
<td>96</td>
<td>176</td>
</tr>
<tr>
<td>1024x768, 85Hz</td>
<td>94.500</td>
<td>1024</td>
<td>48</td>
<td>96</td>
<td>208</td>
</tr>
</tbody>
</table>

### Horizontal Timing Parameters

- **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**

### Vertical Timing Parameters

- **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**
VGA Timing

Final display area
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
VGA Controller (Main Logic)

Avalon Memory-Mapped Interface Handler

<table>
<thead>
<tr>
<th>Address</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>4'h3</td>
<td>Updates the score registers.</td>
</tr>
<tr>
<td>4'h5</td>
<td>Interprets writedata as a sprite packet, extracting the x-coordinate, y-coordinate, sprite index, and sprite entry, and sets these into the respective registers.</td>
</tr>
</tbody>
</table>

Sprite Rendering State Machine

<table>
<thead>
<tr>
<th>State</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Prepares for reading sprite data.</td>
</tr>
<tr>
<td>1</td>
<td>Checks if the sprite is active and within the current line.</td>
</tr>
<tr>
<td>2</td>
<td>Updates the buffer with the sprite's pixel data if it is active.</td>
</tr>
</tbody>
</table>

Buffer Update for Sprite Pixels

buf_e and buf_o: Manage even and odd frames or lines

Smooth display update

(640,4)
Sprite Bitmaps and ROM

```verilog
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

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

Memory Initialization

Data Output
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_pallelete)
Register

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

The menu_setup(sprites) and scorecombosssetup(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**

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

**Sprite Data Initialization in sprite.sv**

```c
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”**

```c
for (int i = 1; i < 5; i++) {
    sprites[i].id = 20; // M
    sprites[i].id = 18; // E
    sprites[i].id = 25; // N
    sprites[i].id = 19; // U
    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
}
```
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