Final Project Report

ECE 385: Digital Systems Laboratory

Flappy Bird

Authors:
Zhuohao Li
Qianhe Ye

Instructor:
Prof. Chushan Li, Prof. Zuofu Chen

Date:
May 26, 2024

1 Overview

Our game is an upgraded version of Flappy Bird built on the DE2-115 board, primarily based on FPGA. Players control a bird that navigates through pipes and collects coins to earn points. Colliding with pipes or the screen boundaries results in the bird's demise. Key features include real-time rendering of moving pipes and backgrounds, displaying the score at the center of the screen, allowing the bird to jump a fixed distance with each press of the space bar, and implementing wing-flapping animation for the bird.

2 Features

Background The background is a image of size 320*480. This is a stationary picture, with a bird flying across this background.

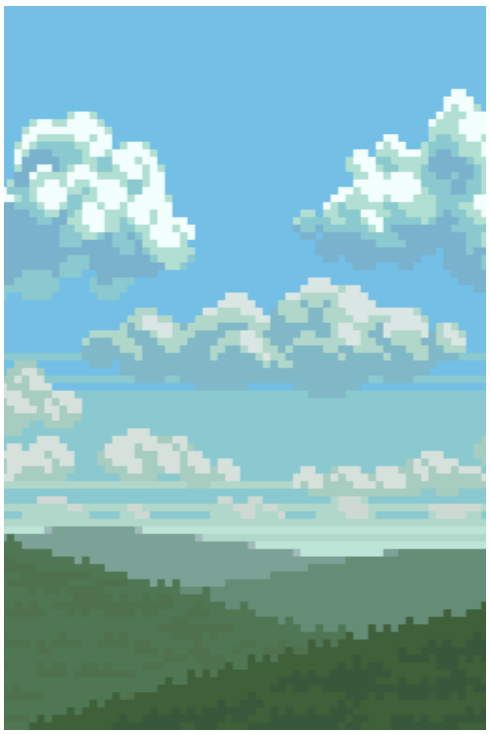

Figure 1: Background

Bird Our bird is an image of size 21*13 drawn by ourselves. The bird has a constant downward velocity. Pressing the "spacebar" once makes the bird jump to a fixed height, while holding down the "spacebar" makes the bird ascend continuously. Our bird has a flapping wing animation consisting of three frames.

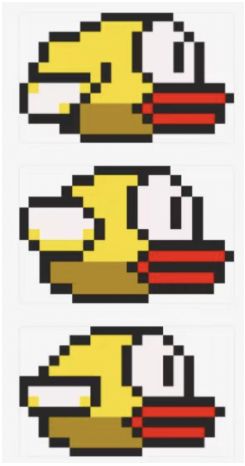

Figure 2: Bird

Pipe Our pipes are images composed of cylinders and edges. The gaps between pipes are fixed in distance but randomized in position. The bird navigates through the pipes by jumping up and down.

Figure 3: Pipe

Ground Our ground moves backward at a fixed speed, which matches the speed of the pipes' movement.

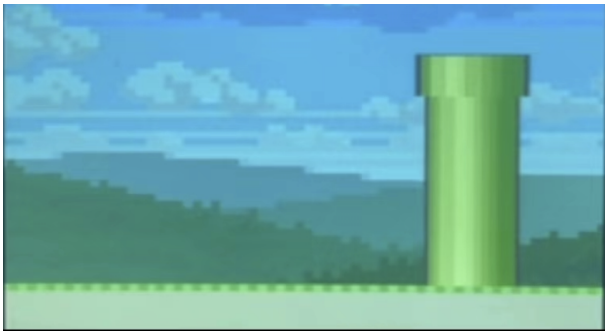

Figure 4: Ground

Coin The coin is drawn by ourselves. The x-coordinate is between two pipes, and the y-coordinate is randomly generated. When the bird collects a coin, one point is added to the score, and the coin disappears after being collected.

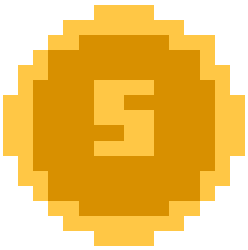

Figure 5: Coin

Score Our score consists of two digits, displayed at the top center of the screen. The bird earns one point by passing through a pipe or collecting a coin. The maximum score is ninety-nine points.

Figure 6: Digits

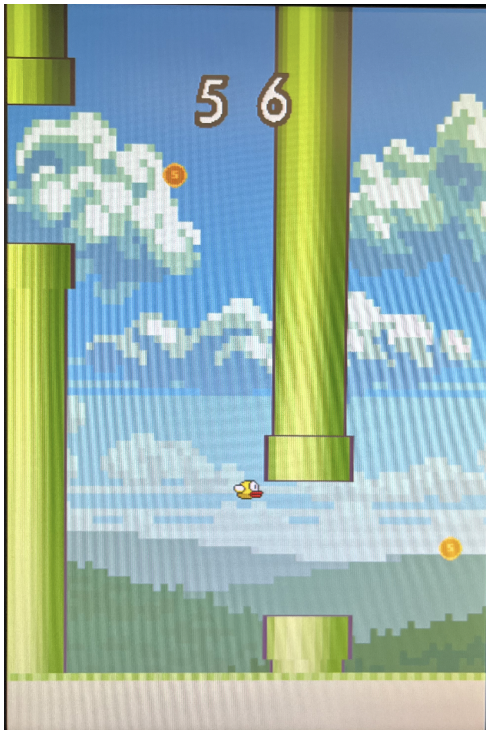

Figure 7: Score

Start Frame Our starting frame features a bird stationary in the center of the screen with a score of 0, and the ground moving backward. Pressing the spacebar to start causes the pipes to move leftward, and the bird begins to descend.

Figure 8: Start Frame

End Frame Our end frame shows the bird frozen in place after colliding and dying, while the pipes and ground remain stationary. The final score is displayed.

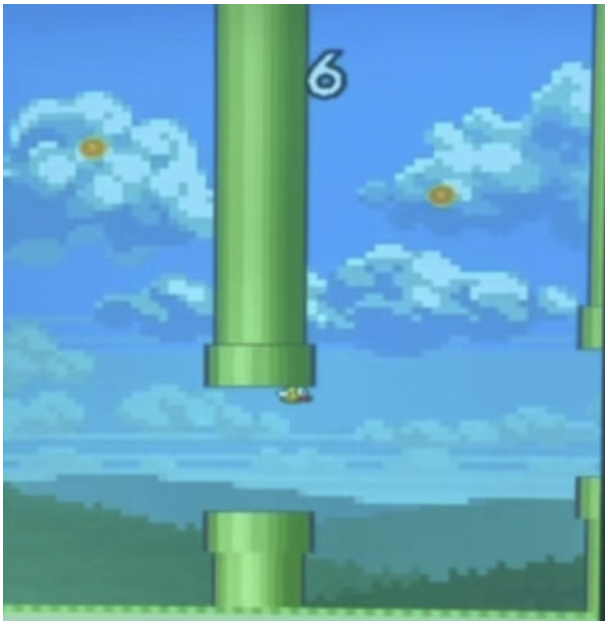

Figure 9: End Frame

3 Block Diagram

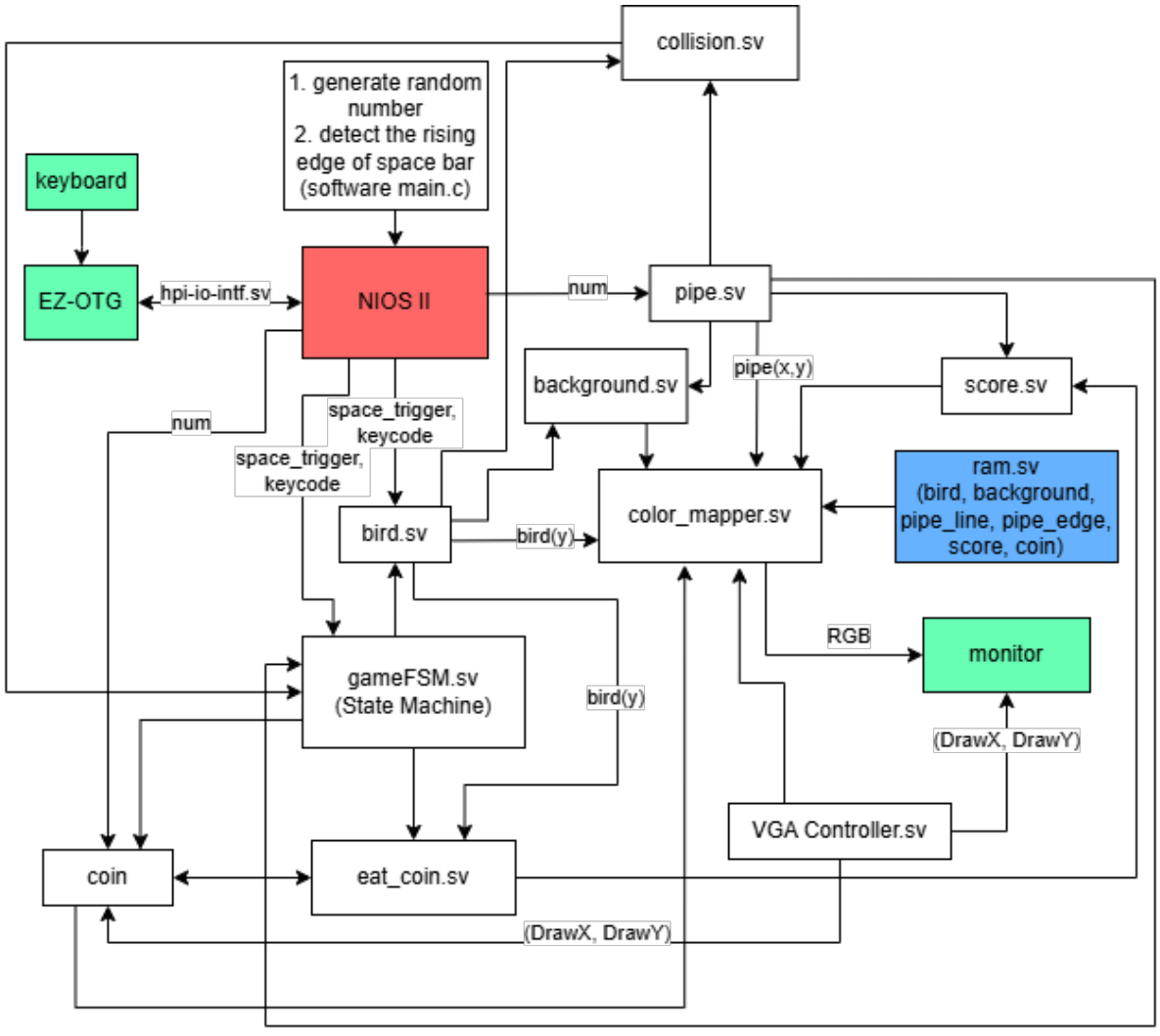

Figure 10: Block Diagram

4 General flow of the circuit

The design is primarily a hardware-based circuit, with C code used in software to generate random numbers and manage some I/O operations in NIOS II. This facilitates hardware interactions, such as reading inputs from the keyboard. The keyboard serves as the input device, while the monitor is used as output devices. Although the EZ-OTG is not integrated into the FPGA, it is present on the DE2-115 board, and we use interfaces to communicate with them. When a player inputs command using the keyboard, the NIOS II transfers the message to the game logic modules. These modules then output various signals to the color mapper module. The VGA controller informs the color mapper of the pixel it is currently drawing, and the color mapper outputs the corresponding RGB values. Consequently, the correct outputs, such as sprite movements and score, are displayed on the monitor.

5 Module Description

5.1 Quartus

Module: collision
Inputs: Clk, Reset, is_ball, is_pipe, is_pipe_edge
Outputs: collision
Description: This module detects the collision of bird between pipe and pipe edge.

Module: GameFSM
Inputs: Clk, Reset, space_trigger, collision, is_bottom, keycode
Outputs: game_state
Description: This module controls the three states of the game, start, active and over. When keycode space(space_trigger) is detected, the game goes to active state; when collision detected, the game goes to over state; when keycode enter(8'h28) is detected, the game goes to start state.

Module: ball
Inputs: Clk, Reset, frame_clk, DrawX, DrawY, space_trigger, game_state
Outputs: Ball_X_Pos, Ball_Y_Pos, is_ball, is_bottom, b_addr
Description: This module generates the position of the bird based on the keyboard input, decides the area which draws the bird sprite, decides the area of ground, and creates the effect of the bird flapping its wings when game is at active state.

Module: pipe
Inputs: Clk, Reset, frame_clk, random_num, DrawX, DrawY, game_state
Outputs: is_pipe, is_pipe_edge, is_grd_y, is_grd_g_d, is_grd_g_l, p_addr, p_e_addr, score1
Description: This module is responsible for generating and managing the positions of pipes in the game environment. It simulates the movement of two pipes representing the obstacles the player must navigate through. Each pipe's opening position is determined by a random number, and when the bird successfully passes through the gap between the pipes, the player's score increases by 1. Additionally, this module includes the logic for drawing a ground that moves synchronously with the pipes.

Module: eat_coin
Inputs: Clk, Reset, frame_clk, Ball_X_Pos, Ball_Y_Pos, coin1_X_Pos, coin1_Y_Pos, coin2_X_Pos,

coin2_Y_Pos, game_state
Outputs: score2, coin1_collected, coin2_collected
Description: This module detects when the bird collects a coin and increments the score accordingly. Since at most two coins are displayed on the screen simultaneously, the module handles the logic for these two coins. The y-coordinate of each coin is determined randomly, while the x-coordinate is always positioned between two pipes. When a coin moves off the left side of the screen, it reappears on the right with a new random y-coordinate. A coin is considered collected when the distance between the bird's center and the coin's center is less than 10 pixels.

Module: coin
Inputs: Clk, Reset, frame_clk, random_num, DrawX, DrawY, game_state, coin1_collected, coin2_collected
Outputs: is_coin, c_addr, coin1_X_Pos, coin1_Y_Pos, coin2_X_Pos,coin2_Y_Pos
Description: This module generates the position of the coin based on the random number and decides the area and address which draws the coin sprite based on the game state. If the coin is told to be collected, then it is not drawn.

Module: score
Inputs: Clk, Reset, score1, score2, DrawX, DrawY
Outputs: is_score1, is_score2, s1_addr, s2_addr
Description: This module generates the position of the score and decides the digit values based on the score which adds the pipe one and the coin one.

Module: background
Inputs: Clk, Reset, frame_clk, DrawX, DrawY, is_ball, is_pipe
Outputs: is_bg, bg_addr
Description: This module generates the position and address of the background, which is under all the other sprites.

Module: ram
Inputs: Clk, read_address
Outputs: data_Out
Description: On-chip memory is used to read text files into RAM. We use this module to store the indices of our palette for various components.

Module: color_mapper
Inputs: Clk, is_ball, is_bg, b_addr, bg_addr, p_addr, p_e_addr, s1_addr, s2_addr, c_addr, is_pipe, is_pipe_edge, is_grd_y, is_grd_g_d, is_grd_g_l, is_score1, is_score2, is_coin, DrawX, DrawY
Outputs: VGA_R, VGA_G, VGA_B
Description: This module decides the output color to VGA for each pixel and draws the picture for each frame.

Module: HexDriver
Inputs: In0
Outputs: Out0
Description: The keyboard input will be displayed on the HexDrive of the DE2 board, allowing us to verify that the keyboard is functioning correctly.

Module: VGA_controller
Inputs: Clk, Reset, VGA_CLK
Outputs: VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, DrawX, DrawY
Description: The module takes the ball's current position data on the screen and updates it on the VGA display. It uses vertical and horizontal sync signals to control the VGA drawing process. This module sets the screen configuration, including size, edges, and sync signals. It also keeps track of the current pixel being drawn, enabling the display functionality.

Module: hpi_io_intf.sv
Inputs: Clk, Reset, from_sw_r, from_sw_w, from_sw_cs, from_sw_reset; [1:0] from_sw_address; [15:0] from_sw_data_out
Outputs: OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N; [1:0] OTG_ADDR; [15:0]from_sw_data_in
Inout: [15:0] OTG_DATA
Description: This module serves as the interface between the NIOS II processor and the EZ-OTG chip. It transmits read, write, address, and data buffer signals to the OTG chip to facilitate accurate data reading and writing. Additionally, it enables control over the input-output bus.

Module: lab8.sv
Inputs: CLOCK_50, OTG_INT; [3:0] KEY
Outputs: VGA_HS, VGA_VS, VGA_BLANK_N, VGA_SYNC_N, VGA_CLK, OTG_CS_N, OTG_RD_N, OTG_WR_N, OTG_RST_N, DRAM_RAS_N, DRAM_CAS_N, DRAM_CKE, DRAM_WE_N, DRAM_CS_N, DRAM_CLK; [1:0] OTG_ADDR, DRAM_BA; [3:0] DRAM_DQM; [6:0] HEX0, HEX1; [7:0] LEDG, VGA_R, VGA_G, VGA_B; [12:0] DRAM_ADDR
Inout: [15:0] OTG_DATA, [31:0] DRAM_DQ
Description: It is the top level module of lab 8.
Purpose: This module connects all other modules and manages the port communication between them.

5.2 Qsys

onchip_memory2_0 This is the CPU's memory block, which can store data for the program.

sdram Due to the limited space of on-chip memory, this block adds an additional SDRAM to our system. We need to use an SDRAM controller to interface with it.

sdram_pll This module generates the clock for the SDRAM. The PLL adds a 3ns delay to the SDRAM to allow time for the outputs to stabilize.

sysid_qsys_0 This block ensures the accurate transfer of software and hardware by cross-referencing between the C code and SystemVerilog. It verifies that data transfer is executed in the correct format.

jtag_uart_0 Enables terminal access for debugging the software.

Keycode Functionality involves reading data, updating the USB chip, and subsequently transmitting it to the software for logical instructions.

otg_hpi_address PIO facilitates locating the desired memory address within the SoC, which is transmitted from the software to the FPGA.

otg_hpi_data This PIO facilitates bidirectional data transfer between the FPGA and software. With a width of 16 bits, it supports substantial data exchange between the two.

otg_hpi_r PIO that enables the read signal to access the memory of the SoC.

otg_hpi_w PIO that enables the write signal to access the memory of the SoC.

otg_hpi_cs PIO that enables the activation and deactivation of the memory within the SoC.

otg_hpi_reset PIO that enables the reset signal of the memory of the SoC.

space_trigger This PIO transfers a 8-bit signal to denote that if the space has been pushed.

random_num This PIO transfers the random number generated in main.c.

clk_0 This module acts as the master clock for the entire system, controlling the clock signals for all other modules. It also has the capability to reset the clock signals for other modules.

nios2_gen2_0 The Nios II block translates C code into SystemVerilog, enabling it to be executed on the FPGA board.

6 Design Procedure

The foundation of this project is based on LAB8 ”Bouncing Ball,” where the circuits for VGA display, keyboard input, and ball motion function effectively. The project building procedure is as follows.

- Building on the ball’s motion, we add more complex features to the motion model. For example, we introduce gravity to the model and generate space_trigger in NIOS II to let the model go up a certain distance when space is pressed once.
- Similar to the procedure of creating a ball, pipes and pipe edges are created, with the position of blank in the middle decided by the random number generated in NIOS II.
- The sprite of bird is drawn at the position of ball model, with the feature of flapping its wing.
- Background generated under all other pictures.

- Draw the ground at the bottom, which moves along the pipe.
- Score shown on top of the screen.
- Add coin feature and link the eating of coins to the adding of score.

7 State Machine

Our game primarily consists of three states: Start Page, Game Active, and Game Over. Upon entering the game, it initially remains in state 1. The bird is stationary at the center of the screen. When the player presses the spacebar, the game transitions to state 2. Pipes and ground start moving backward. If the bird collides with a pipe or the ground, it dies, and the game enters state 3. In this state, the bird, ground, coins, and pipes are all stationary. When the player presses Enter, the game restarts.

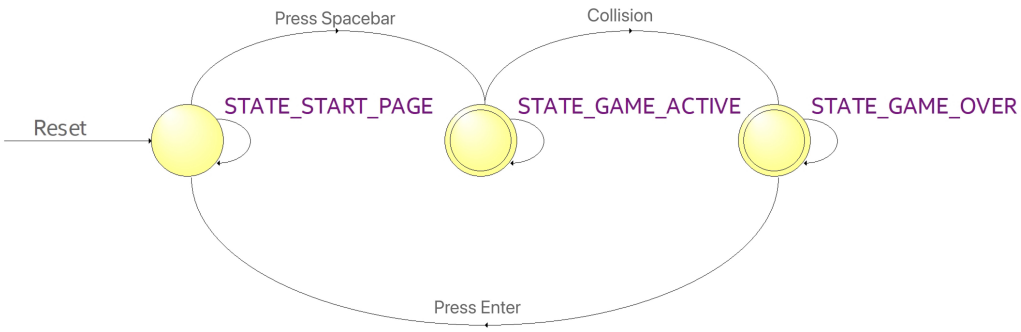

Figure 11: State Machine

8 Design statistics

LUT	4456
DSP	20
Memory	921600 bits
Flip-Flop	2417
Frequency	143.27MHz
Static Power	105.28mW
Dynamic Power	0.76mW
Total Power	175.73mW

Table 1: Design resources and statics

9 Known Bugs

- When the bird collides with the pipe, it will overlap with the pipe.
- If the bird goes straight to the top of the screen, it will disappear, and then the game will end.

10 Accomplishment

Our project perfectly replicates the game of ”Flappy Bird,” featuring well-crafted visuals, fluid animations, and gameplay mechanics that closely mirror those of the original game. And we add some additional features like a coin-collecting function.

11 Conclusion

Thanks to the Flappy Bird! It marks our last joint course project during our undergraduate years. Happy graduation!