ECE 385

Fall 2023

Final Project

Flappy Bird

Hank Zhou & Yuxuan Ma
TY/Friday
Tianhao Yu

1. Overview

In this project we implemented the popular game "Flappy Bird" on the Urbana board. Originally released in 2013, Flappy Bird is a side-scrolling game in which players control a small bird and try to fly between green pillars without hitting them. In our version, players use the space bar to control the bird to jump and navigate through the pillars. In addition, players can also press “1” and “2” to control the moving speed of the background to control the difficulty of the game; press “A” and “S” to change the background schemes; press “I” to turn on the invincible mode: when the bird hits the pillar, the game will not end, but the bird will "destroy" the pillars. The bird gets one point for each pillar it passes.

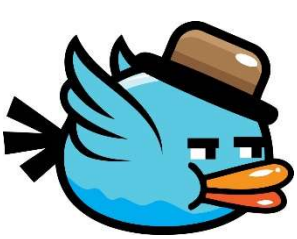
2. List of Features

Background The game background is an image of size 640 * 480 pixels. Background scrolls from right to left.

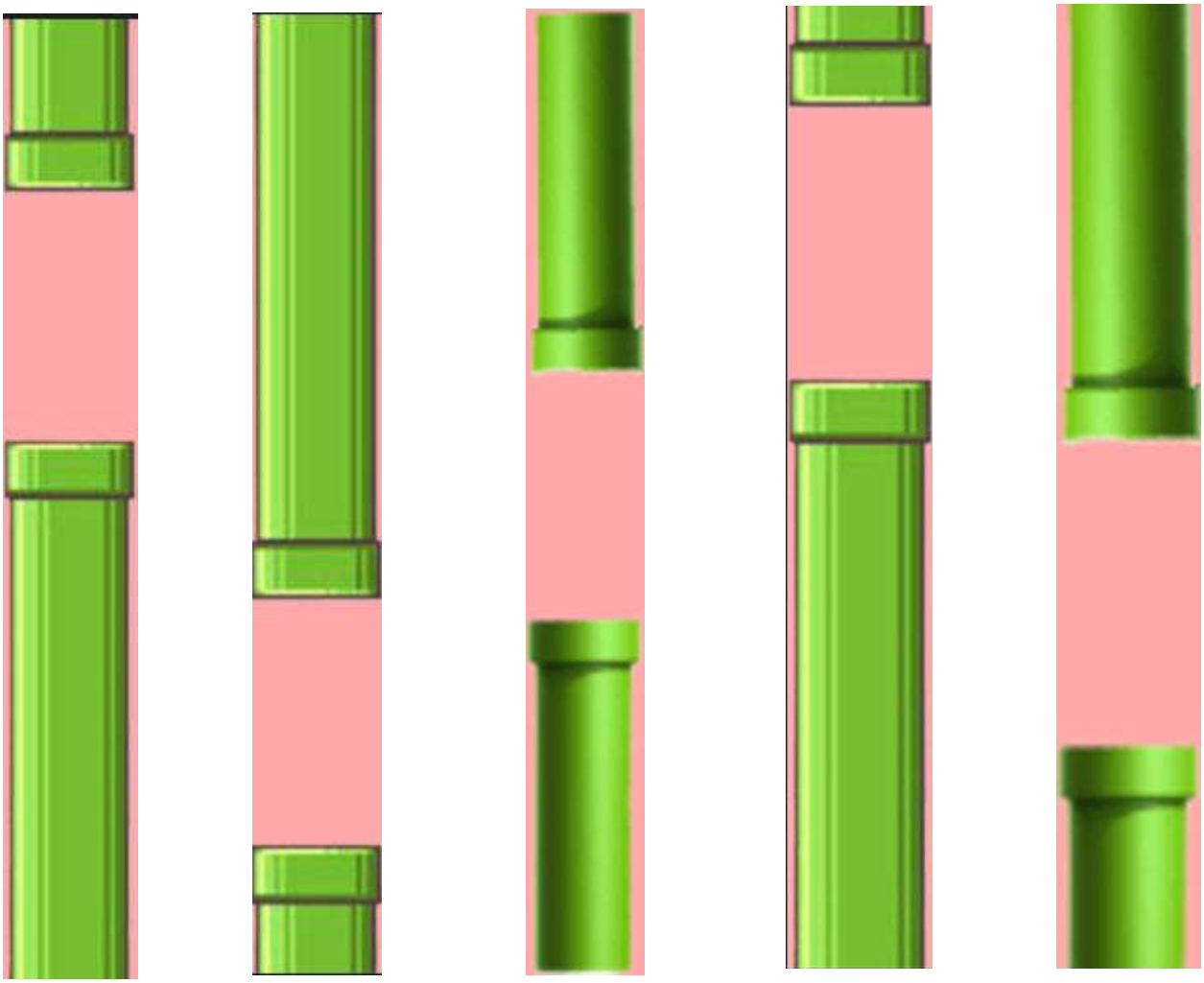
Start Frame An image of a button will appear before the game starts. After pressing the space bar, the game will enter the next scene. The words "FlappyBird Get Ready!" will appear on the background. Press the space bar to start the game.

End Frame When the bird hits the pillar, the game will end and a "Game Over" image will appear.

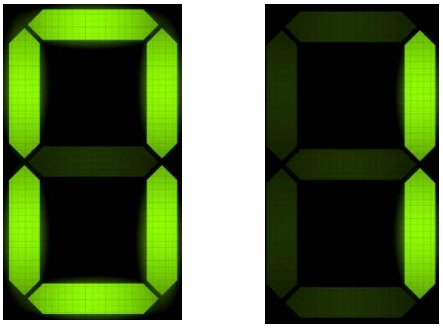
Bird Represents the player's character. The bird can jump, and the player uses the space bar to control the bird's jump. In addition to this, we added animation. When the space bar is pressed, the bird will flap its wings.



Pillars The green pillars are obstacles that the bird must cross. The bird needs to pass through the gap between the upper and lower pillars.



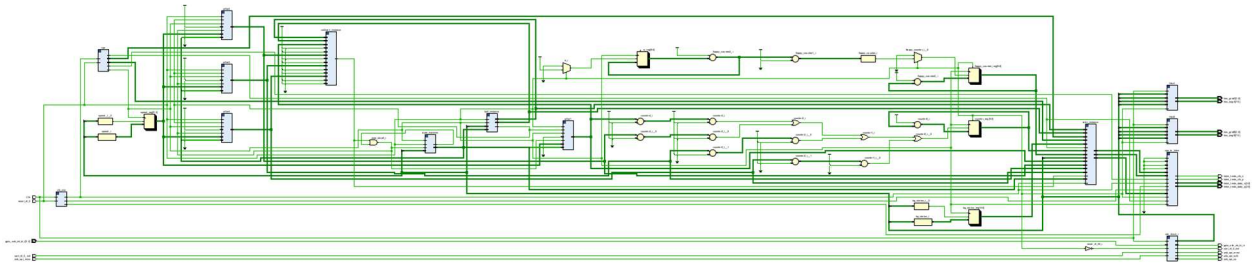
Score The score is displayed on the screen, representing the number of pillars successfully passed by the bird.



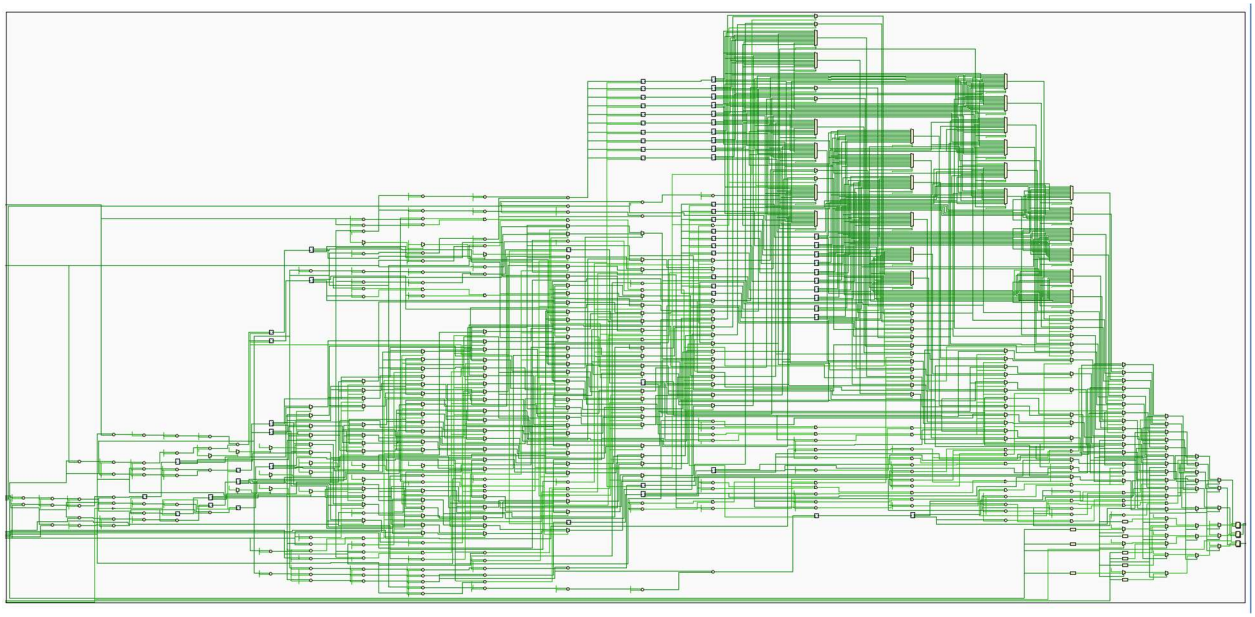
Controllable Settings When press “1” and “2”, the speed at which the screen scrolls will change; when press “A” and “S”, the color of the sky in the background will change; when press “I”, the game will enter the invincible mode, and the words "invincible" in traditional Chinese will appear in the upper left corner of the screen. When the bird hits the pillar, it will "destroy" the pillar and the game will not end.

3. Block Diagram

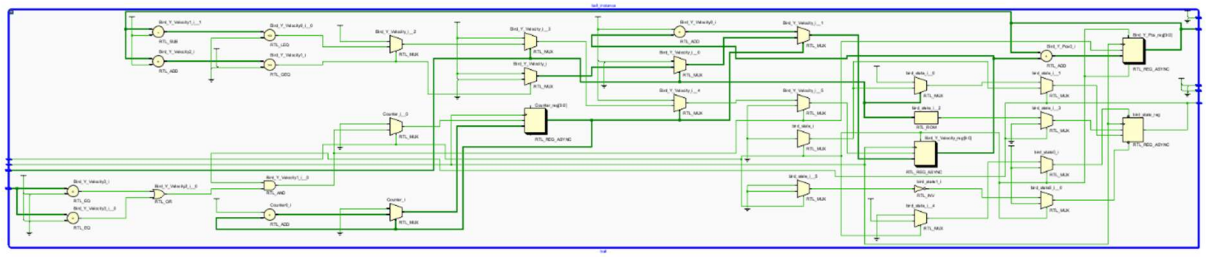
Overall:



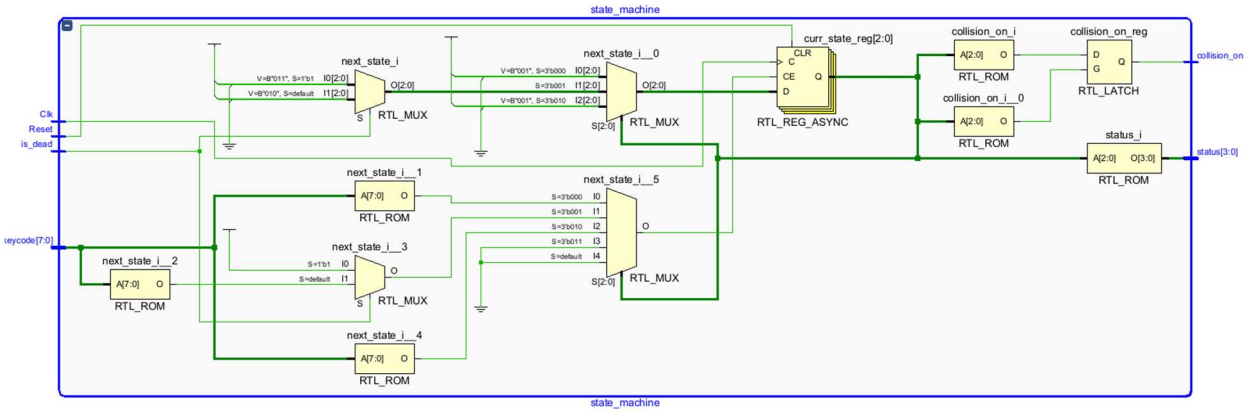
Color_Mapper:



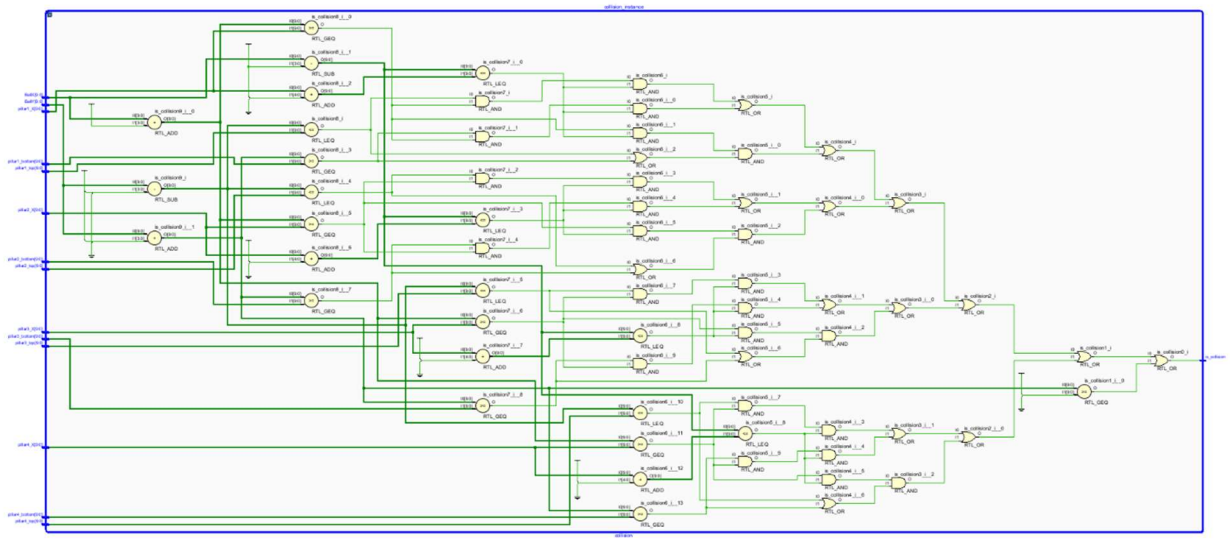
Ball:



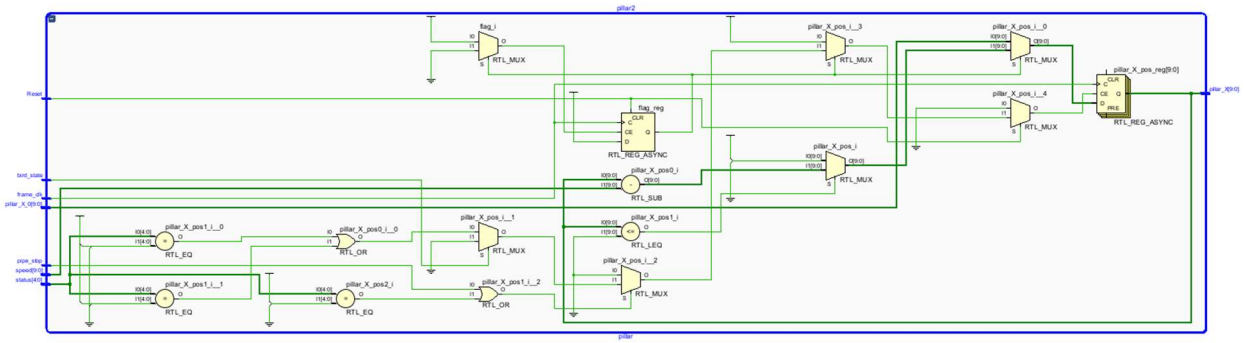
State_machine:



Collision_check:



Pillar:



4. Module Descriptions

Module: mb_usb_hdmi_top.sv

Inputs: Clk, reset_rtl_0, [0:0] gpio_usb_int_tri_i, usb_spi_miso, uart_rtl_0_rxd,

Outputs: gpio_usb_rst_tri_o, usb_spi_mosi, usb_spi_sclk, usb_spi_ss, uart_rtl_0_txd, hdmi_tmds_clk_n, hdmi_tmds_clk_p, [2:0]hdmi_tmds_data_n, [2:0]hdmi_tmds_data_p, [7:0] hex_segA, [3:0] hex_gridA, [7:0] hex_segB, [3:0] hex_gridB

Description: It functions as a central control hub within a larger system configuration, overseeing the integration of crucial components such as USB communication, HDMI video output, and VGA graphics display. This module serves as a key coordinator, ensuring the harmonious interaction between diverse subsystems and effectively managing USB input, VGA display, and HDMI output. With its dependence on a MicroBlaze block design, it showcases its ability to govern the synergy between hardware and software, making it a valuable asset for applications requiring interactive graphics and multimedia integration. In essence, "mb_usb_hdmi_top" serves as a pivotal element in the system, facilitating the implementation of sophisticated embedded systems, multimedia projects, or any scenario demanding seamless coordination between USB data exchange and video presentation.

Purpose: Its primary function is to receive input and instantiate modules. The module's timescale is defined as 1ns / 1ps. Serving as the top-level module, mb_usb_hdmi_top accepts various inputs and outputs, including clock signals, USB, UART, HDMI, HEX displays, and other control signals. The module's objective is to coordinate the interaction among these components, creating a cohesive and operational system. It incorporates instances of several modules, such as HexDriver, mb_block, clk_wiz, vga_controller, hdmi_tx_0, ball, and color_mapper. These individual modules handle specific tasks like driving HEX displays, managing USB and UART communication, generating VGA and HDMI signals, controlling ball movement, and color mapping. Overall, the top-level module ensures the seamless collaboration of these components to achieve the intended system functionality.

Module: hex.sv

Inputs: clk, reset, [3:0] in[4], [3:0] nibble,

Outputs: [7:0] hex_seg, [3:0] hex_grid, [7:0] hex

Description: This digital hardware component is crafted to facilitate the presentation of hexadecimal values on HEX displays. It receives various inputs, including a clock signal (clk), a reset signal (reset), and a 4-bit nibble (in[4]). The module produces two primary outputs: hex_seg, indicating the segments to be activated on the HEX display to represent the hexadecimal value, and hex_grid, defining the grid configuration for the HEX display. To accomplish this, HexDriver internally incorporates another module known as nibble_to_hex. This internal module is responsible for converting a 4-bit binary nibble into an 8-bit hexadecimal representation.

Purpose: Its aim is to offer a user-friendly and effective method for showcasing hexadecimal values on HEX displays within digital systems or FPGA-based designs. Taking a 4-bit binary input, the module transforms it into the relevant hexadecimal representation and proceeds to allocate the segments on the HEX display to visually depict the hexadecimal digit. This proves beneficial in situations demanding the visualization of hexadecimal values, including applications like debugging tools, digital counters, or any context requiring the communication of hexadecimal information. The HexDriver module simplifies the portrayal of hexadecimal values, contributing to the improved usability and clarity of digital systems.

Module: VGA_contrtoller.sv

Inputs: pixel_clk, reset,

Outputs: hs, vs, active_nblank, sync, [9:0] drawX, drawY

Description: This serves as a crucial element in digital systems, particularly in applications involving the generation of video signals for VGA (Video Graphics Array) displays. Functioning as a video signal generator, its role is to create the required timing signals for driving a VGA display. The module is designed to produce horizontal sync (hs), vertical sync (vs), and active/blanking interval signals (active_nblank) in accordance with VGA display standards. These signals play a crucial role in coordinating the presentation of graphics and images on a VGA monitor. Taking input signals such as the pixel clock (pixel_clk) and reset, the module generates key timing signals controlling the position of the display cursor (drawX and drawY) and the synchronization signals (hs and vs). It is tailored to be compatible with VGA displays, supplying the essential signals necessary for generating a stable and properly synchronized video output.

Purpose: Its primary function is to conform to VGA display timing standards, ensuring accurate synchronization and the proper display of graphics on a VGA monitor. It processes input signals like the pixel clock and reset, generating horizontal and vertical synchronization pulses while determining the active video area, dictating where pixels should appear. This module holds significance in various applications, including gaming, image processing, and video streaming, where a visual output is imperative. By accurately producing VGA timing signals, this module facilitates the integration of digital systems with VGA monitors, enabling users to visualize graphical content on standard VGA displays. Its purpose lies in providing an essential component for creating and managing VGA video output, establishing its critical role in numerous FPGA-based systems requiring visual representation.

Module: Ball.sv

Inputs: Reset, frame_clk, status, [7:0] keycode,

Outputs: [9:0] BallX, BallY, BallS

Description: It is crafted to oversee the dynamics of a bird-like entity within a digital game or simulation, taking charge of its position and state on the display screen. Key parameters include defining the bird's central, minimal, and maximal positions on the X and Y axes, along with specifying its step size in these directional dimensions. Additional factors, such as the bird's size, constants for gravity, ball width, and height, are also established. The module's primary logic, encapsulated within an always block triggered by a rising edge of frame_clk or Reset, governs the bird's velocity and position by considering both external inputs and internal conditions. This encompasses the adjustment of the bird's velocity and position, responsiveness to key presses (e.g., 8'h2C for jumping), and the management of vertical movement through a rudimentary gravity simulation. Notably, the bird's horizontal position remains fixed at the center, while its vertical placement dynamically evolves based on velocity and interactions with screen boundaries.

Purpose: It orchestrates the movement and behavior of a bird-like object in the context of a digital game or simulation. By defining crucial parameters and implementing logical operations, the module ensures the accurate portrayal of the bird's position and state on the display screen. It responds to user inputs and internal conditions to simulate realistic behaviors, including vertical movements influenced by gravity and horizontal stability at the screen center. This module serves as a fundamental building block for creating interactive and visually engaging digital games or simulations, offering precise control over the bird's dynamics within the defined constraints of the display environment.

Module: Color_Mapper.sv

Inputs: status, [9:0] BallX, BallY, DrawX, DrawY, Ball_size,

Outputs: [3:0] Red, Green, Blue

Description: It generates real-time colors in a video game or simulation, specifically catering to elements such as birds, pillars, backgrounds, and scoreboards. Functioning in tandem with a VGA clock, the module processes a multitude of inputs, including object coordinates, game status, and player inputs, to dynamically produce the appropriate color for each pixel displayed on the screen. Leveraging ROMs and predefined color palettes for individual game elements, the module employs calculated ROM addresses based on object positions and screen coordinates to determine the color output. This sophisticated mechanism of dynamic color mapping contributes to a visually captivating and responsive gaming experience, where the display intelligently adjusts to the evolving game scenario and player interactions.

Purpose: It is to serve as the backbone for graphical output in video games or simulations. By intricately managing color generation for diverse in-game elements, it ensures that the display accurately reflects the positions and states of birds, pillars, backgrounds, and scoreboards. Operating in real-time with the VGA clock, the module facilitates a seamless and visually immersive gaming experience by adapting the color scheme dynamically based on the ongoing game dynamics and user inputs. Its role extends beyond mere color assignment, actively contributing to the overall engagement and responsiveness of the gaming environment, thereby enriching the user experience.

Module: pillar.sv

Inputs: Reset, frame_clk, pipe_stop, bird_state, pillar_X_0, status, speed,

Outputs: pillar_X

Description: It manages the horizontal displacement of a pillar within a graphical simulation or game. Its output, pillar_X, signifies the current X-axis position of the pillar. The module's core logic operates within an always block triggered by frame_clk or Reset, orchestrating the pillar's position. Upon a reset, it initializes both the position and a corresponding flag. During normal operation, the module dynamically updates the pillar's position by decrementing it based on the provided speed input, resetting the position when it reaches the left edge of the screen. The movement is contingent on certain conditions, including pausing when pipe_stop is true or specific status conditions are met. Movement resumes when bird_state indicates ongoing gameplay, creating the effect of a persistently moving obstacle in a side-scrolling game environment.

Purpose: It is to emulate the behavior of a continuously shifting obstacle within the context of a side-scrolling game or graphical simulation. By processing inputs like Reset, frame_clk, and various status indicators, the module efficiently manages the dynamic positioning of the pillar along the X-axis. This controlled movement introduces challenges for the player, as the pillar's position is intricately tied to the game's ongoing status and user interactions. The module's design facilitates the creation of an engaging gaming environment where players navigate through a series of pillars, adding an element of skill and strategy to the overall gameplay experience.

Module: state_machine.sv

Inputs: Reset, Clk, [7:0] keycode,

Outputs: status, collision_on,

Description: It serves as a finite state machine (FSM) integral to the logical operations of a digital game. Functioning on a 50 MHz clock (Clk) and responsive to an active-high reset signal (Reset), this FSM employs keycode input to navigate through distinct game states and produces outputs indicating the game's status and a collision flag. The states within the FSM encompass START, initiating the game with an initial background display; GAME, representing active gameplay with collision detection enabled; INVINCIBLE, a unique state with collision detection deactivated; and END, signaling the conclusion of the game. Transitions between these states are contingent upon specific keycode inputs. Pressing enter (keycode 8'h28) initiates the game from the START state, while keycode 8'h0C toggles between the GAME and INVINCIBLE states. Additionally, the is_dead input prompts a transition to the END state, denoting the conclusion of the game. The status output conveys the ongoing game state, and the collision_on flag regulates collision detection, remaining deactivated exclusively in the INVINCIBLE state.

Purpose: It is to establish and regulate the finite state machine that governs the core gameplay mechanics within a digital game. Operating on a defined clock and responding to user inputs through keycodes, the module orchestrates transitions between various game states, effectively steering the flow of the game. By managing states like START, GAME, INVINCIBLE, and END, the module ensures a structured and responsive gaming experience. It allows for dynamic changes in gameplay based on user interactions, providing a foundational framework for implementing diverse game scenarios and maintaining a coherent logic throughout the gaming environment.

Module: collision

Inputs: [9:0] BallX, BallY, pillar_X, pillar_top_Y, pillar_bottom_Y,

Outputs: is_collision,

Description: It is specifically engineered to facilitate collision detection within a game environment, focusing on discerning interactions between a mobile ball entity and four stationary pillars. Employing calculated positions of the ball's edges, the module systematically compares them against the boundaries of the pillars, effectively identifying instances of collision. It encompasses distinct logic for various collision scenarios, encompassing upper, lower, and front collisions with each pillar. Additionally, the module scrutinizes whether the ball has reached the ground, defined by a predetermined Y-coordinate. The output signal, is_collision, assumes a logical state to signify the occurrence or absence of any collisions. This module plays a pivotal role in shaping gameplay dynamics by determining crucial moments when the ball engages with obstacles, thereby influencing the overall outcome of the game.

Purpose: It is to provide a robust mechanism for detecting collisions within the context of a digital game. By intricately calculating the ball's position and assessing its interactions with stationary pillars, the module effectively identifies instances of collision. The nuanced logic implemented for different collision types ensures comprehensive coverage, including upper, lower, and front collisions with each pillar, along with ground collisions. The module's output, the is_collision signal, serves as a critical feedback mechanism, conveying the occurrence or absence of collisions. This functionality is indispensable for shaping the dynamics of gameplay, enabling the game to respond dynamically to interactions between the ball and environmental obstacles, thereby influencing the game's progression and outcome.

IP: …_rom

Inputs: addra, clka,

Output: douta,

Description: It proves highly beneficial for applications demanding rapid retrieval of extensive data volumes, notably in domains such as image processing or data buffering. Its significance lies in the capability to initialize these memory blocks with pre-defined data, a process facilitated through a Coefficient (COE) file. This makes the component integral to FPGA-based systems where the initial states of data play a pivotal role, especially in applications like image processing, lookup tables, or firmware storage.

Purpose: The essence of this component is best described in its application across various scenarios that necessitate swift access to substantial data quantities. Its unique ability to be configured with predetermined data, facilitated by the utilization of a Coefficient (COE) file, positions it as a crucial element in FPGA-based systems. This becomes particularly apparent in contexts such as image processing, lookup tables, or firmware storage, where the establishment of initial data states holds paramount importance for the effective

functioning of the system.

AXI Interconnect: The AXI interconnect stands as a high-performance and extensively customizable interface linking numerous AXI masters, such as CPUs and DMA controllers, to multiple AXI slaves like memory and peripherals. It oversees the efficient exchange of data and control signals between these masters and slaves, ensuring optimal communication.

MicroBlaze: MicroBlaze represents a soft microprocessor core developed by Xilinx for integration into FPGAs. Operating on a RISC architecture, it can be tailored to meet specific application needs, commonly serving to oversee and manage diverse tasks within an FPGA design.

VGA Controller: Tasked with generating video signals compatible with VGA monitors, the VGA controller orchestrates horizontal and vertical synchronization, pixel data, and other timing signals to produce a display on monitors that adhere to VGA standards.

HDMI Interface: The HDMI interface facilitates the transmission of audio and video signals to HDMI-compliant devices like high-definition TVs and monitors. Comprising HDMI transmitters and receivers, it facilitates the seamless exchange of multimedia data.

USB Interface: Responsible for connecting the FPGA to USB devices, the USB interface enables data transfer between the FPGA and USB peripherals such as keyboards, mice, or USB storage devices.

UART Interface: Functioning as a hardware module for serial communication, the UART interface plays a vital role in linking the FPGA to devices communicating through serial data.

Clock Management: These modules take charge of clock signals, encompassing tasks such as clock generation, distribution, and synchronization. Their role is to ensure that all components within the design are synchronized to a common clock domain.

Memory: Serving as repositories for data and code, memory modules encompass on-chip block RAM and so on.

GPIO (General-Purpose Input/Output): GPIO modules offer digital input and output pins, facilitating the connection of the FPGA to external components and peripherals. These pins serve diverse purposes, including button press reception and interfacing with other digital devices.

DMA Controller: Enabling efficient data transfer between different system components, such as memory and peripherals, the DMA controller operates without involving the CPU. It enhances data throughput and can alleviate CPU load.

Interrupt Controller: Responsible for managing the assignment and handling of interrupt requests from various sources, the interrupt controller ensures that the CPU promptly responds to critical events.

Serial Peripheral Interface (SPI) Interface: Utilizing the SPI protocol, this interface facilitates synchronous serial communication, commonly employed for connecting digital sensors, memory devices, and other peripherals to the FPGA.

5. Design Procedure

The Final Project builds upon the foundation laid during Lab 6.

In the initial two weeks, we achieved the following key features:

-- Implementation of the space bar functionality, enabling users to control the bird for upward jumps.

-- Incorporation of gravity to simulate realistic bird movement.

Moving on to the subsequent two weeks, we successfully implemented a comprehensive set of functions, including:

-- Integration of sprite mapping for various elements such as the background, pillars, birds, and scoreboard.

-- Implementation of a collision check mechanism to enhance gameplay dynamics.

-- Introduction of opening and game over screens, providing a structured game flow.

-- Implementation of a scoring function to keep track of player performance.

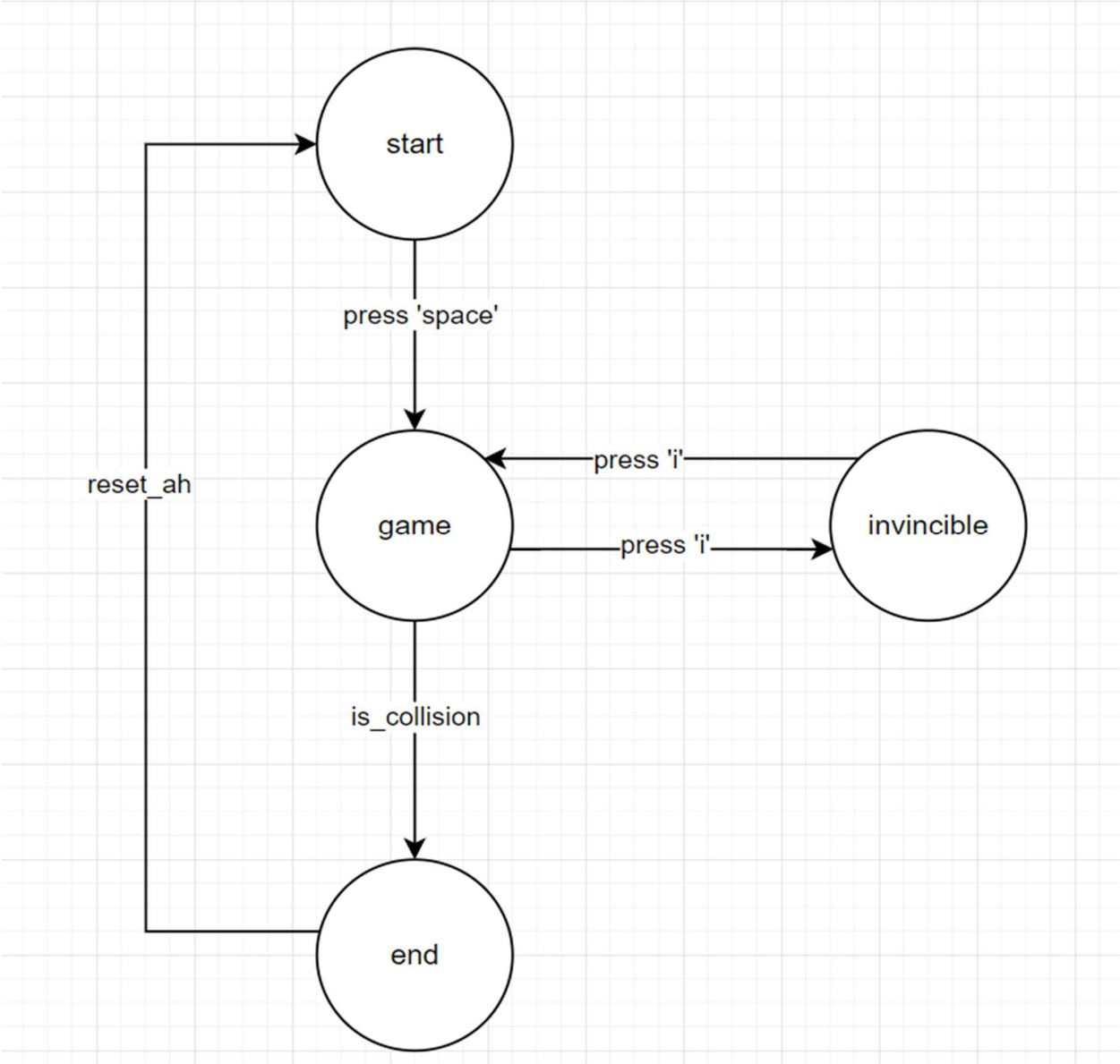
-- Integration of background movement for a visually engaging gaming experience.

-- Development of an algorithm for determining column height, contributing to the game's overall challenge.

-- Implementation of a bird flapping its wings animation, enhancing the game's visual appeal.

-- Introduction of controllable settings, allowing users to customize the game environment according to their preferences.

6. State Diagram



7. Design Statistics

LUT	8089
DSP	21
Memory (BRAM)	73.5
Flip-Flop	3108
Frequency	55.54MHz
Static Power	0.079W
Dynamic Power	0.429W
Total Power	0.508W

8. Some known bugs

When mapping, there is a straight line on the left side of the upper and lower pillars that cannot be eliminated. It may be fixed with a more refined cutout.

9. Conclusion

The Flappy Bird project serves as a comprehensive and well-executed endeavor, providing an entertaining and customizable gaming experience on the Urbana board. The inclusion of additional features and attention to detail in module design contribute to the project's success in bringing the iconic game to FPGA-based hardware.