ECE 385

Fall 2023

Final Project Report

Design and Implementation of The King of Fighters '94 game on an FPGA with MicroBlaze CPU

Yuqi Wen, Yunxuan Yang
2023/12/12

1. Introduction

This project aims to recreate the classic arcade game, King of Fighters '94, on a Spartan 7 FPGA platform. The objective is to design a digital system that can accurately emulate the game's hardware and software environment. This includes graphics rendering, sound processing, game logic, and control mechanisms. By using an FPGA, the project seeks to explore hardware/software co-design principles, emphasizing real-time processing and hardware emulation.

The primary focus was on integrating advanced peripherals with the Microblaze-based system-on-chip (SoC) on the Spartan 7 FPGA. Specifically, a USB host controller was interfaced with the Microblaze CPU to communicate with a USB keyboard for user input. Concurrently, a VGA signal was converted to HDMI, enabling the display of graphical content on an HDMI monitor.

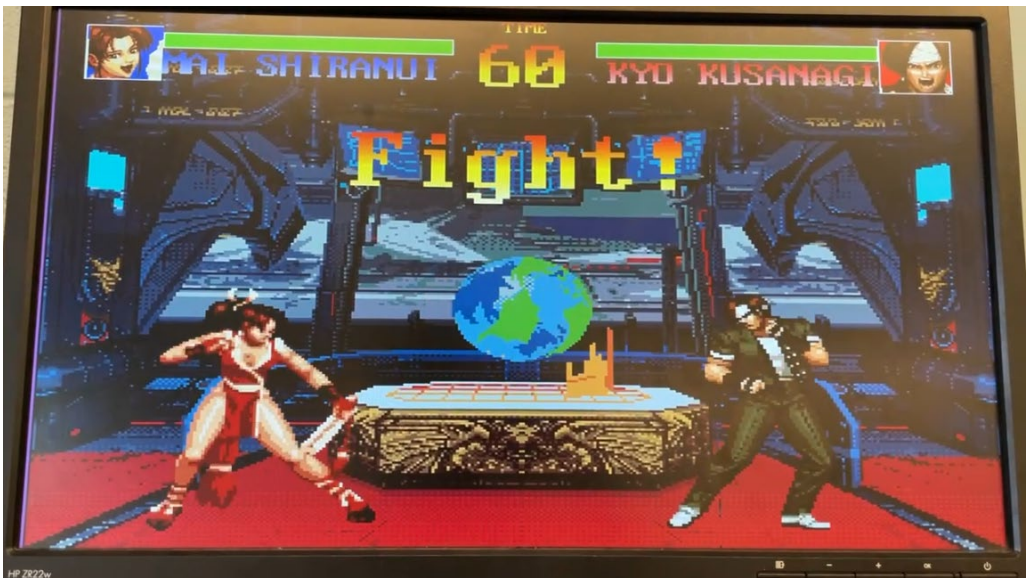


Figure 1: game display

2. Written Description of System

1) Drawing

Drawing of sprites is a critical aspect of the game's visual presentation, bringing characters and environments to life with vivid detail and animation. Our approach to sprite drawing focuses on accurately representing the iconic look and feel of the original King of Fighters series, while adapting to the technical constraints of our development platform.

The sprites for characters and backgrounds are meticulously designed and rendered. Each character sprite consists of multiple frames, allowing for smooth and realistic animations that reflect various actions like walking, jumping, kicking, and punching. These frames are carefully sequenced to create fluid movements, enhancing the overall gaming experience.

Figure 2: character 1 punch sprite sheet

Above is a sample sprite sheet for punch action of character 1. To create this sheet, we first obtained single sprites from online resources (https://www.spriters-resource.com/playstation_2/thekingoffighters94rebout/), then used image processing tool to put the sprites together into one sheet. After that we used our python scripts to set the background color of sprite sheets into pink for later use in drawing transparent background. Later, we used the sprites to COE tool written by one of the CAs (https://github.com/amsheth/Image_to_COE) and modified the palletizer to reach our needs to generate COE files and loaded them to BRAM. In the hardware logic, we calculated the offset of sprites to draw on the screen, and used a counter to loop over the frames, enabling smooth visual effects.

Background sprites are created in the similar way. There are several steps we figured out to enable background scrolling and animation. First, we made the background image with 512*240 size, and then doubled it in the hardware logic to get a 1024*480 image. Since the VGA screen size is 640*480, we can scroll the background left and right corresponding to the movement of two players. To make the animation, we created a sprite sheet of the middle part of the background and employed the similar logic of character sprites drawing.

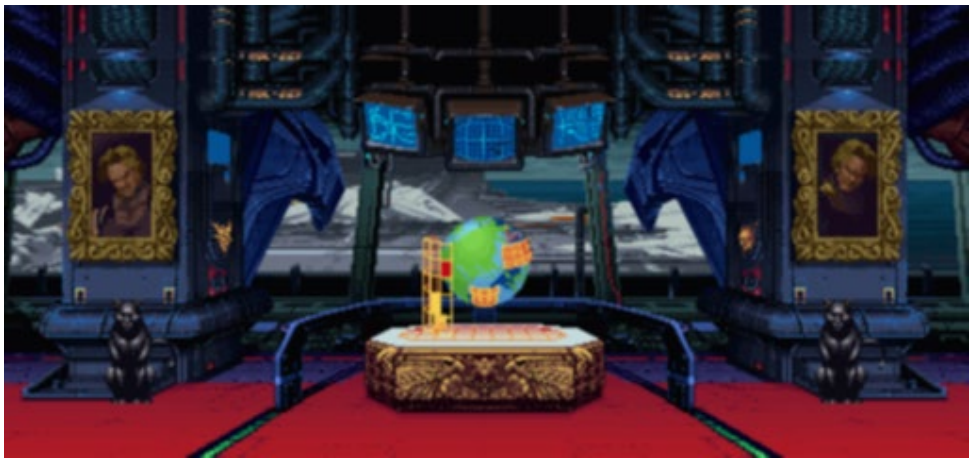


Figure 3: original background image

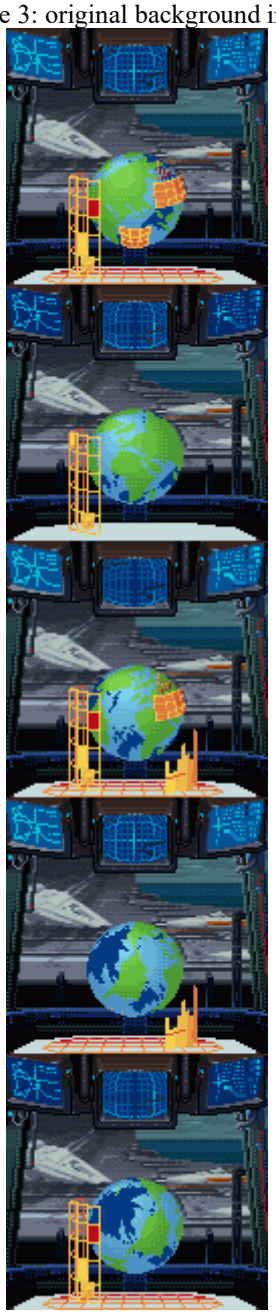


Figure 4: background animation part sprite sheet

One of the challenges we navigated was optimizing sprite resolution and memory usage. Due to memory limitations, we had to balance the quality and quantity of sprites, ensuring that each sprite is visually appealing without exceeding the memory constraints of our hardware.

2) Player Control

In our game, player control is designed to be intuitive and responsive, closely mirroring the classic King of Fighters gameplay experience while accommodating the limitations of our development environment. Players interact with the game using a set of defined controls that manage the movements and actions of their chosen characters in the game arena.

Each player has a series of basic controls that include directional inputs for movement (forward, backward, jump, and squat) and action buttons for different combat moves (punch and kick). Player 1 uses A, D to move forward and backward, W to jump, S to squat, and J K to kick and punch. Player 2 uses Left Right to move forward and backward, Up to jump, Down to squat, and 1 0 on the numpad to kick and punch.

With the limitation of on chip memory and the consideration of smoothness of movement, we can only implement basic punch and kick in our game. In order to have a better gameplay experience, we have to give up other movements.

3) Health

Each character in the game is assigned a health bar displayed at the top of the screen. This health bar visually represents the character's current health status, decreasing in response to received attacks. The health value is quantified, typically starting at a set point value (e.g., 100) and decreasing with each hit taken.

Critical to our health system is the concept of fairness and balance. For character 1 the hitbox of punch is wider than character2, while the kick of character2 is wider than character 1. Each character's attacks are calibrated to ensure no unfair advantage is given to any single character, maintaining competitive integrity.

When one player’s health goes to zero, the game ends.

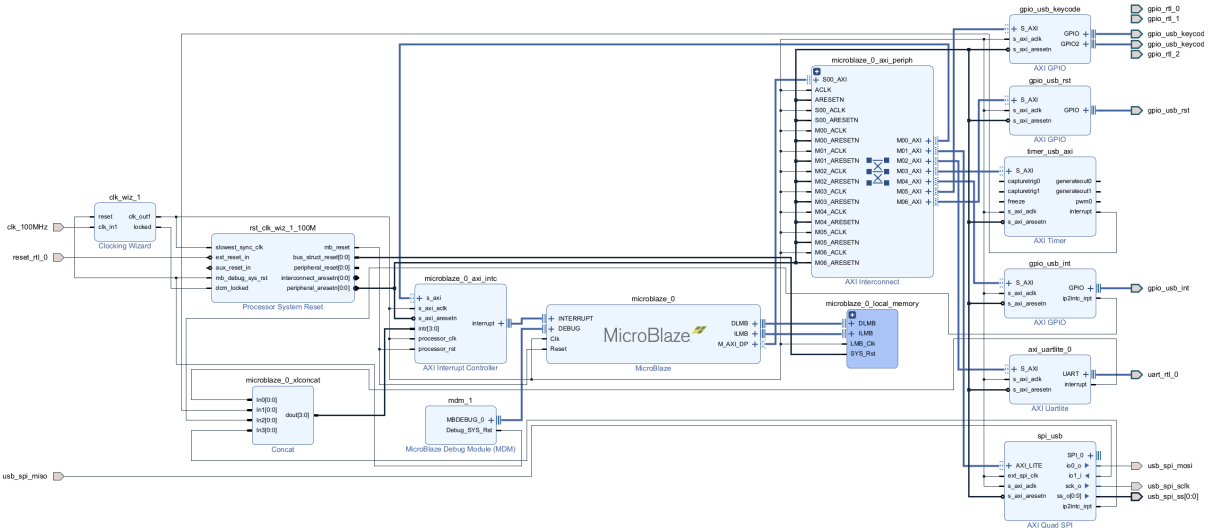
4) Timer

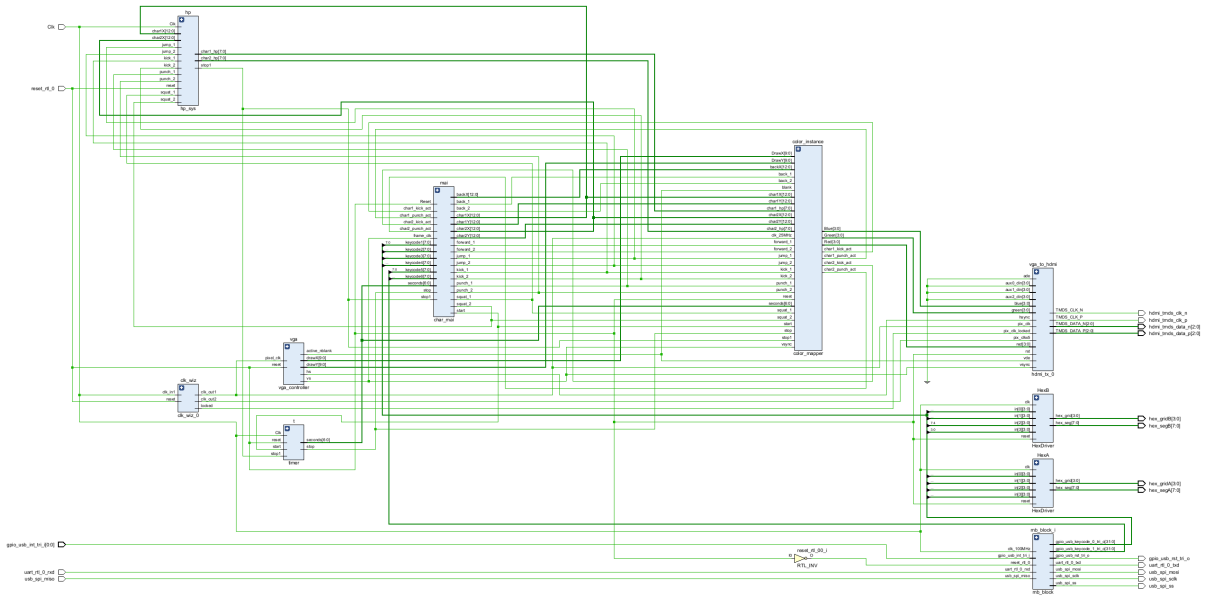
At the start of each round, the timer is set to 60 seconds. This countdown is prominently displayed on the screen, at the top, where it's easily visible to both players. As the round progresses, the timer counts down, with each second ticking away in real-time.

The presence of the timer compels players to actively engage and make strategic decisions within a limited timeframe. Players must balance their offensive and defensive strategies, knowing that the round could end when the timer reaches zero.

This encourages a more dynamic and active gameplay, as opposed to a purely defensive or cautious approach. If the two players' health are the same when time ends, the game result will be determined as draw.

3. Block Diagram





4. Module descriptions

Module descriptions:

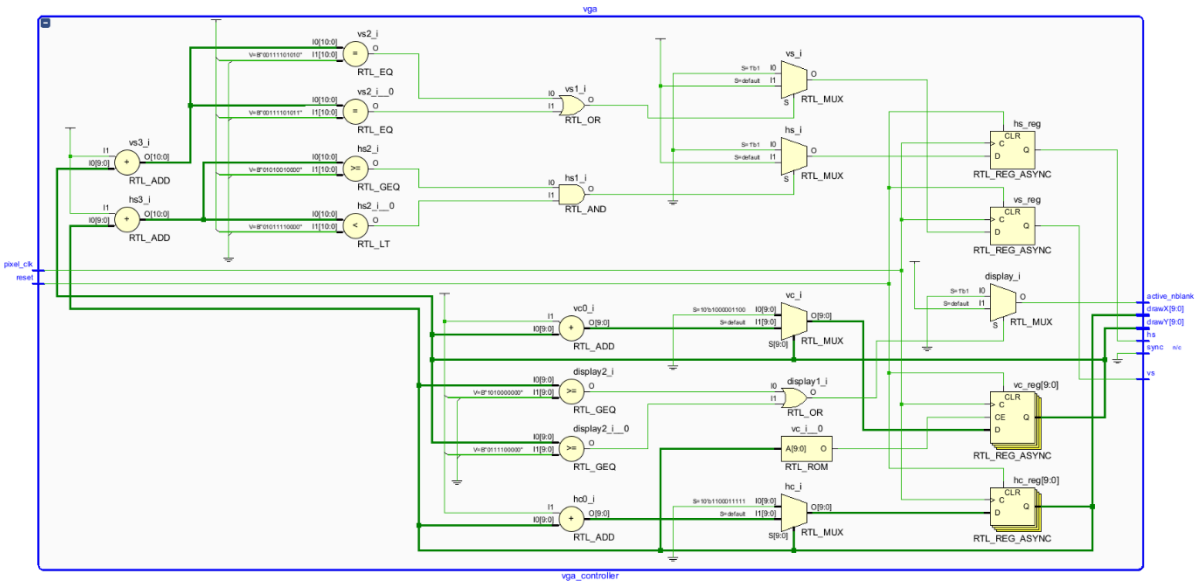
1. Module: VGA_Controller.sv

Inputs: pixel_clk, reset

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

Description: This module produces X, Y counters along with H, V sync signals. The blank and sync are left untouched.

Purpose: It's set up to display 640x480 pixels.



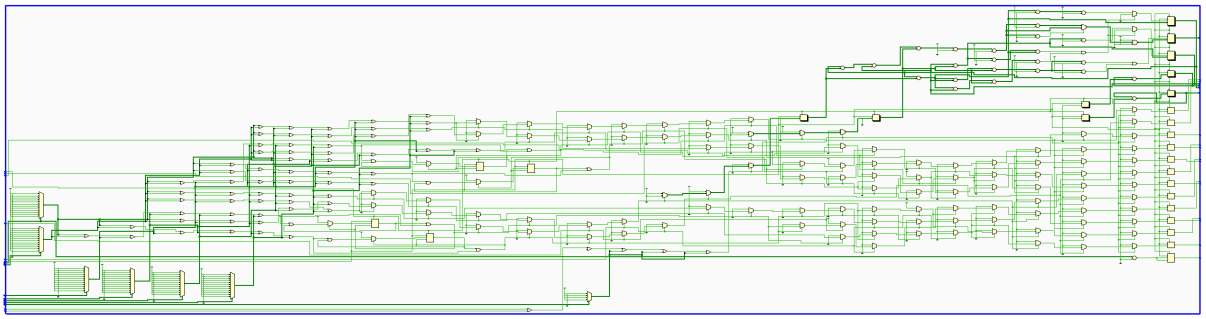
2. Module: char1.sv

Inputs: Reset, frame_clk, stop, stop1, [7:0] keycode1, keycode2, keycode3, keycode4, keycode5, keycode6, char1_punch_act, char1_kick_act, char2_punch_act, char2_kick_act, seconds,

Outputs: [12:0] char1X, char1Y, char2X, char2Y, backX, forward_1, back_1, punch_1, squat_1, kick_1, jump_1, forward_2, back_2, punch_2, squat_2, kick_2, jump_2, start

Description: This module updates characters' position and action in every frame. Reacts to keystrokes W, S, A, D, J, K, Up, Down, Left, Right, 0, 1.

Purpose: The two characters' instance



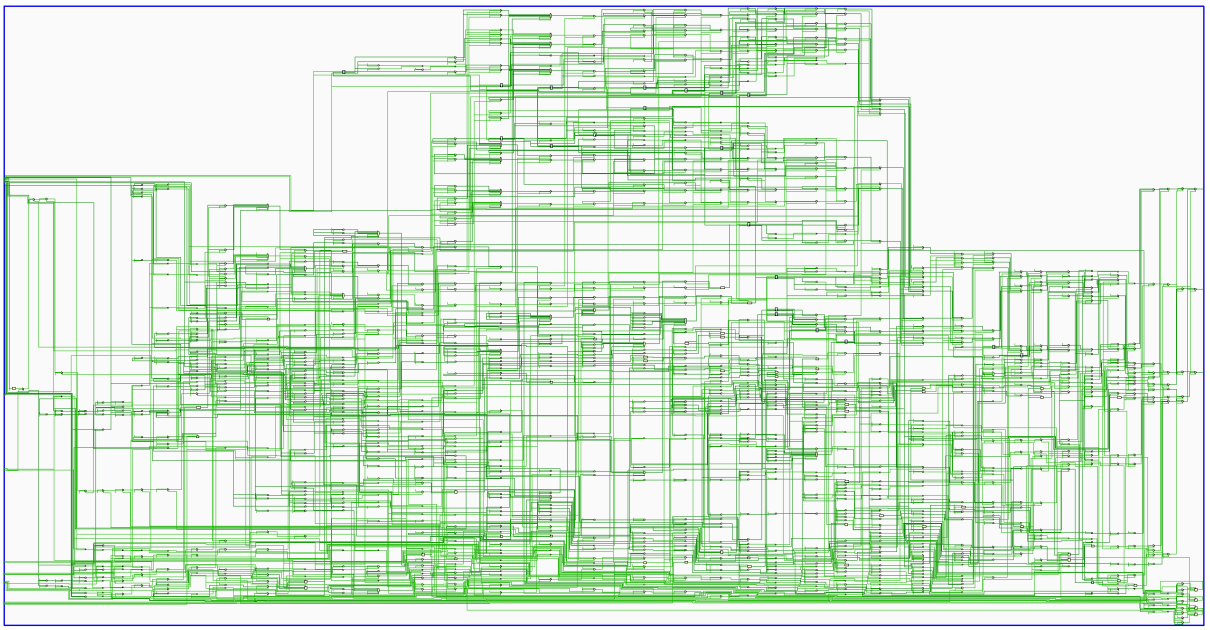
3. Module: Color_Mapper.sv

Inputs: [9:0] DrawX, [9:0] DrawY, clk_25MHz, blank, reset, vsync, [12:0] char1X, char1Y, char2X, char2Y, backX, [6:0] seconds, forward_1, back_1, punch_1, squat_1, kick_1, jump_1, forward_2, back_2, punch_2, squat_2, kick_2, jump_2, stop, stop1, start, [7:0] char1_hp, char2_hp

Outputs: [3:0] Red, [3:0] Green, [3:0] Blue， char1_punch_act, char1_kick_act, char2_punch_act, char2_kick_act

Description: Translate the complex interplay of the game's logic, character actions, and environmental states into vivid visual representations on the screen. A key feature of this module is its use of counters to draw sprites. These counters track the progress of character animations and background movements, ensuring that each sprite is rendered at the correct time and position. This mechanism allows for smooth and responsive animation, crucial for a dynamic gaming experience. The module also utilizes ROM addresses to retrieve sprite data, dynamically selecting and displaying the appropriate sprites based on the game's current state and character actions.

Purpose: Designed to handle the critical task of graphics rendering by generating RGB color outputs for the game's display.



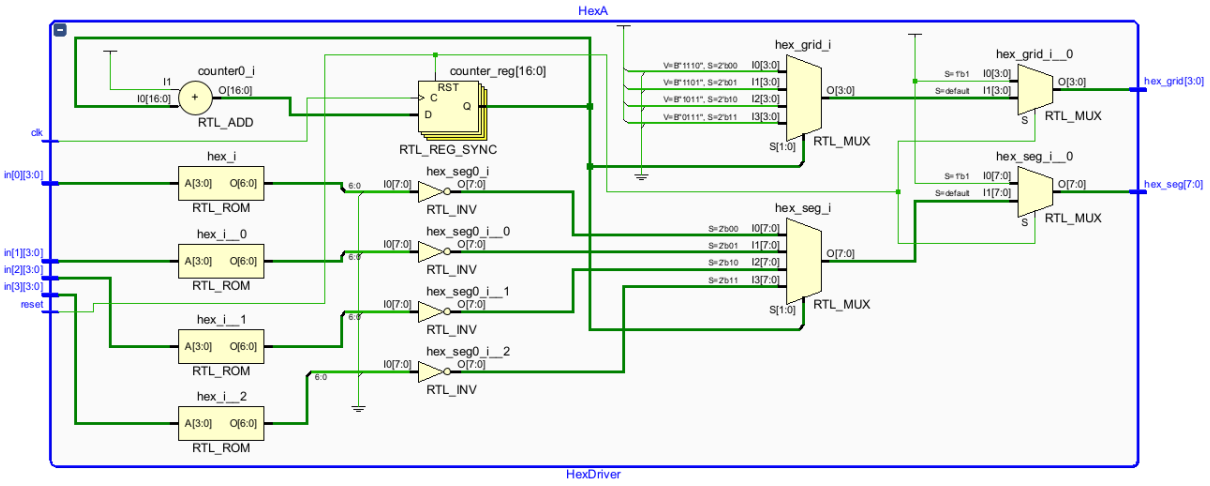
4. Module: HexDriver.sv

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

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

Description: This module converts 4bit binary numbers from registers into hexadecimal to display digit on LED.

Purpose: Set the 7-segment LED display on board.



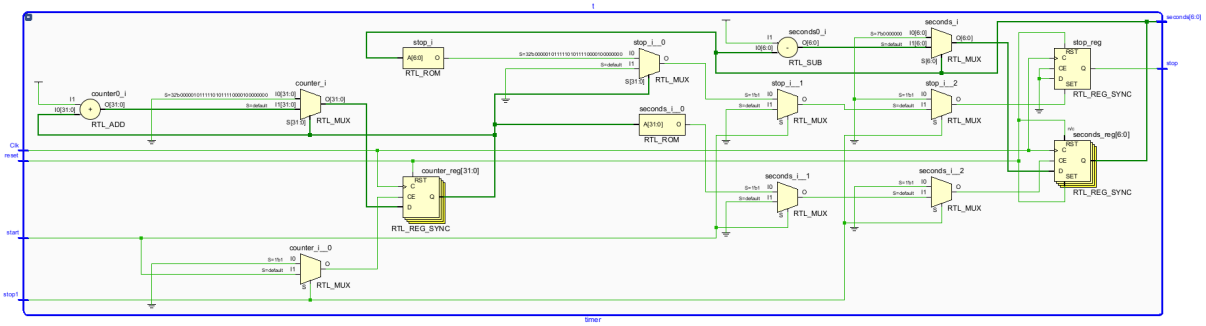
5. Module: timer.sv

Inputs: Clk, reset, stop1, start,

Outputs: [6:0] seconds, stop

Description: This is a countdown timer for the game, using a 100 MHz clock input. It features inputs for reset, start, and stop, and outputs a 7-bit seconds count and a stop signal. Internally, it uses a 32-bit counter to track time, decrementing the seconds count every second and signaling when 60 seconds have elapsed.

Purpose: 60s timer for the game



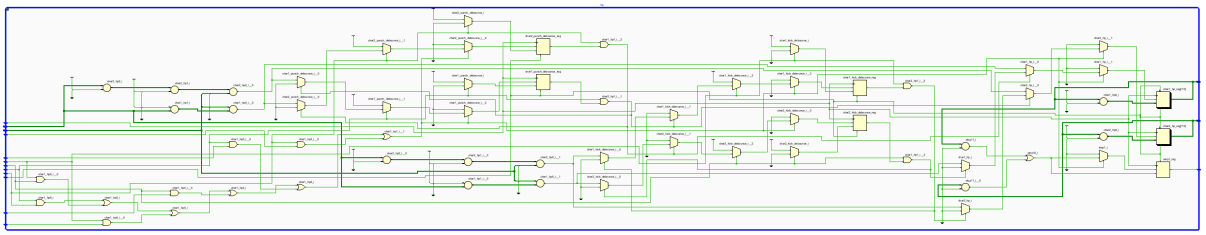
6. Module: hp.sv

Inputs: Clk, reset, [12:0] char1X, char2X, punch_1, squat_1, kick_1, jump_1, punch_2, squat_2, kick_2, jump_2,

Outputs: [7:0] char1_hp, char2_hp, stop1

Description: The module tracks and updates the HP of each player based on received attack signals.

Purpose: Health system for each player.



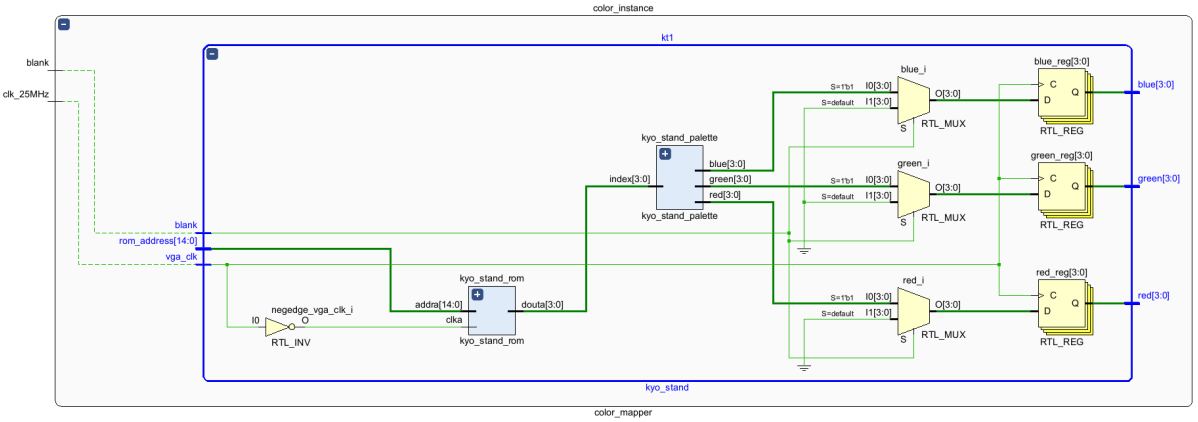
7. mai_stand.sv, mai_forward.sv, mai_backward.sv, mai_jump.sv, mai_squat.sv, mai_kick.sv, mai_punch.sv, mai_hit.sv, mai_win.sv, kyo_stand.sv, kyo_forward.sv, kyo_backward.sv, kyo_jump.sv, kyo_squat.sv, kyo_kick.sv, kyo_punch.sv, kyo_hit.sv, kyo_win.sv, scene1.sv, bg1.sv, bg2.sv, bg3.sv

Inputs: vga_clk, [15:0] rom_address, blank

Outputs: [3:0] red, green, blue

Description: This is the module to read COE data, connecting to color mapper to draw sprites

Purpose: Read COE data



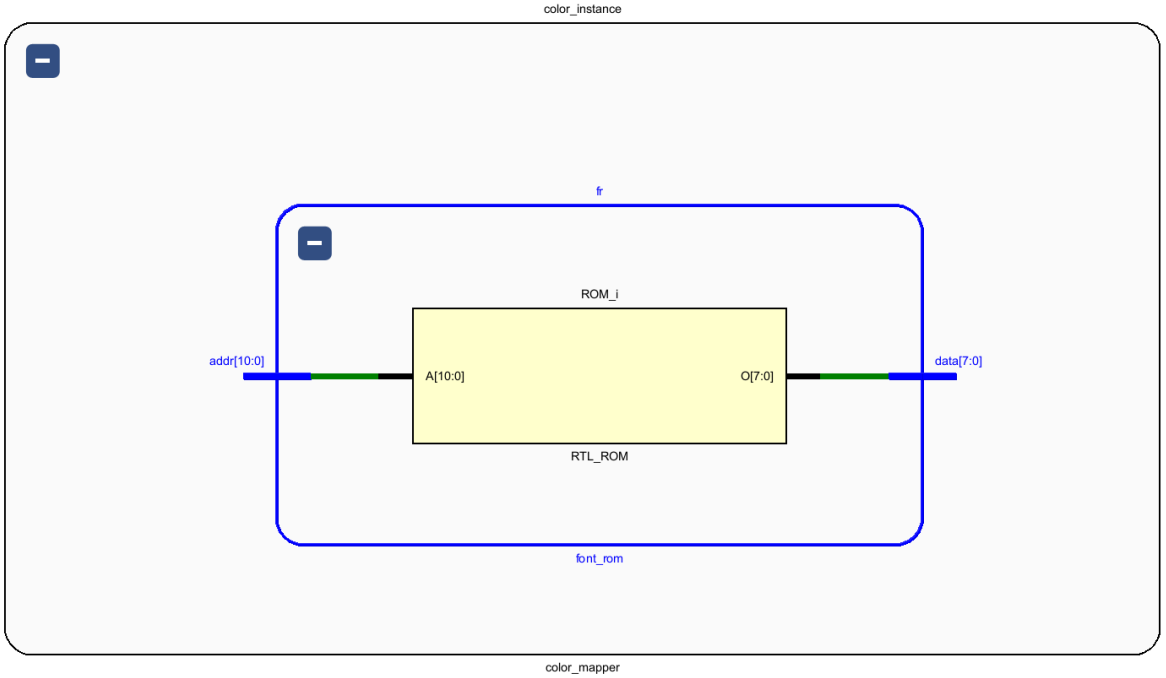
8. font_rom.sv

Inputs: [10:0] addr

Outputs: [7:0] data

Description: This is the module to read font data, connecting to color mapper to draw contexts

Purpose: Read font data



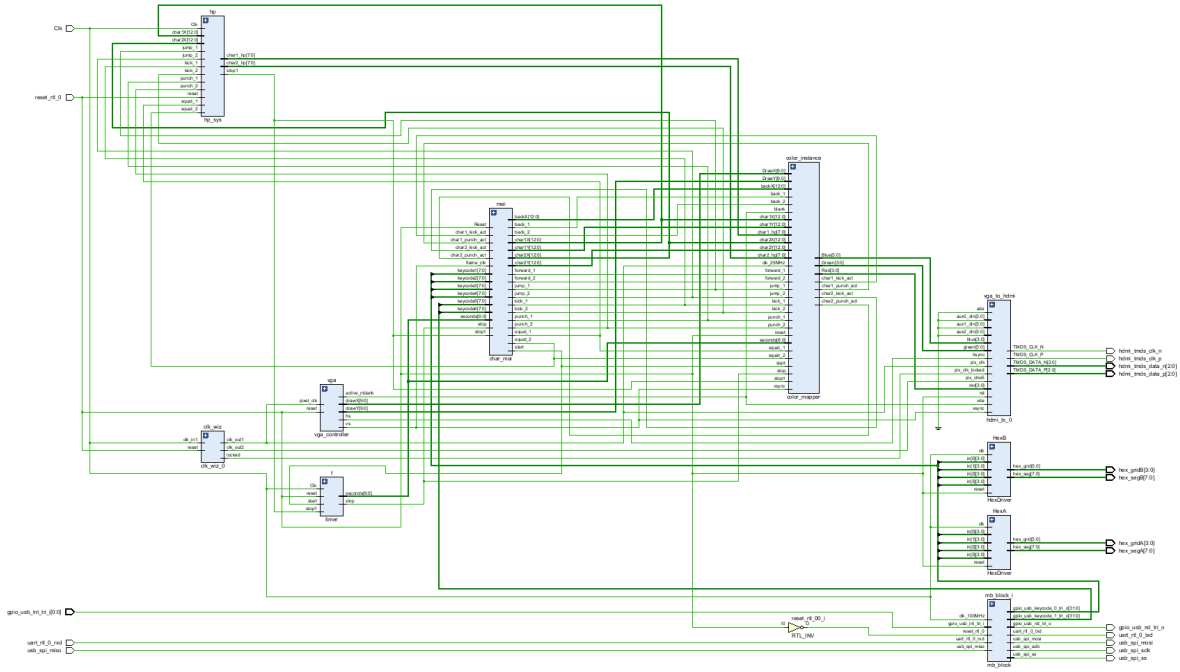
9. Module: mb_usb_hdmi_top.sv

Inputs: Clk, reset_rtl_0: 1-bit, 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, [3:0] hdmi_tmds_data_n, [3:0] hdmi_tmds_data_p, [8:0] hex_segA, [8:0] hex_segB, [4:0] hex_gridA, [4:0] hex_gridB

Description: This module is designed to manage the integration and communication between various sub-modules in the project. It facilitates the interaction of the MicroBlaze block with peripheral devices through USB, UART, and HDMI interfaces. The module also generates necessary synchronization and control signals for VGA to HDMI conversion, hex display control, and gameplay mechanics. Additionally, it plays a crucial role in rendering graphics and managing game logic, contributing significantly to the overall functionality of the gaming project.

Purpose: This is the core of the game project, serving as the top-level unit.



5. Design Resources and Statistics in table

LUT	12351
DSP	19
Memory	74
Flip-Flop	3329
Latches	0
Frequency	60.76 MHz
Static Power	79 mW
Dynamic Power	442 mW
Total Power	521 mW

6. Conclusion

As we conclude our KOF94 project, we take a moment to reflect on the journey and the achievements of our team. This endeavor, inspired by the classic King of Fighters series, was an ambitious undertaking that presented us with numerous challenges and learning opportunities. Through the collaborative efforts of our team and the guidance of our course assistants, we successfully created a functioning version of KOF94, albeit with

some simplifications in terms of attack complexity.

Over the course of four weeks, we encountered various technical hurdles, from intricate bugs to hardware limitations. These challenges often constrained our ability to add more features or refine the game further. One of the key features we intended to implement was round tracking, which, due to time constraints, had to be set aside. Nonetheless, the game operates effectively without it, providing a satisfying experience that captures the essence of the original KOF series.

A significant limitation we faced was the maximum capacity of on-chip memory, which restricted our ability to include a broader range of sprites and animations. While optimizing screen resolutions for sprites and backgrounds was a potential solution, the progress we had already made with the sprite images meant reworking them would be impractical. Additionally, we addressed a timing issue causing visual artifacts, which improved the animations' smoothness at the cost of longer compile times.

The experience and skills gained from our previous coursework were invaluable in navigating these challenges. They enabled us to adapt and apply creative solutions to complex problems. Despite the limitations and unimplemented features, our project stands as a testament to our team's resilience, adaptability, and technical prowess. We are proud of what we have accomplished, and this project has not only enhanced our understanding of game development but also strengthened our problem-solving and teamwork skills. Looking forward, we are excited to take the insights and experiences from this project into future endeavors in the realm of game design and development.