HDL Project: FPGA-Based Video Game Console

By: Alan O'Connell,

Cathal O'Regan,

Jamie O'Connor

Lecturer: Paddy Collins

Module: HDL Digital System Design

Date: 14/12/2025

I acknowledge, by submitting this report, that I am aware of the provisions in MTU's Student Regulations which specifically cover cheating where any attempt is made to gain unfair advantage, whether in coursework, assessments or examinations.

I further acknowledge that I understand the Policy and associated Procedures together with the Supplementary Policy and Supplementary Procedure which have been approved for this period

Contents

1. Introduction

This project involved the design of a digital hardware platform on the Digilent Basys3 FPGA board intended to support video game style applications. The aim of the project was not to design a complete or playable game, but instead to design a reusable and modular hardware system that could support different games in the future.

The platform makes use of the Basys3 VGA output for displaying graphics on an external monitor and several Digilent PMOD modules to provide user input and non-volatile storage. The overall system is split into separate functional blocks, with each team member responsible for a specific part of the design. This modular approach makes the system easier to understand, discuss, and extend, while also reflecting how larger digital systems are typically designed in practice.

2. Project Objectives

The main objective of this project was to design a modular FPGA-based digital system capable of supporting a video game style application. This included generating VGA-compatible video output, interfacing with multiple external input devices using PMOD modules, and providing non-volatile memory for storing user or system data. A further objective was to apply system-level design techniques and demonstrate effective teamwork by dividing the project into clear functional blocks and integrating them into a single overall design.

3. Overall System Overview

3.1 System Description

At a high level, the system is made up of a central system controller connected to several peripheral subsystems. These include a VGA display controller, input and output interfaces for a keypad and joystick, and a serial flash memory interface. All blocks are designed to operate synchronously using a shared system clock derived from the Basys3 on-board 100 MHz oscillator.

The system is designed to be game agnostic. This means that different games could be supported by changing the control logic and display behaviour without needing to redesign the underlying hardware interfaces. This approach supports the overall goal of creating a flexible platform rather than a single fixed application.

3.2 Top-Level Architecture

The top-level architecture integrates the system controller, VGA display controller, input and output interfaces, and the serial flash memory interface. The system controller acts as the central coordination block within the design. It receives processed input data from the input subsystem, manages system states, controls access to memory, and determines what information is passed to the VGA controller for display.

This structure allows each subsystem to focus on its own responsibilities while still working together as part of a complete system.

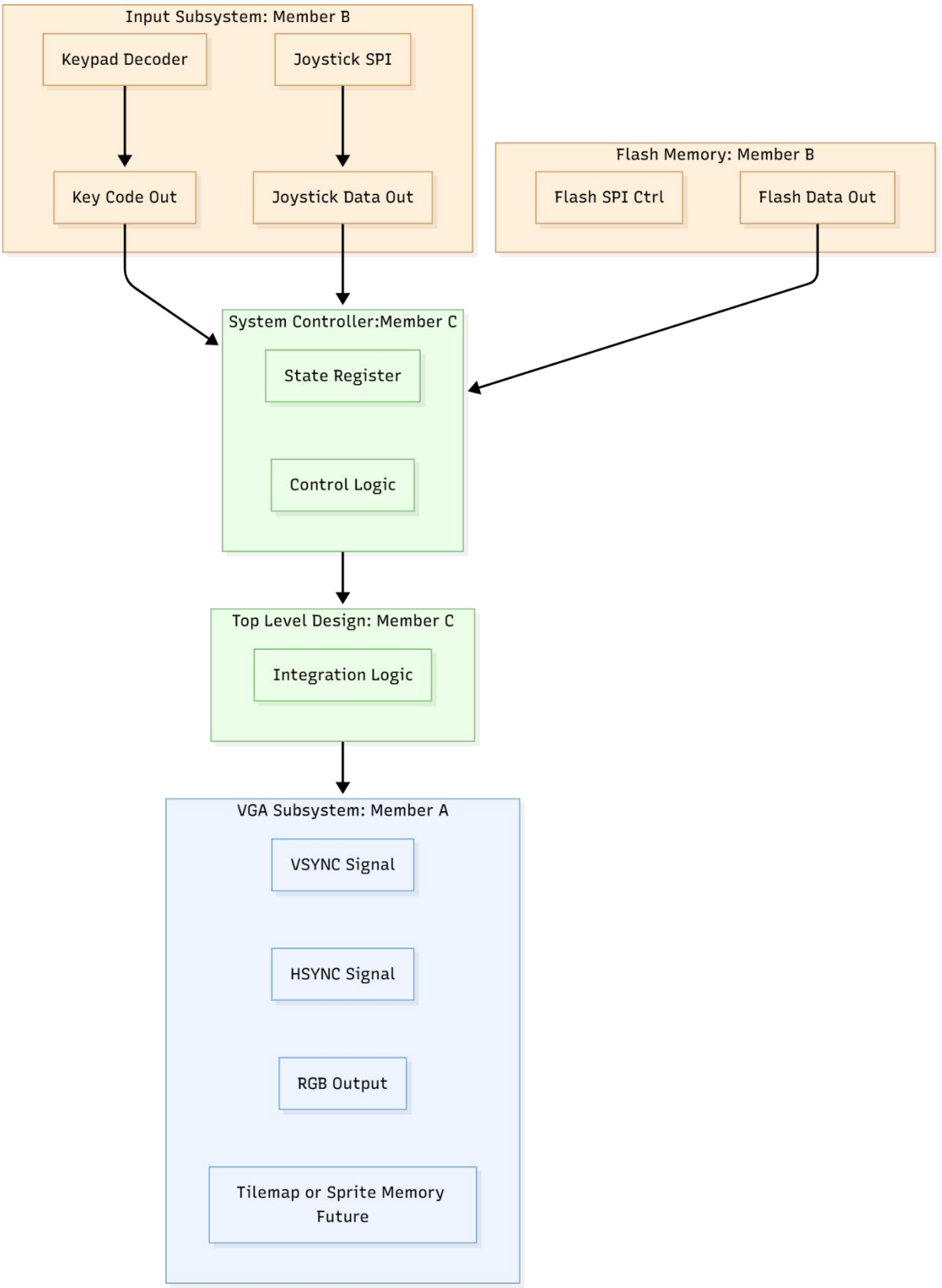

Figure 1 : Project Top Level Block Diagram

4. Hardware Platform

4.1 Basys3 FPGA Board

The Digilent Basys3 board is based on a Xilinx Artix-7 FPGA and provides a 100 MHz on-board clock, a VGA output connector, multiple PMOD headers, and on-board switches, buttons, and LEDs. These features make it suitable for implementing and testing digital systems with both input and output requirements.

4.2 Clocking and Reset Strategy

A single global clock is distributed to all blocks in the system. Where required, clock division is used to generate slower clocks such as the VGA pixel clock. A global synchronous reset signal is used to ensure all blocks enter a known state on reset.

5. Functional Block Descriptions

5.1 VGA Display Controller

Responsible: Cathal

The VGA display controller generates the horizontal and vertical synchronisation signals and the RGB colour signals required to drive an external VGA monitor. The controller is designed to support a resolution of 640 × 480 at 60 Hz, which requires a 25 MHz pixel clock. Horizontal and vertical counters are used to track the current pixel position on the screen. These counters determine when the controller is within the active display region and when synchronisation signals should be asserted. Pixel colour data is provided by the system controller, allowing the VGA block to remain independent of specific game logic.

5.1.1 VGA Timing Requirements

The VGA display controller works by reproducing the exact timing characteristics needed for a 640 × 480 resolution at a frequency of 60 Hz. This mode was chosen because it works with a lot of different systems and is a good fit for the Basys3 hardware. The controller needs to make a pixel clock of about 25 MHz from the board's 100 MHz oscillator using a simple clock division scheme in order to meet the VGA standard. The exact specification says 25.175 MHz, but most monitors can manage the 25 MHz approximation, which keeps the display stable.

VGA video output is based on cycles of horizontal and vertical scanning. Each line has 640 active pixel periods, and then there is a break that lets the display retrace. This interval is made up of a front porch, a sync pulse, and a back porch, bringing the total horizontal cycle length to 800-pixel clocks. Vertically, a similar pattern occurs across 480 visible lines, followed by several non-visible lines that support frame retraces, resulting in a total of 525 lines per frame. To implement this behaviour, the controller uses horizontal and vertical counters that track the current position within each line and frame. The controller makes active low HSYNC and VSYNC signals during the parts of the counters that correspond to the sync pulse regions. These signals mark the lines and frames edges. These transitions happen at exactly the right time, which makes sure that the monitor locks onto the incoming signal and shows a steady image.

5.1.2 Pixel Data Handling and Colour Output

The VGA controller is in responsible for generating the timing signals and making sure that each pixel is in the right place, but it does not decide what the display shows. Instead, it gets colour values from the game logic or system controller and uses the current pixel coordinates as a guide for what should be drawn on the screen. Because of this separation of duties, the VGA controller can stay a purely timing-based part, while the higher-level modules manage the rendering behaviour that is specific to the game or menu.

The Basys3 board uses a 12-bit RGB encoding scheme to output colour, which means that each colour channel has four bits. This format works with the resistor DAC that is connected to the VGA port and is an uncomplicated way to turn digital signals into analog voltage levels. The controller sends the RGB value directly to the output pins when the pixel is in the active display area. When it is outside, the output is forced to black to make sure that colours or transitional artefacts that shouldn't be there don't show up in the frame.

The controller also makes the current x and y pixel coordinates available to the rest of the system. The game or menu logic can use these coordinates to figure out exactly what should be at that spot, like a background colour, a menu item, a sprite, or a game object. The VGA controller provides synchronised coordinate information, which serves as a basis for numerous rendering techniques without altering the fundamental timing structure.

5.1.3 Integration with system Controller

Integration of the VGA subsystem with the system controller is achieved through a clean and modular interface. The VGA block supplies continuous position information through its horizontal and vertical counters, indicating the exact pixel currently being output. The system controller uses this information to compute or retrieve the appropriate colour value for each pixel. Because the VGA controller handles timing exclusively, the system controller can focus on higher-level tasks such as menu navigation, gameplay logic, sprite behaviour, or pause-state management.

The two modules operate concurrently and synchronously, sharing the same global clock to ensure that pixel data updates align perfectly with the controller's timing signals. During the active display region, the system controller provides the RGB values to be output, and during retrace intervals, it can update internal game state, compute new positions for objects, or prepare data for the next visible frame. This approach keeps the system well structured: the VGA controller guarantees that the output signal is always compliant with the VGA specification, while the system controller ensures that the displayed content matches the current state of the game or user interface.

This division of labour also supports scalability. Additional features such as animation, sprite layering, or tile-based backgrounds can be added entirely within the system controller or its supporting modules, without requiring any changes to the VGA timing logic. The result is a flexible and reusable video output architecture that remains stable regardless of the complexity of the graphical features built on top of it.

5.2 Input / Output Interfaces

Responsible: Jamie

The input/output subsystem is responsible for interfacing the external PMOD peripherals with the FPGA board. The goal of the subsystem is to present stable, decoded data to the system controller. The block acts as a hardware abstraction layer — isolating the rest of the system from low-level timing constraints, communication protocols, and also device-specific behaviour.

As previously mentioned, the subsystem supports three primary peripherals: a 4x4 matrix keypad for discrete user input, a two-axis analogue joystick for continuous control, and a NOR serial flash memory device for non-volatile storage.

As illustrated in Figure 1, the Input / Output Interfaces sit between the external PMOD peripherals and the central System Controller, providing a clean separation between device-level communication and system-level control.

5.2.1 PMOD KYPD Interface

The keypad interface scans a 4x4 matrix keypad using a row–column scanning technique. Columns are driven sequentially while rows are monitored to detect key presses. Detected inputs are debounced and decoded into a single digital key value, ensuring that only valid and stable key events are presented to the system controller. This prevents spurious transitions that may be caused by mechanical switch bounces and allows for the controller to respond to discrete user actions without needing additional filtering logic. [1]

5.2.2 PMOD JSTK2 Interface

The joystick interface communicates with the JSTK2 module using the SPI protocol. A dedicated FSM-controlled SPI master manages the complete transaction sequence, including specific timing parameters such as the chip-select timing, clock generation, and serial data capture — as specified in their respective datasheets.

The interface retrieves the joystick's X-axis and Y-axis positions along with the push-button state and converts this information into structured digital signals. By managing all protocol timing internally, the joystick module provides the system controller with reliable positional data without exposing the complexity of the SPI-level. This approach simplifies the top-level design, improves modularity, and allows the controller to focus on higher-level decision-making rather than low-level communication timing. [2]

5.2.3 PMOD SF3 Interface

The serial flash interface provides access to the SF3 NOR non-volatile memory module using SPI communication. This interface supports basic read and write operations required for storing configuration data and game-related values.

All command sequencing, address transmission, data transfer, and write-completion polling are managed internally by a dedicated control state machine. As a result, the system controller interacts with the flash memory using simple control signals and status flags, rather than implementing device-specific timing or polling logic. [3]

Collectively, the input/output interfaces ensure that all external peripherals are managed in a deterministic and modular manner. By encapsulating timing-critical behaviour and protocol handling within dedicated hardware blocks, the subsystem simplifies the system-level control, improves reliability, and supports the reuse of the platform across different game implementations.

5.3 System Controller

Responsible: Alan

The system controller defines the overall control flow of the FPGA game console platform. Its purpose is to decide what mode the system is operating in and how it should respond to user input, memory status, and internal control signals. My work focused on designing the structure and logic of this controller, rather than achieving a fully working hardware implementation.

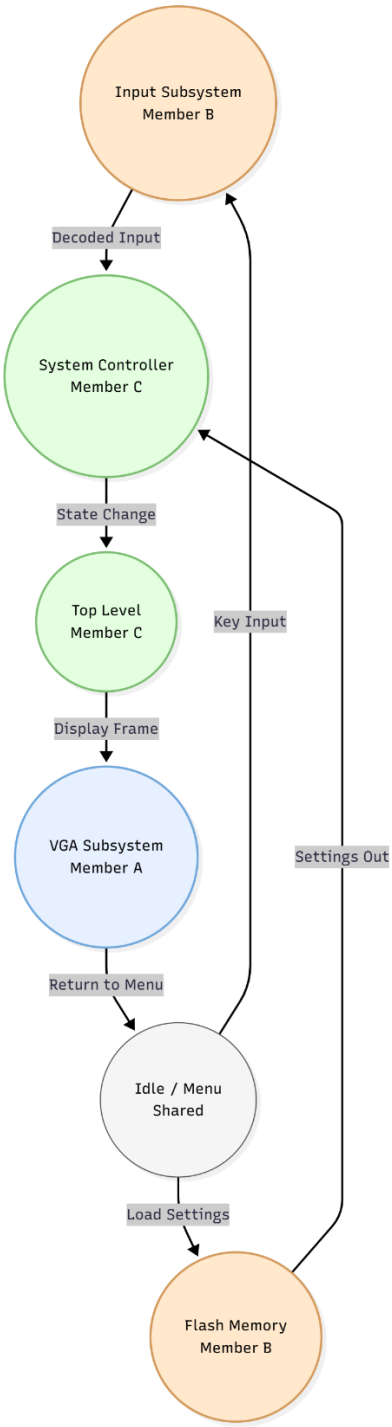

Figure 2 : System Group FSM

The controller is designed as a finite state machine. This approach was chosen because it allows the intended system behaviour to be broken into clear and understandable states. Each state represents a specific mode of operation, and transitions between states only occur under defined conditions. This makes it easier to reason about how the system should behave and prevents undefined behaviour.

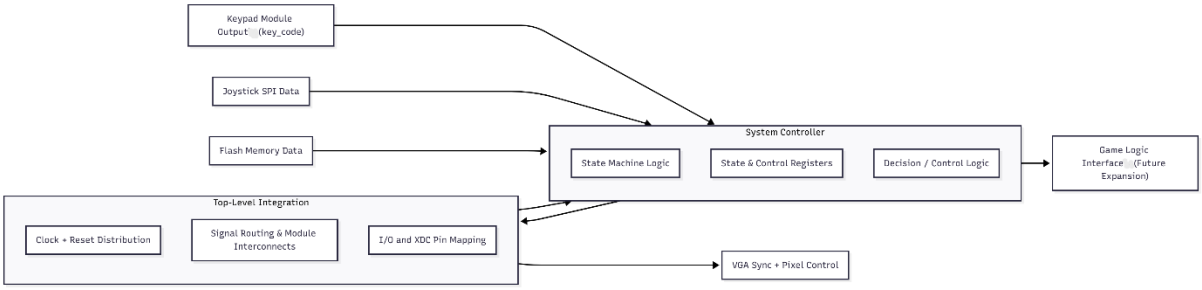

Figure 3 : System Controller Block Diagram and Top-Level Integration

As shown in the block diagram, the system controller receives decoded key input from the keypad module, joystick data from the joystick SPI interface, and status information from the flash memory interface. These signals are assumed to be processed by their respective subsystems before reaching the controller. This means the controller only deals with meaningful data such as key codes, joystick positions, and flash operation results, rather than raw signals or communication protocols.

Internally, the controller is split into state machine logic, state and control registers, and decision logic. The state register stores the current FSM state. The decision logic uses the current state and the input signals to decide which state the system should move to next and which control outputs should be active. This separation keeps the design structured and easier to understand.

One important behaviour that was considered in detail during the design is the pause function. Pausing is not implemented by stopping the FPGA clock or freezing the entire system. Instead, pausing is managed by moving the FSM into a dedicated pause state. When the controller enters this state, control signals that normally allow the game state to update are disabled. This means values such as player position, score, or timers would remain unchanged while the system is paused.

While in the pause state, the controller continues to monitor user input. This allows the system to respond to a resume command without resetting the game. When a resume input is detected, the FSM transitions back to the active game state and normal updates continue from where they were paused. This approach avoids losing game state and makes the pause behaviour predictable.

The same control principle applies to menu and settings states. When the controller is in the menu state, gameplay updates are disabled and only menu-related input is acted upon. When entering a settings state, gameplay remains inactive while settings are changed or written to flash memory. This prevents multiple parts of the system from trying to update data at the same time.

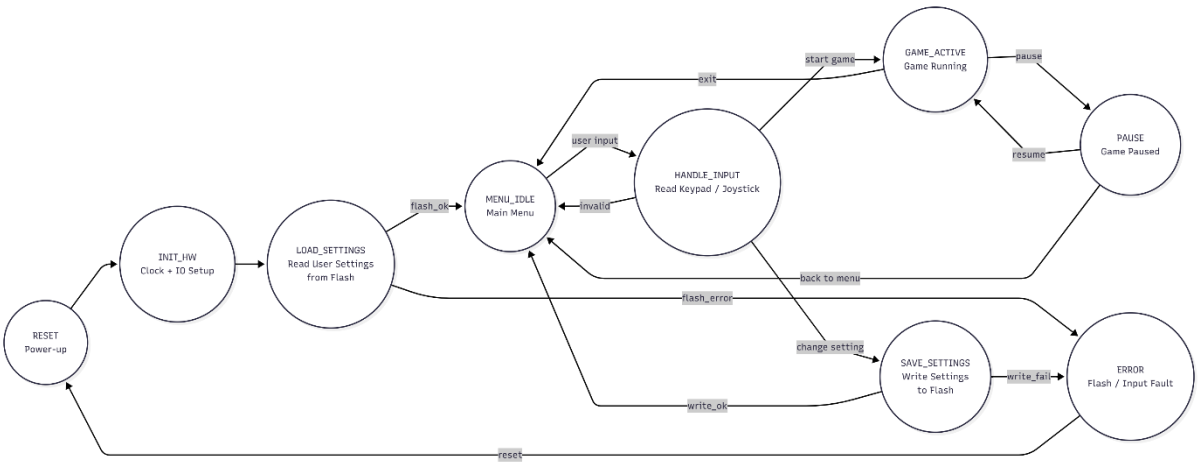

Figure 4 : Finite State Machine Diagram for the System Controller

The FSM begins in a reset state that represents system power-up. From there, the controller moves through initialisation and settings load states before entering the main menu. From the menu, user input can trigger transitions into gameplay, pause, settings save, or error handling states. The FSM also includes error states that are entered if a flash operation fails or invalid input is detected. These states ensure that faults are managed in a controlled way rather than allowing the system to behave unpredictably.

The system controller does not directly generate VGA timing or pixel data. Instead, it outputs high-level control information indicating what type of screen should be displayed, such as menu, gameplay, or pause. The VGA subsystem then uses this information to generate the required sync and RGB signals. This keeps the controller focused on control flow and allows the display logic to remain separate.

Although the full system was not completed and evaluated on hardware, the system controller design shows a clear and well-thought-out control strategy. The FSM structure demonstrates how features such as pause, menu navigation, and settings handling would work in a real system. If the project were continued, the next step would be to connect the FSM outputs to the VGA and memory subsystems and verify correct state transitions on hardware.

6. Top-Level Integration

At the top level, all functional blocks are instantiated and connected together through clearly defined interfaces. The system controller sits at the centre of this integration and is responsible for coordinating the flow of data and control signals between the input subsystem, VGA subsystem, and flash memory interface.

A single global clock and reset signal are shared across all blocks to ensure synchronous operation. By keeping clocking and reset behaviour consistent across the design, the risk of timing-related issues is reduced, and the system behaviour becomes easier to reason about.

7. Teamwork and Project Management

The project was divided into functional blocks, with each team member responsible for the design and documentation of a specific subsystem. Regular meetings were held to discuss overall system architecture, clarify block interfaces, and review design decisions as the project progressed.

Design updates and diagrams were shared among team members to ensure that individual blocks would integrate correctly at the top level. This collaborative approach helped maintain consistency across the design and allowed potential issues to be identified early.

8. Weekly Progress Reports Summary

Weekly progress reports were produced from week nine onwards, as required by the project specification. Each report summarised meetings held during that week, attendance, work completed by each team member, and any changes or updates to the system design. Responsibility for producing the weekly report rotated between team members to ensure equal contribution.

9. Challenges and Design Considerations

One of the main challenges in this project was defining clear and consistent interfaces between different subsystems. Care had to be taken to ensure that each block exposed only the signals needed by the rest of the system, while hiding unnecessary internal details.

Another challenge was managing timing requirements, particularly for VGA output and SPI-based peripherals. These challenges were addressed by separating timing-critical functionality into dedicated modules and maintaining a modular design approach. Although not all aspects of the system were fully implemented in hardware, these considerations informed the overall structure of the design.

10. Conclusion

In conclusion, this project resulted in the design of a modular hardware platform for video game style applications on the Digilent Basys3 FPGA. The system integrates VGA output, multiple user input devices, and non-volatile storage through a structured and reusable architecture. While the full system was not completed and evaluated on hardware, the design demonstrates a clear understanding of digital system organisation, modular design, and control flow using finite state machines. The platform provides a solid foundation that could be extended into a fully working game system with further development.

References

Group Project Github link: https://github.com/404JayNotFound/FPGA_GameSystem/tree/main

[1] Digilent, “PmodKYPD Reference Manual,” [Online]. Available: https://digilent.com/reference/_media/pmod:pmod:pmodKYPD_rm.pdf.

[2] Digilent, “Pmod JSTK2 Reference Manual,” [Online]. Available: https://digilent.com/reference/pmod/pmodjstk2/reference-manual.

[3] Digilent, “Pmod SF3 Reference Manual,” [Online]. Available: https://digilent.com/reference/pmod/pmodsf3/reference-manual.