Digital Lock Design Report

Phil Nevins
P.Nevins971@gmail.com
Electrical and Computer Engineering
FPGA Design
Portland State University

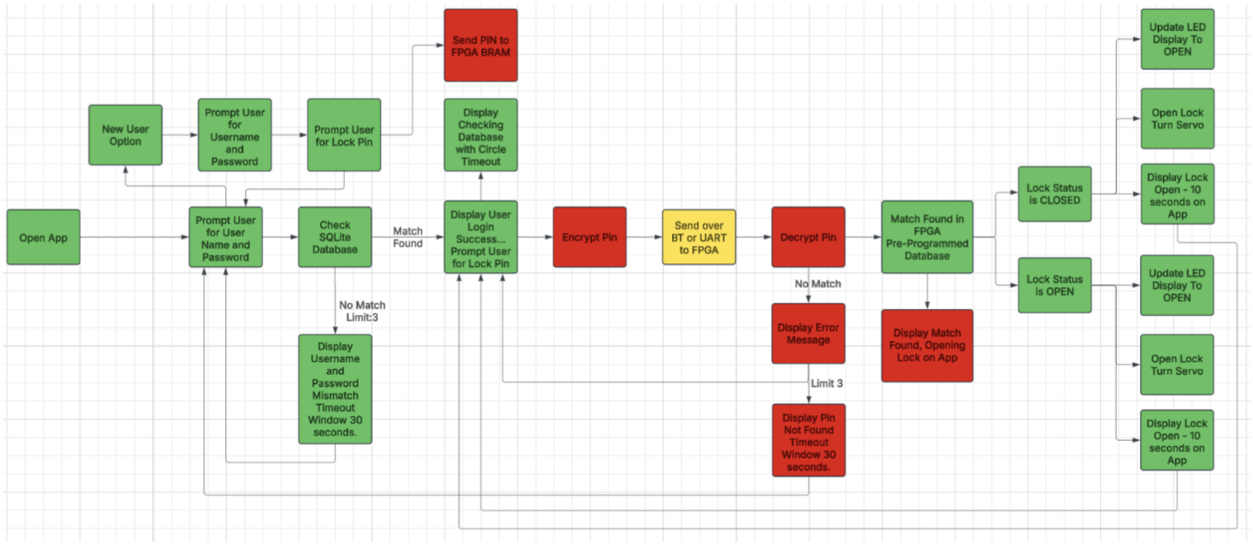
Nick A
nall2@pdx.edu
Electrical and Computer Engineering
Android App Design
Portland State University

1. Introduction

The FPGA hardware design process underwent multiple iterations as we worked towards establishing secure (AES-128 encryption) wireless communication between an FPGA and an Android device. Our original plan was to implement a Bluetooth-based system with an SQL database, enabling user authentication with dynamically added usernames, passwords, and PINs. This system would facilitate bidirectional communication between the FPGA and an Android app.

However, due to hardware limitations and persistent issues with Bluetooth reliability, we pivoted to a UART and MQTT-based approach. Unfortunately, MQTT failed to work as expected at WCC and doesn't implement secure transfer via TLS TCP, and additional delays in component availability hindered further progress. As a result, the final implementation demonstrates the FPGA-side functionality and the Android app working independently. Pre-assigned PINs and values are used to showcase system capabilities, validating core features despite the lack of direct communication between the two systems.

This section details the **major hardware design changes**, the **challenges encountered at each stage**, and the **final implementation** that will be demonstrated.

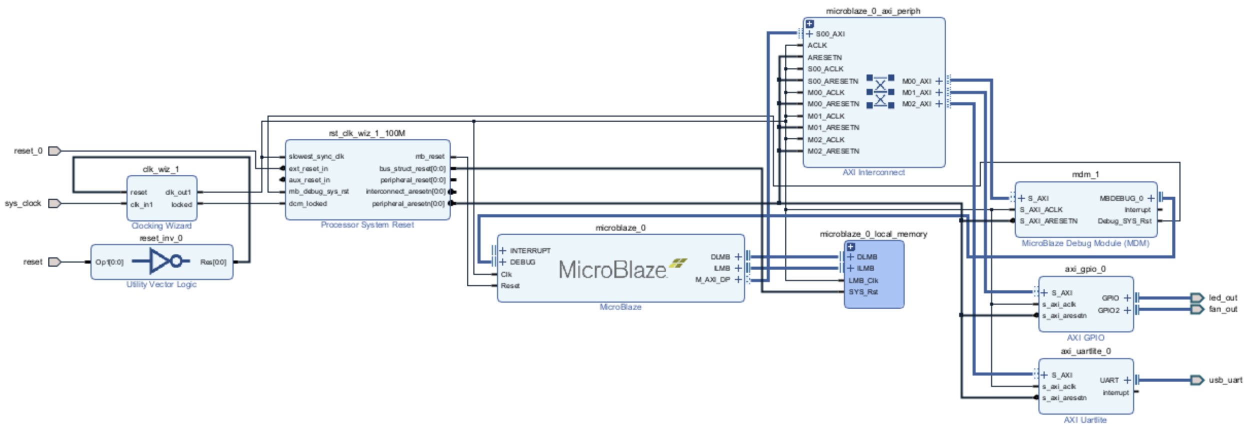

Open App
New User Option
Prompt User for Username and Password
Prompt User for Lock Pin
Send PIN to FPGA BRAM
Prompt User for User Name and Password
Check SQLite Database
Match Found
No Match Limit:3
Display Username and Password Mismatch Timeout Window 30 seconds.
Display Checking Database with Circle Timeout
Display User Login Success... Prompt User for Lock Pin
Encrypt Pin
Send over BT or UART to FPGA
Decrypt Pin
No Match
Display Error Message
Limit 3
Display Pin Not Found Timeout Window 30 seconds.
Match Found In FPGA Pre-Programmed Database
Display Match Found, Opening Lock on App
Lock Status is CLOSED
Lock Status is OPEN
Update LED Display To OPEN
Open Lock Turn Servo
Display Lock Open - 10 seconds on App
Update LED Display To OPEN
Open Lock Turn Servo
Display Lock Open - 10 seconds on App

Project Flow Chart

Block Design

---

2. FPGA Hardware



Plan A: AES-128 Module Implementation (Failed)

The initial approach was to implement an **AES-128 encryption module** on the FPGA to securely process authentication data. However, the module could not be validated successfully in **Vivado 2022.2**, leading to persistent issues with synthesis and hardware validation. Additionally, integrating the AES module with the FPGA's **I/O introduced unexpected pin assignment errors**, preventing successful bitstream generation. Due to these roadblocks, the AES-128 approach was **abandoned** in favor of pre-built alternatives.

Plan B: Pre-Made IP Core for Encryption (Failed)

To circumvent the validation issues in Plan A, we explored **pre-made IP cores** for encryption. The first IP core considered (**TinyAES** or a similar module) was built for a **different mode of operation**, where signals were left **floating** instead of being packaged into a **single RX/TX communication line**, making integration difficult.

Further attempts to use **other IP cores** were unsuccessful due to:

- Compatibility issues with **Vivado 2022.2** and **Vitis**.
- Incorrect signal mapping in pre-made IPs.
- Some IP cores being built for **older versions** of Vivado, making them **incompatible** with our hardware.

After multiple failed integration attempts, we **discarded** Plan B and moved to direct Bluetooth communication.

Plan C: HC-05 Bluetooth Module Direct Connection (Failed)

With encryption no longer viable, the next approach was to establish **direct Bluetooth communication** between the **FPGA and a Raspberry Pi 5 (RPi5) using an HC-05 Bluetooth module**. The **HC-05 was physically connected to the FPGA's PMOD ports (JB header), and initialized properly via Vitis**.

Challenges Encountered:

- The **HC-05 module was recognized and initialized**, but it **did not show up in Bluetooth device scans** on **any phone**.
- The **MAC address was detected**, but attempting to manually connect via **bluetoothctl** on the **RPi5 resulted in persistent connection failures**.
- The **HC-05 would enter pairing mode, allow pairing, but fail to establish an actual connection**.
- Every attempted **fix suggested by ChatGPT** was tested, including:
  - **Manually binding RFCOMM** using:
  - sudo rfcomm bind 0 XX:XX:XX:XX:XX:XX 1
  - **Restarting Bluetooth services**:
  - sudo systemctl restart bluetooth
  - **Resetting the HC-05 via AT commands** (in case it had an incorrect baud rate):
  - AT+UART=9600,0,0
  - **Disabling ModemManager (which could interfere with /dev/rfcomm0)**:
  - sudo systemctl stop ModemManager

RFCOMM Connection Issues

When attempting to manually create an RFCOMM connection, we faced persistent errors:

rfcomm connect 0 XX:XX:XX:XX:XX:XX 1 → Failed with error:
org.bluez.Error.NotAvailable br-connection-profile-unavailable
rfcomm output showed "clean" instead of "connected", indicating a partially established but inactive connection.
Manually sending messages via /dev/rfcomm0 resulted in:
echo "Hello FPGA!" > /dev/rfcomm0
Device or resource busy

Terminal & Debugging Issues

minicom -D /dev/rfcomm0 -b 9600 would freeze upon opening, not allowing input.
screen /dev/rfcomm0 9600 would immediately exit with:
Screen is terminating
picocom -b 9600 -r -l /dev/rfcomm0 failed with:
FATAL: read zero bytes from port
cat /dev/rfcomm0 resulted in no output or a hung terminal.

After **several days of debugging without success**, we shifted to **Plan D**.

Plan D: FPGA as I²C Slave, RPi5 as Bluetooth Relay (Failed)

The next attempt involved **replacing the HC-05 module entirely** and using the **Raspberry Pi 5 as a Bluetooth relay**. The idea was:

1. The **RPi5 receives Bluetooth messages from a phone**.
2. The **RPi5 forwards messages over I²C to the FPGA**.
3. The **FPGA processes the received data**.

Issues with Plan D:

- The FPGA was **configured as an I²C slave**, but the design **failed to compile** due to hardware constraints.
- **I²C communication validation failed**, preventing message exchange between the RPi5 and FPGA.
- Due to the **time remaining before the project demo**, debugging the I²C issue became infeasible.

At this point, we **abandoned** Plan D and moved to a **simplified demonstration approach**.

Plan F: Demonstration of Bluetooth Initialization & Message Handling

Since neither **HC-05 nor I²C communication** could be successfully implemented within the project timeline, we chose an **demonstration-based approach** (Plan F).

Implementation:

- The **HC-05 module is initialized and verified via the Vitis application**.
- The **RPi5 attempts to connect to the HC-05 module**—this connection will fail, as previously encountered.
- **Dummy code is implemented to handle received UART messages**, even though no actual messages will be received.

Code Implementation for Received Messages

To simulate the intended functionality, we implemented a message handling function that **waits for a PIN input**, then **controls a servo and LED** based on the received message.

```
// Wait for PIN
if (ReceivedBytes == PIN) {
   xil_printf("Pin Match Found! Opening Lock...\n\r");
   usleep(1000);

   // Send PWM Control Signal to Servo on JB3 (Channel 1)
   XGpio_DiscreteWrite(&GpioPWM, 1, ON);

   // Send PWM Control Signal to LED on JB4 (Channel 2)
   XGpio_DiscreteWrite(&GpioPWM, 2, ON);

   usleep(3000); // Keep on for 3 seconds
   xil_printf("Lock Open!\n\r");
} else {
   xil_printf("Pin Mismatch! Try again..\n\r");
}
```

Plan G: Debugging the AXI GPIO and Vitis Launch Issues (Success!)

While working on a **MicroBlaze-based project** in **Vivado and Vitis**, we encountered multiple issues, including:

- **AXI GPIO missing from `xparameters.h`**
- **Incorrect clock configurations**
- **Vitis application launch failures**
- **IO standard conflicts (`fan_out_tri_o` & `led_out_tri_o`)**

After extensive troubleshooting, the final **solution required rebuilding everything from scratch** in a new workspace.

Problems Encountered and Debugging Steps Taken

AXI GPIO Missing from xparameters.h

Issue

- The **AXI GPIO module (axi_gpio_0)** was correctly present in the **Vivado Block Design** and had a **valid base address**, but **did not appear in xparameters.h** in Vitis.

Debugging Steps

- **Checked Address Editor** – Verified axi_gpio_0 had a valid base address.
- **Regenerated Bitstream & Exported Hardware** – No change.
- **Rebuilt the BSP in Vitis** – Still missing.
- **Manually enabled AXI GPIO in BSP settings** – No effect.
- **Checked if AXI GPIO was optimized out** – It was **not removed** during synthesis.
- **Tried increasing GPIO width** – No effect.
- **Fully deleted and recreated the Vivado and Vitis projects – Finally fixed it!**

Differential Clock Issue

Issue

- A **differential clock (diff_clock_rtl_clk_n & diff_clock_rtl_clk_p)** unexpectedly appeared in the design, causing **NSTD-1 and UCIO-1 DRC errors**, preventing bitstream generation.

Fix

- **Checked Clocking Wizard Configuration** – Found it was set to **differential input**.
- **Changed to a single-ended clock** – Fixed the issue.
- **Removed unnecessary differential clock constraints from .xdc**.
- **Re-validated and re-generated the bitstream** – No further errors.

Vitis "Unable to Launch" Issue

Issue

- Vitis displayed **"Unable to Launch: The selection cannot be launched, and there are no recent launches."**

Fix

- **Checked if ELF file was generated** – The project was built, but **no valid .elf was found**.

- **Rebuilt the BSP and Application Project** – No change.
- **Manually created a new Debug Configuration** – Did not fix it.
- **Checked FPGA programming status** – The FPGA was **not programmed**.
- **Manually programmed the FPGA from Vivado and retried launching** – Still failed.
- **Deleted .launch files & cleaned Vitis workspace** – No fix.
- **Fully deleted and recreated both Vivado and Vitis projects – Finally worked!**

IO Standard Conflict (fan_out_tri_o & led_out_tri_o)

Issue

- **Vivado DRC Error (BIVC-1)**:
- Conflicting VCCO voltages in Bank 15. For example, the following two ports have conflicting VCCOs:
- fan_out_tri_o[0] (LVCMOS18, requiring VCCO=1.800)
- reset (LVCMOS33, requiring VCCO=3.300)
- **Despite setting fan_out_tri_o to LVCMOS33**, Vivado still treated it as LVCMOS18 due to an **interface constraint**.

Fix

```
set_property INTERFACE "" [get_ports fan_out_tri_o]
reset_property IOSTANDARD [get_ports fan_out_tri_o]
set_property IOSTANDARD LVCMOS33 [get_ports fan_out_tri_o]
set_property INTERFACE "" [get_ports led_out_tri_o]
reset_property IOSTANDARD [get_ports led_out_tri_o]
set_property IOSTANDARD LVCMOS33 [get_ports led_out_tri_o]
```

Verified fix

```
report_property [get_ports fan_out_tri_o]
report_property [get_ports led_out_tri_o]
```

Result: No more IO standard conflicts, and synthesis completed successfully.

Final Resolution: Rebuilding Everything from Scratch

After exhausting all debugging options, the **only reliable solution** was to **delete both Vivado & Vitis projects** and start fresh.

Steps Taken to Fix Everything

Created a New Vivado Project

- **Recreated the Block Design** from scratch.

- **Manually re-added AXI GPIO, AXI Interconnect, and MicroBlaze**.
- **Configured the Clocking Wizard** properly to use a **single-ended input**.
- **Regenerated the bitstream**.
- **Exported hardware (.xsa)** including the bitstream.

Created a New Vitis Workspace & Platform

- **Imported the new .xsa file**.
- **Regenerated the BSP** (**ensuring AXI GPIO was enabled**).
- **Built the application project**.

Successfully Launched the Program

- **No missing xparameters.h definitions**.
- **No clock-related issues**.
- **No launch errors**.

Expanding GPIO to Control a Fan

After getting **AXI GPIO working**, we expanded it to **control a 5V fan**.

Modifications Made

1. **Increased AXI GPIO output width to 2:**
    - Updated the **Vivado Block Design** to support **two output signals**.
    - Modified the .xdc file to **assign the second GPIO pin** to a Pmod connector.
2. **Added a 2N2222A Transistor for Fan Control:**
    - Since the Nexys A7 **cannot supply 5V directly**, we used a **2N2222A transistor** to **switch the fan on/off**.
    - A **1kΩ resistor** was added between the GPIO pin and the transistor base.
3. **Used Raspberry Pi 5 to Power the Fan:**
    - Nexys A7 **Pmod pins only supply 3.3V**, which is **not enough**.
    - The fan's + **terminal was connected to Raspberry Pi 5 (5V)**.
    - The **fan's - terminal was connected to the transistor collector**.
    - The **transistor emitter was connected to ground**.
4. **Updated the Code to Control Both LED and Fan:**

```
#define LED_ON  1
#define LED_OFF 0
#define FAN_ON  2
#define FAN_OFF 0

    if (strcmp(input_buffer, PIN) == 0) {
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, ON);
        XGpio_DiscreteWrite(&Gpio, FAN_CHANNEL, ON);
        sleep(3);
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, OFF);
        XGpio_DiscreteWrite(&Gpio, FAN_CHANNEL, OFF);
        return 0;
    } else {
        // Blink LED 3 times
        for (i = 0; i < 3; i++) {
            XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, ON);
            sleep(1);
            XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, OFF);
            sleep(1);
            return 0;
        }
    }
```

Lessons Learned

1. **Vivado & Vitis Sometimes Fail to Update Properly:**
   - Even after regenerating the bitstream, **Vitis might still use outdated BSPs**.
   - If **something is missing, recreating the project may be the only solution**.
2. **Clocking Wizard Configuration is Critical:**
   - If using **single-ended clocking**, ensure the **Clocking Wizard** is set **correctly**.
3. **Power Considerations Matter:**
   - **The Nexys A7 cannot power a 5V fan**, so an **external supply** was required.
   - **Raspberry Pi 5 provided 5V**, while **FPGA controlled the transistor switch**.
4. **Transistors Work, But MOSFETs Are Better:**
   - While the **2N2222A worked**, a **logic-level MOSFET (IRLZ34N)** would be **more efficient**.
5. **When in Doubt, Start Fresh:**
   - **After multiple failed attempts, a full rebuild in a new workspace was the ultimate fix.** ✅

---

3. Android Application

Overview:

The android app for the joint project between ECE 544 and ECE 558 is the user portal to allow a user to remotely unlock a lock. To access the ability to send the PIN that will unlock the lock the user must first login via the login screen. If the users credentials match those found in the database the user will be able to send their pin over MQTT to unlock a lock. The user is also able to add other users to the database, thus allowing multiple people to unlock said lock, making it perfect for an administrator handling digital security for an apartment complex, or possibly a homeowner "duplicating" their digital keys to allow friends and family access to their home.

Design Details/Theory of Operation:

The App utilizes 5 separate screens: The Login Screen and associated Login Splash screen, the add user screen and associated add user splash screen and finally the SendPin/LockStatus Screen. Quite a bit of effort was placed into clear and simple UI design utilizing common features one would expect from a modern application running on a smartphone:

Login Screen: This is the app's home screen. It allows the user to login to send their pin, or add a user to the database. Checking is done to make sure that a blank username and password cannot be sent, and the password is obfuscated unless the user clicks the icon to make it visible.

Login Splash Screen: Given that our database is incredibly small, this screen uses a spinning progress bar to notify the user that the database is benign checked. This is purely simulated via a kotlin coroutine. If the login credentials are not found in the database the user is returned to the login screen, otherwise a successful login notifies the user as such and lets them know that they are connecting to the MQTT broker so the transmission can occur. Once the MQTT broker is connected to the user is transferred to the screen to send their pin.

Add User Screen: This screen allows for adding other users to the database. It features the following robust error checking mechanisms to keep the database full of only good, safe user data: It forces a user to verify their username/password/PIN via corresponding matching entries. It checks for blank usernames/passwords/pins and associated verification fields. It does not allow for usernames/passwords to be null (not necessary in kotlin but a good practice nonetheless). It checks that the Pin/Pin verification fields are exactly 4 digits i.e. only 4 numerical characters.

Add User Splash Screen: Utilizes the same user feedback mechanisms as the login splash screen. If a username is already taken this screen rejects the addition to the database and sends the user back to the add user screen. If the username is not taken then the user is added to the database and then the app user is redirected to the login screen so they may login.

Send PIN/Lock Status Screen: This allows the user to send their PIN over MQTT to unlock a lock. The screen publishes status updates to notify the user about the lock status. It notifies a user if no match is found, as well as if a match is found and the status of the lock itself: Open or closed.

Results:

Despite some troubles and shifts in specification the App does what it is supposed to do. All the main functionality is there and listed above which was possible due to the following changes:

Instead of using Bluetooth we switched to MQTT. While this didn't work out during our demo at WCC (I believe this is something to do with their network) we did include links to video evidence of the project working and simulating a lock opening via an LED lighting up.

The database in the app is not persistent. This means it only lives in RAM. I joked during the demo that this makes your data safe (nothing to steal if we don't have it), we would actually want a working database to store a user's credentials long term. If we did get this working a future build should incorporate the ability to change a users username/password/PIN as right now there is no way to access the database except for entering users.

Some minor persistent issues are that the text only displays while the app is in light mode and the back arrow the app uses is deprecated in android studio, so it could be a problem in the future. The interface is a bit clunky with the keyboard so that could also be ironed out too, but these were considered relatively low priority issues before the demonstration of the App.

The security timeout feature had to be scrapped entirely. An issue likely to do with a race condition on two separate LaunchedEffects().

The WCC demo wasn't perfect on the App side. The usual and unfortunate MQTT issue arose again but the other issue was entirely created by me in a last minute and sleep deprived panic. The length between the login splash screen and the sen pin/lock status screen was because of the MQTT connection taking a long time. I was either working off of expectations from my previous versions, or the MQTT connection was much faster at my home. One app crash at WCC led to a panic rebuild of the project shortly before our demonstration time. While the video in class and the live demo met in terms of showing the full functionality, the links to the demo included in the project submission are much more detailed and clear to show that the app works.

---

4. Conclusion

The **FPGA hardware implementation** went through multiple iterations as we navigated **hardware limitations, software compatibility challenges, and debugging difficulties**. Our initial goal of establishing **secure AES-128 encrypted wireless communication** between the **FPGA and an Android device** had to be adjusted due to **persistent Bluetooth reliability issues and hardware constraints**. Despite attempting an **alternative UART and MQTT-based approach**, **MQTT did not function as expected**, and **delays in component availability** further impacted progress.

Ultimately, the **final implementation** showcases the **FPGA-side functionality** and the **Android app working independently**. While **direct communication** between the two systems could not be established, **pre-assigned PINs and values** effectively demonstrate **core system capabilities**. This **iterative design process** highlights the **complexity of Vivado and Vitis workflows**, where **small misconfigurations** can lead to **significant roadblocks**. Through **extensive troubleshooting**, we successfully **validated key aspects of the system**, reinforcing the **importance of adaptability in FPGA-based development**.

Key Takeaway: If you're stuck with persistent Vivado/Vitis issues, don't be afraid to nuke the project and start fresh!

Given additional time, further improvements would include:

- **Revisiting Bluetooth communication between the RPi5 and FPGA using HC-05**
- **Replacing the HC-05 with a Bluetooth module known to work reliably with modern Linux systems**.
- **Debugging MQTT communication issues.**
- **Creating a persistent Database**
- **Allowing users to edit their credentials: Username/Password/PIN**
- **Weak/Medium/Strong Password checking**
- **Adding the app security timeout**
- **Fixing the low priority app issues listed above**

The final demonstration serves as **proof of concept**, showcasing the work completed and highlighting areas for further refinement.

Here is our github repo: https://github.com/Dawgburt/ECE5xx-FinalProject/tree/main

---

5. Acknowledgements

We acknowledge the contributions of various resources and tools that have supported us throughout this project. The successful completion of our work would not have been possible without the assistance of these references and technologies.

We would like to express our gratitude to our instructor and teaching assistants for their invaluable guidance, as well as to our peers for fostering an engaging and collaborative learning environment. Their insights helped us refine our understanding of computer architecture and pipeline simulation.

Additionally, we recognize the role of artificial intelligence tools in supporting our development process. We utilized **ChatGPT-4o** and **ChatGPT-4.5** (OpenAI, 2024, 2025) for assistance in debugging, conceptual clarifications, and documentation refinement. These AI tools provided valuable insights, coding suggestions, and verification of our implementation strategies.

We also acknowledge the technical documentation that played a crucial role in our hardware and software implementation:

- **Nexys A7-100T Reference Manual** (Digilent, 2024) for detailed information on FPGA architecture, I/O peripherals, and interfacing methodologies.
- **Raspberry Pi 5 Documentation** (Raspberry Pi Foundation, 2024) for insights into the board's capabilities, interfacing with peripherals, and performance optimization.
- **Raspberry Pi Pico W Documentation** (Raspberry Pi Foundation, 2024) for guidance on microcontroller programming, wireless communication, and real-time control applications.
- **Vivado Design Suite Documentation** (Xilinx, 2024) for FPGA design flow, synthesis, and hardware debugging techniques.
- **Vitis Unified Software Platform Documentation** (Xilinx, 2024) for software-hardware co-design, FPGA acceleration, and embedded system development.

The knowledge derived from these sources enabled us to integrate hardware and software effectively, ensuring a robust and functional implementation.

6. References

- Digilent. (2024). *Nexys A7-100T Reference Manual*. Retrieved from https://digilent.com
- OpenAI. (2024). *ChatGPT-4o*. Retrieved from https://openai.com
- OpenAI. (2025). *ChatGPT-4.5*. Retrieved from https://openai.com
- Raspberry Pi Foundation. (2024). *Raspberry Pi 5 Documentation*. Retrieved from https://www.raspberrypi.com/documentation
- Raspberry Pi Foundation. (2024). *Raspberry Pi Pico W Documentation*. Retrieved from https://www.raspberrypi.com/documentation/microcontrollers
- Xilinx. (2024). *Vivado Design Suite Documentation*. Retrieved from https://www.xilinx.com/products/design-tools/vivado.html
- Xilinx. (2024). *Vitis Unified Software Platform Documentation*. Retrieved from https://www.xilinx.com/products/design-tools/vitis.html

- Stevdza-San on YouTube

- Codex Creator on YouTube

- Coding with Joyce on YouTube

- Droid.com - Mastering Navigation in Jetpack Compose