# Digital Lock Design Report

*Nick A*

*Email*

*Electrical and Computer Engineering*

*FPGA Design  
Portland State University*

*Phil Nevins*

[*P.Nevins971@gmail.com*](mailto:P.Nevins971@gmail.com) *Electrical and Computer Engineering*

*FPGA Design  
Portland State University*

# 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, 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.

A diagram of a diagram

AI-generated content may be incorrect.  
*Project Flow Chart*

*Block Design*

# FPGA Hardware

A computer diagram of a computer

AI-generated content may be incorrect.

#### 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.

DebuggingSteps

* **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.** ✅

# Android Application

# Conclusion

The FPGA hardware implementation went through **six different design plans** due to **unforeseen hardware constraints, software compatibility issues, and debugging difficulties**. Despite setbacks, the final demo will demonstrate the Bluetooth module’s **initialization and connection attempts**, along with a **simulated message handling system**. This debugging process highlights the complexity of **Vivado & Vitis workflows** and how small misconfigurations can cause major issues. **Rebuilding everything from scratch in a new workspace** ultimately solved the problem, and the project was successfully expanded to control both an **LED and a fan**.

**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.**

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