Bridging Cross-Domain Communication in Asynchronous and Synchrnous Digital Systems

Captone Design Project - Final Presentation

Supervised by: Professor Julius Georgiou

Petros Petrou
Undergraduate Student
Department of Electrical and Computer Engineering

University of Cyprus
Department of Electrical and Computer Engineering

INTRODUCTION

The Thesis:

- Investigates the communication dynamics between synchronous and asynchronous digital circuits to enhance interoperability and efficiency.

Highlights the asynchronous system technology, including:

- Flexibility in timing and data handling
- Advantages in performance and efficiency
- Advanced communication protocols
- Comparative analysis with synchronous systems to showcase benefits and differences

Showcases a communication system designed to:

- Effectively bridge synchronous and asynchronous elements
- Demonstrate the practical integration of both systems
- Provide a user-friendly environment
- Significantly improve overall system efficiency and reliability

Part I:
Foundational Concepts and Theoretical Background

SEMICONDUCTOR TECHNOLOGY

Merits

- Miniaturization
- Communication Revolution
- Industrial Automation
- Commercial Contribution

Impact on Circuit Design

- Design for Manufacturability
- Use of Advanced Modeling Tools
- Focus on System-Level Integration
- Enhanced Reliability and Testing

Challenges

- Fabrication Limitations
- Thermal Management
- Power Consumption
- Signal Integrity

New Trends

- 3D Integration
- Multi-layer PCBs
- Silicon Formulations
- Use of new materials

SYNCHRONOUS SYSTEMS

All the digital circuits in a synchronous system work under a unified clock that orchistrates the timing signals.

Advantages

- Predictability/Simplified Debugging
- Ease of Design

Synchronous Clock Domains

Challenges

- Clock Domain Crossing
- Metastability
- Jitter and Skew

Management

- Static Timing Analysis
- Asynchronous FIFOs
- Dynamic Frequency Scale
- Domain Separation
- Conservative Margins
- Regular Testing

ASYNCHRONOUS SYSTEMS

All the digital circuits in an asynchronous system work with the absence of a clock. Instead, their signals rely on handshake mechanisms and protocols

High Performance

By eliminating clock skew, they can operate more efficiently in average scenarios than the worst-case scenarios often considered in synchronous systems

Low Power Consumption

They use power more efficiently because they do not rely on the constant operation of a global clock.

Reduced Electromagnetic Interference

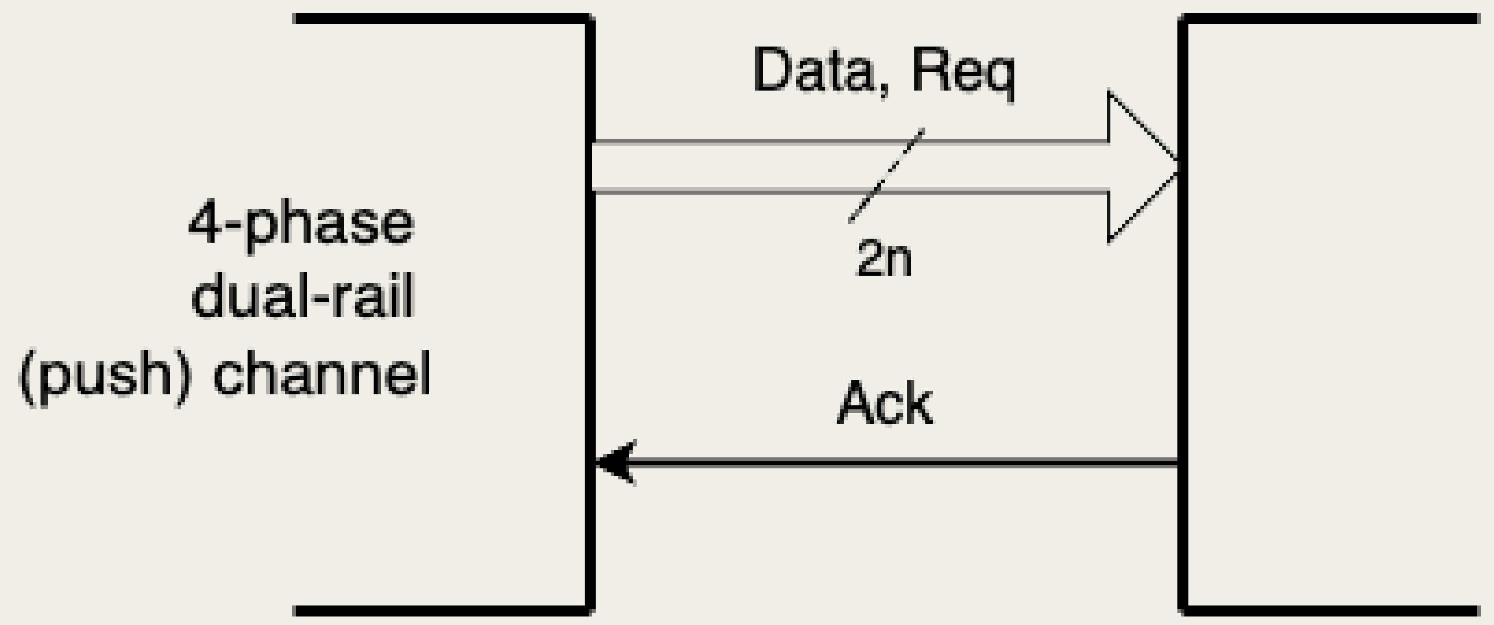
They have a more distributed noise spectrum rather than synchronous systems where nearly all energy is concentrated in narrow spectral bands around the clock frequency

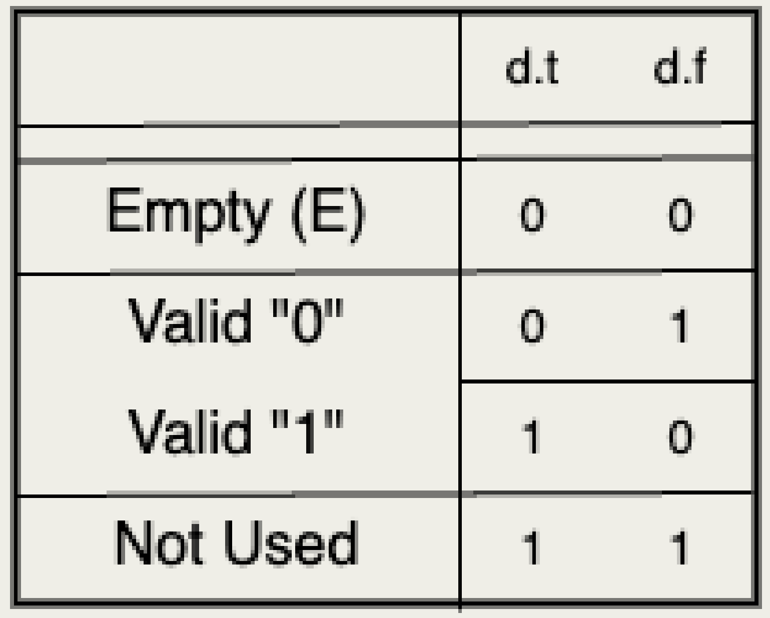
Modularity

Channel-based transmit and receive discipline in an asynchronous design promotes modularity and enables easy plug-and-play connectivity between blocks.

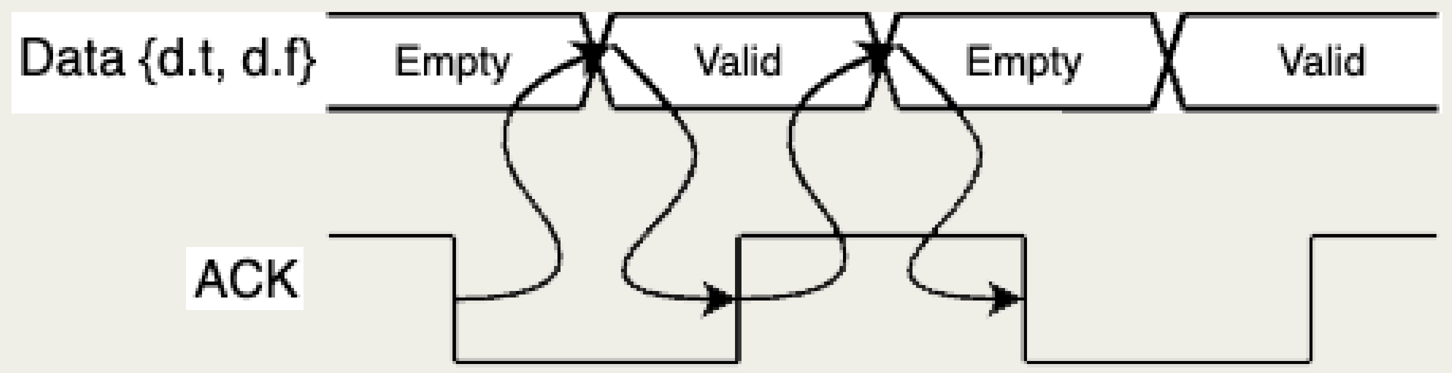
ASYNCHRONOUS SYSTEMS

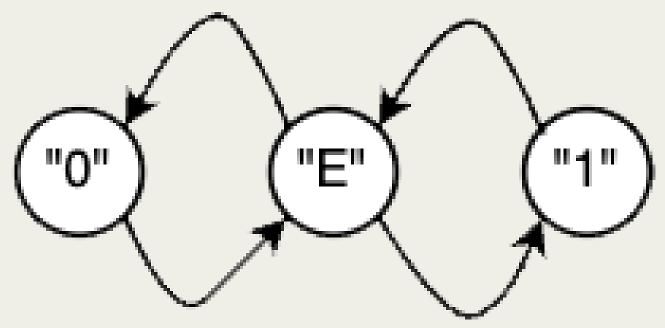
4-Phase Dual-Rail Protocol





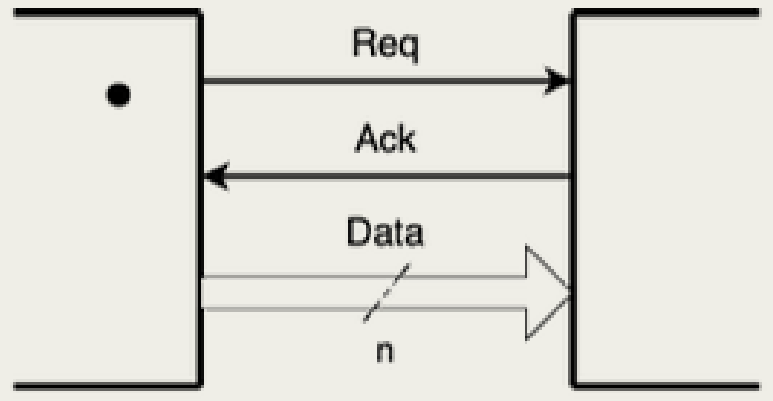
	d.t	d.f
Empty (E)	0	0
Valid "0"	0	1
Valid "1"	1	0
Not Used	1	1



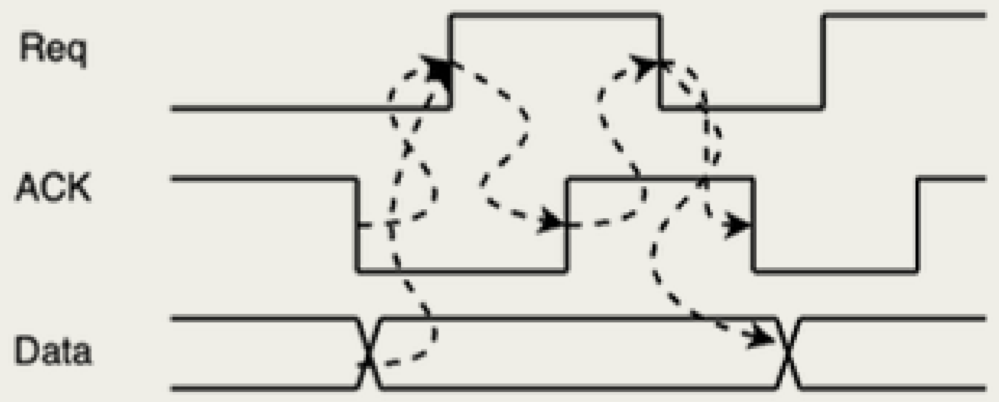


ASYNCHRONOUS SYSTEMS

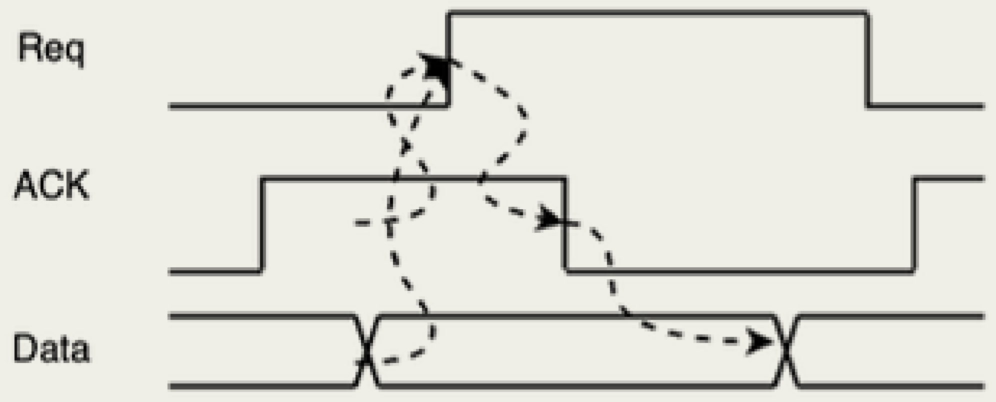
Bundled-Data Protocols



(a) Bundled data (push) channel



(b) 4-phase protocol



(c) 2-phase protocol

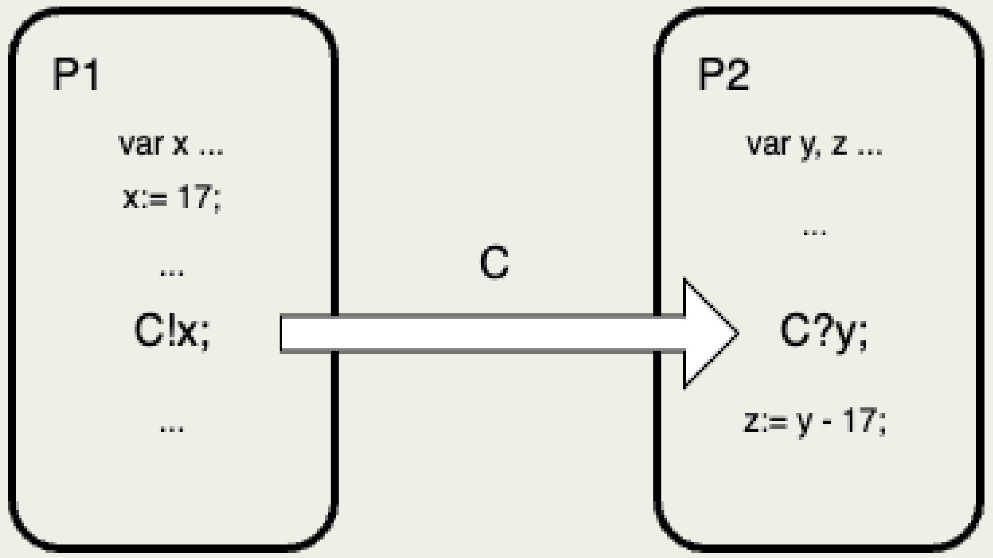
HIGH-LEVEL LANGUAGES AND TOOLS IN ASYNCHRONOUS CIRCUITS

Communicating Sequencial Processes

A hardware description language for modeling asynchronous circuits

Key Features include:

- Concurrent Processes
- Composition of Processes
- Synchronous Message Passing



Martin's Translation Process

A structured methodology for transforming high-level asynchronous circuits design to optimised hardware implementations

1. Process Decomposition
   - Decompose complex processes into simpler
2. Handshake Expansion
   - Translate actions into signal transitions

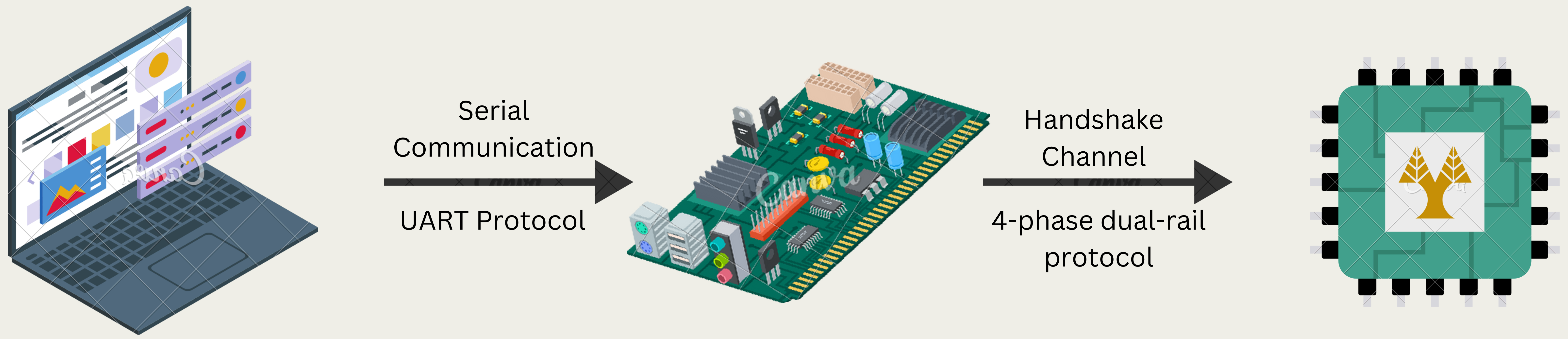
$$\left[\underline{C_{\text{req}}}\right]; y := \underline{\text{data}}; \underline{C_{\text{ack}}}\uparrow; \left[\neg \underline{C_{\text{req}}}\right]; \underline{C_{\text{ack}}}\downarrow$$

3. Production Rule Expansion
   - Replace handshake expansions with conditional action to define the behaviour
4. Operator Reduction
   - Group rules into clusters

Proposed System for Asynchronous - Synchronous Communication

PROPOSED COMMUNICATION SYSTEM

Graphical User Interface | **Embedded System** | **Asynchronous Chip**



Specifications - Requirements
- User is able to send bit streams
- User is able to monitor the communication process

Specifications - Requirements
- Store the received bit streams
- Send the received bit streams to the asynchronous chip

Specifications - Requirements
- Able to receive the incoming streams
- Send ACK back for verification

Part II:
Comparative Technical Exploration and Hardware Selection

HARDWARE SELECTION

Why using a Bridge Hardware?

- It gives the ability for easier configuration.
- With the use of an embedded system as an intermediary, there is easier control of the voltage levels that are being sent and the shifting of them.

Embedded System Requirements

- Serial Connection Establishment
- Data Reception and Storing to the Non-Volatile Memory
- General Purposed Input/Output for the Communication Establishment with the Asynchronous Chip

The search focused on:

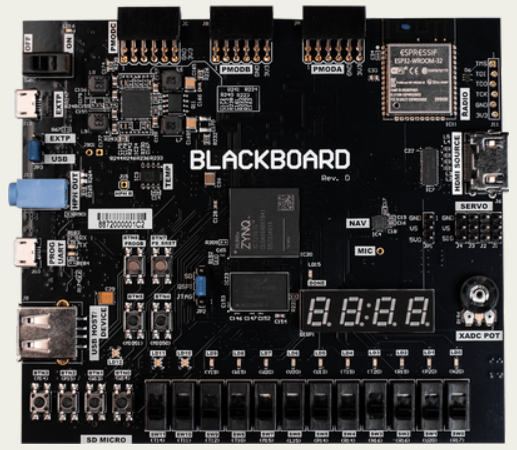
- Field Programmable Gate Arrays
- Microcontrollers

Comparison Metrics

- Features and Characteristics
- Advantages of each type
- Implementation Strategies

HARDWARE SELECTION

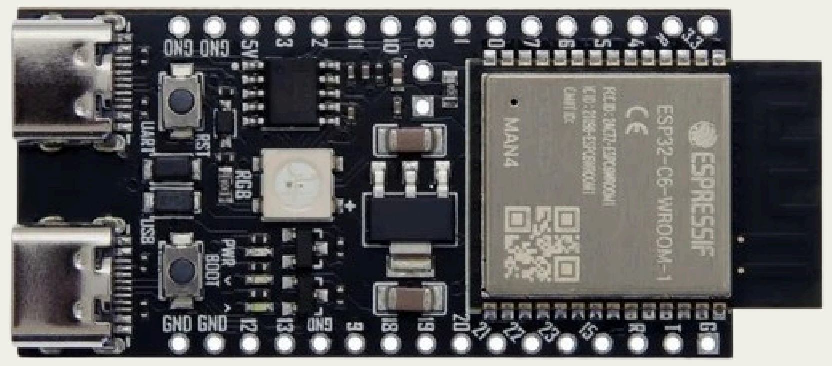
FPGA Selected Device



- Board Name: Blackboard
- Manufacturer: Real Digital
- Chip Model: Xilinx XC7007S
- ZYNQ
- Features: UART, 3 Pmod ports, Bluetooth
- Price: $149.00

Challenges - Limitations: Lack of Bluetooth, Need of GPIO Pmod, Need of Level Shifter Pmod

MCU Selected Device



- Board Name: ESP32-C6-DevKitM-1 - MCU
- Manufacturer: Espressif
- Chip Model: ESP32-C6
- Features: UART, Bluetooth, 32 pins GPIO, Output 3.3V
- Framework: ESP-IDF (Built on C)
- Price: €7 - €15 - Mouser

ENGINEERING OF THE SOFTWARE

Purpose of the System:

- Give the ability to the user to send data packets through a user-friendly environment to the bridge hardware and therefore to an Asynchronous chip.

Software Requirements:

Functional Requirements

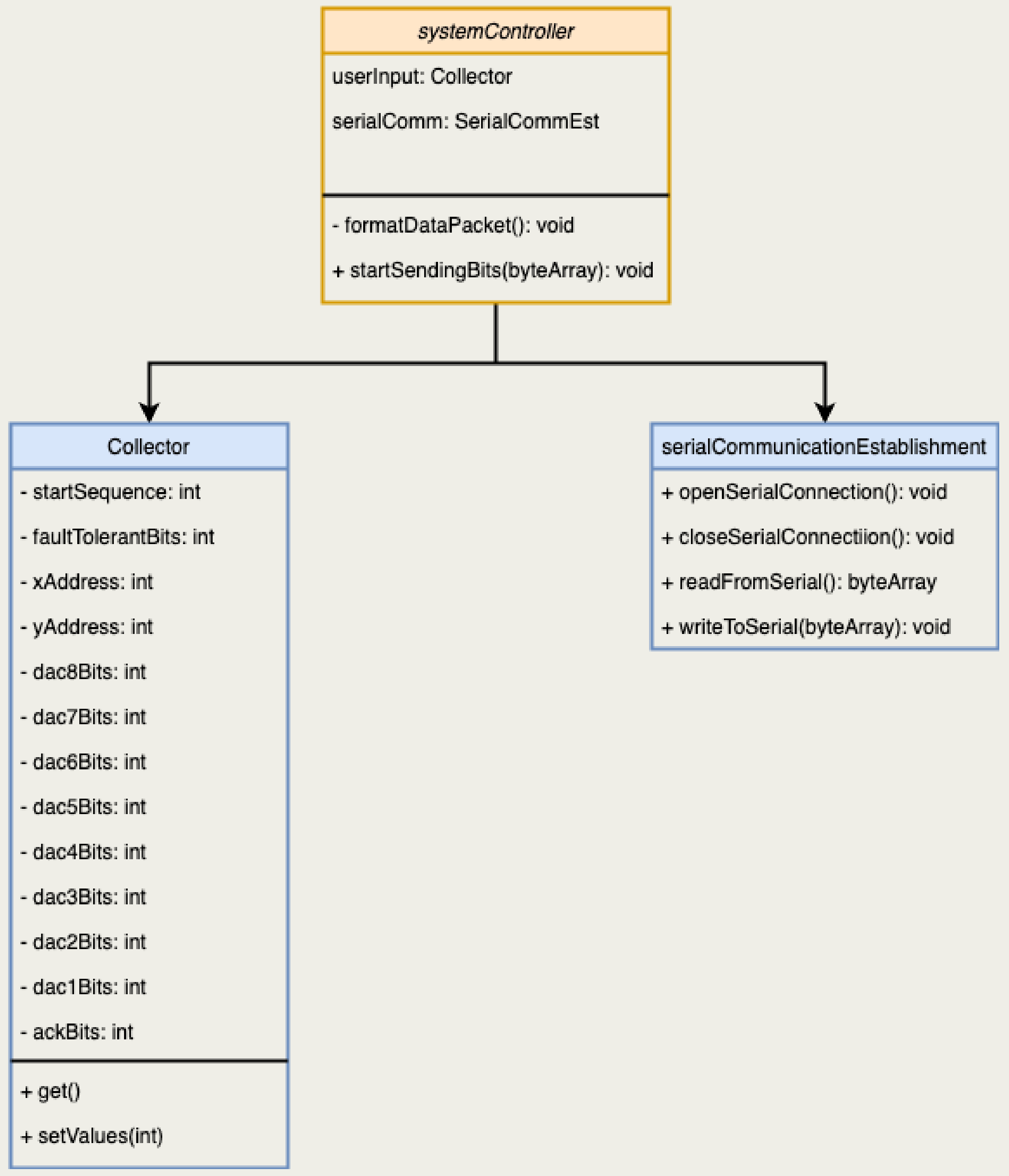
- Add Input Data
- Start Sending Bits Function
- Data Packet Formation
- Serial Communication Establishment

Non-Functional Requirements

- User Friendly Operational Environment
- Terminal Integration
- Log Activity - History

ENGINEERING OF THE SOFTWARE

Software's Class Diagram



ENGINEERING OF THE SOFTWARE

Implementation Framework Selection

.NET MAUI

- Programming Language: C#
- Creator: Microsoft
- Community: Strong Documentation
- Performance: Native code compilation, resource management, optimised for async programming
- Usage: Suitable for Scalability

KIVY

- Programming Language: Python
- Creator: Independent Organisation
- Community: Weaker, early stage
- Performance: Python Interpretation Overhead, Not optimised resource usage
- Usage: Prototyping GUIs

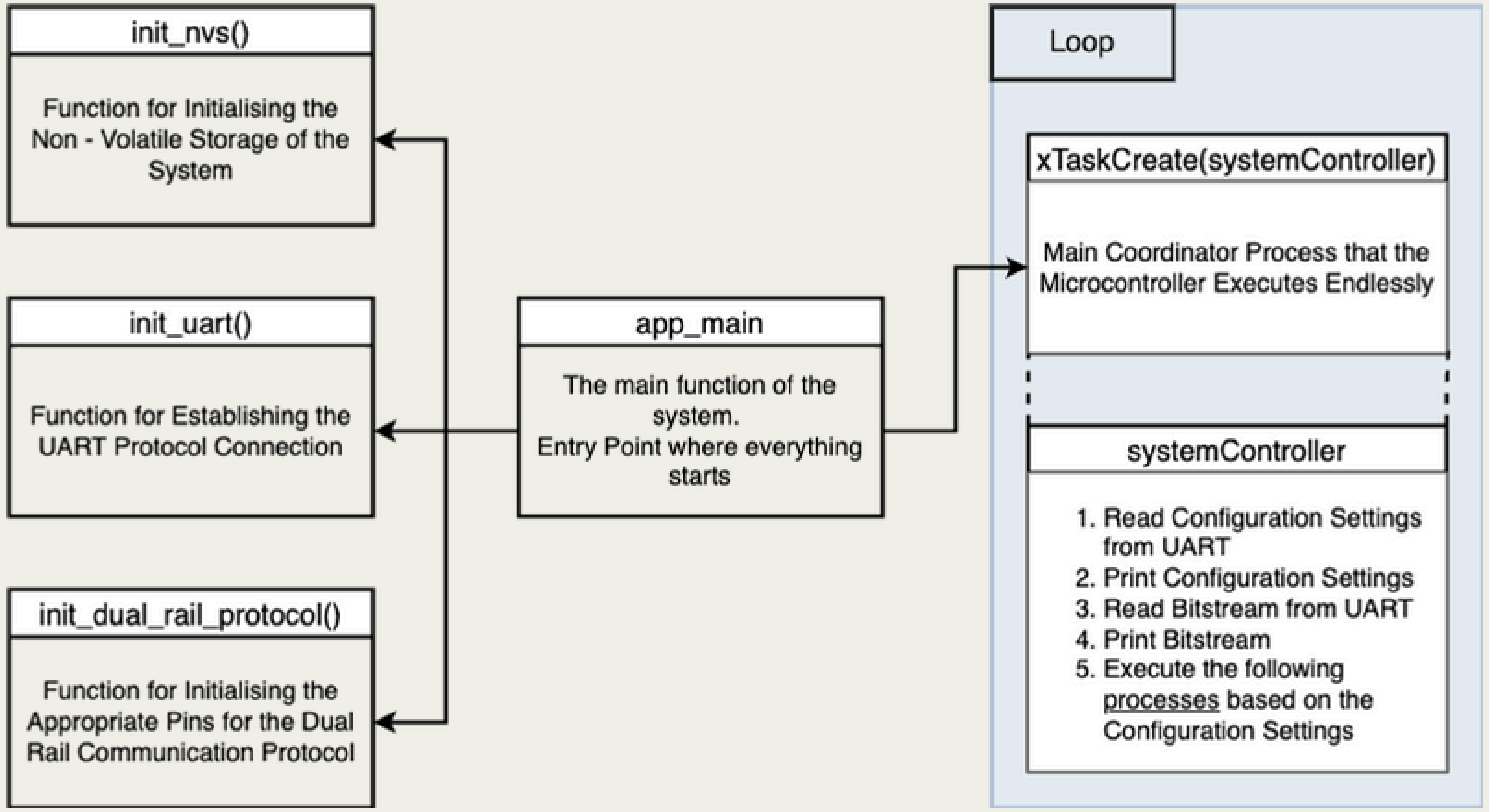
Qt

- Programming Language: C++
- Creator: Qt Group
- Community: Strong Documentation
- Performance: High with emphasis on runtime performance
- Usage: Suitable for scalability

Part III:
Practical Implementation, Evaluation and Synthesis

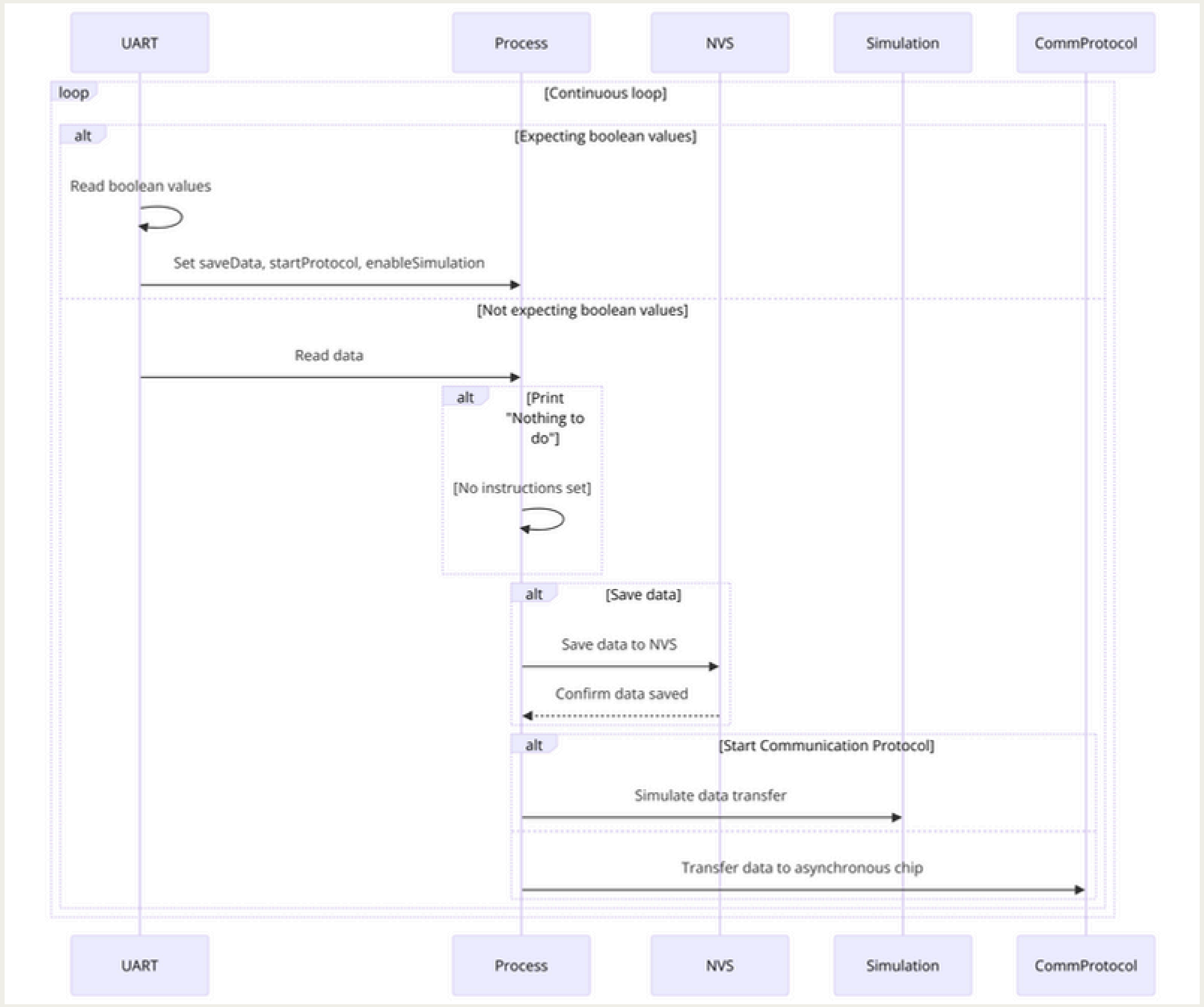
BRIDGE HARDWARE IMPLEMENTATION

System Architecture


init_nvs()
Function for Initialising the Non - Volatile Storage of the System
init_uart()
Function for Establishing the UART Protocol Connection
init_dual_rail_protocol()
Function for Initialising the Appropriate Pins for the Dual Rail Communication Protocol
app_main
The main function of the system. Entry Point where everything starts
Loop
xTaskCreate(systemController)
Main Coordinator Process that the Microcontroller Executes Endlessly
systemController
1. Read Configuration Settings from UART
2. Print Configuration Settings
3. Read Bitstream from UART
4. Print Bitstream
5. Execute the following processes based on the Configuration Settings

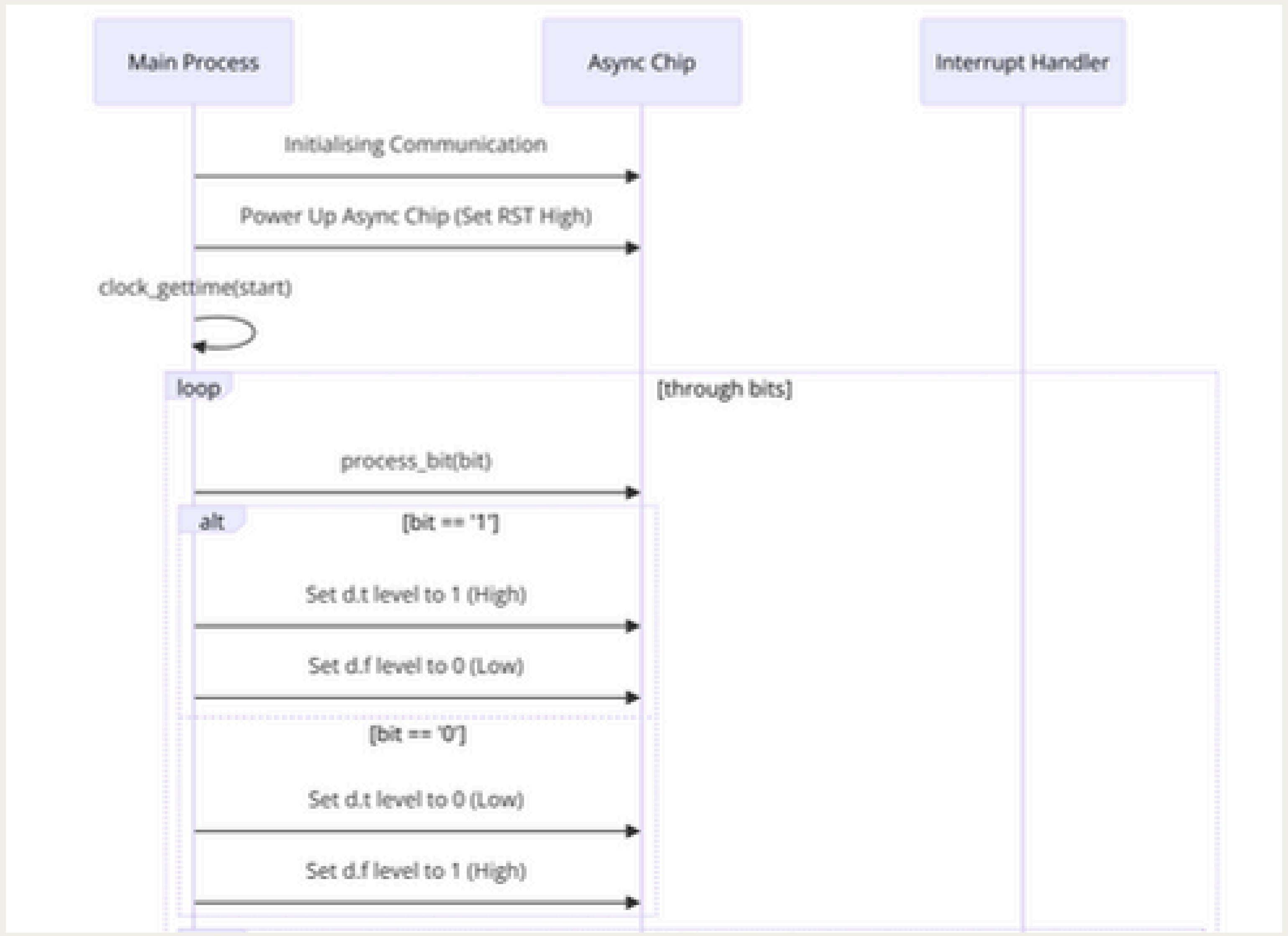
BRIDGE HARDWARE IMPLEMENTATION

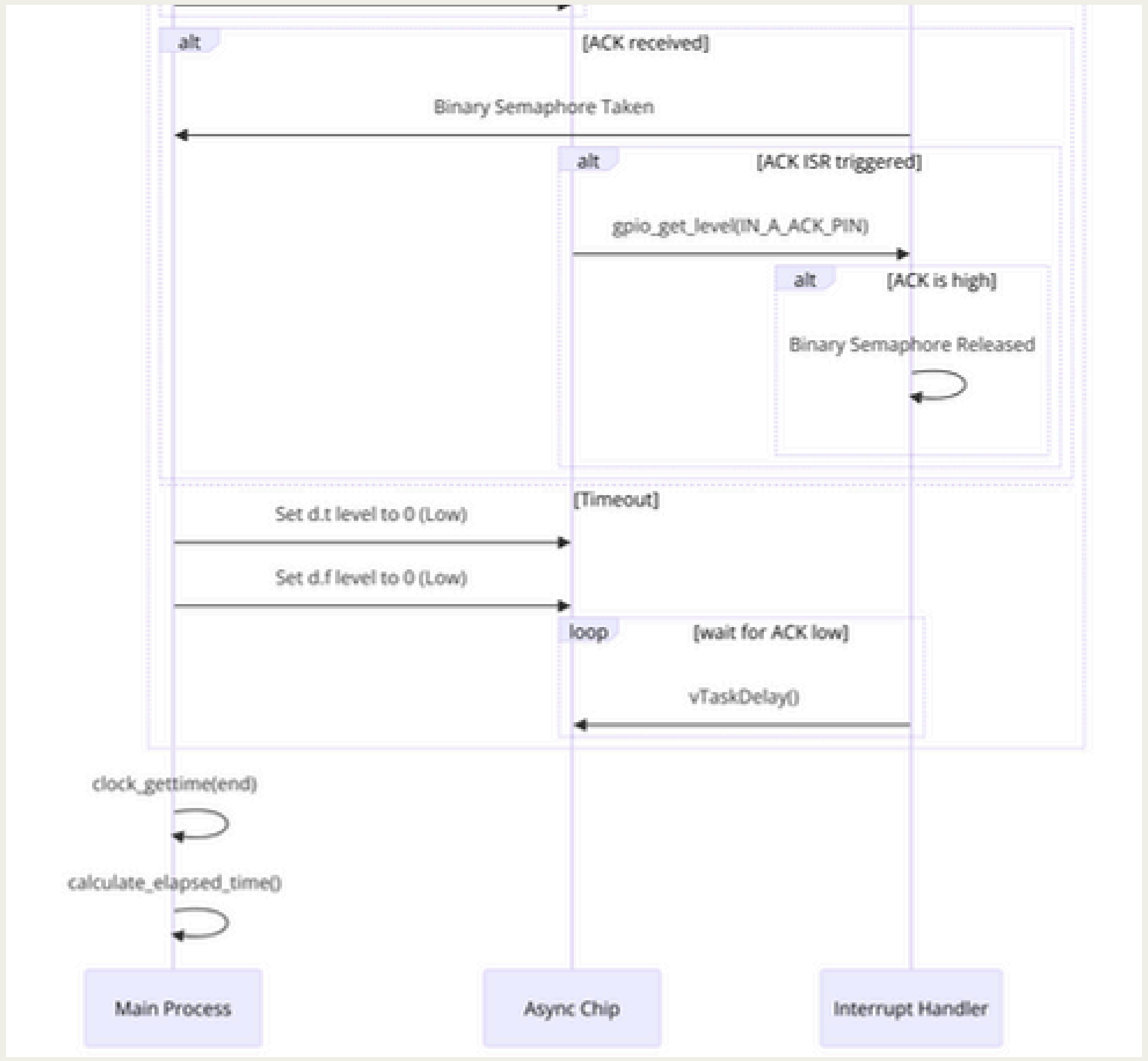
System Controller



BRIDGE HARDWARE IMPLEMENTATION

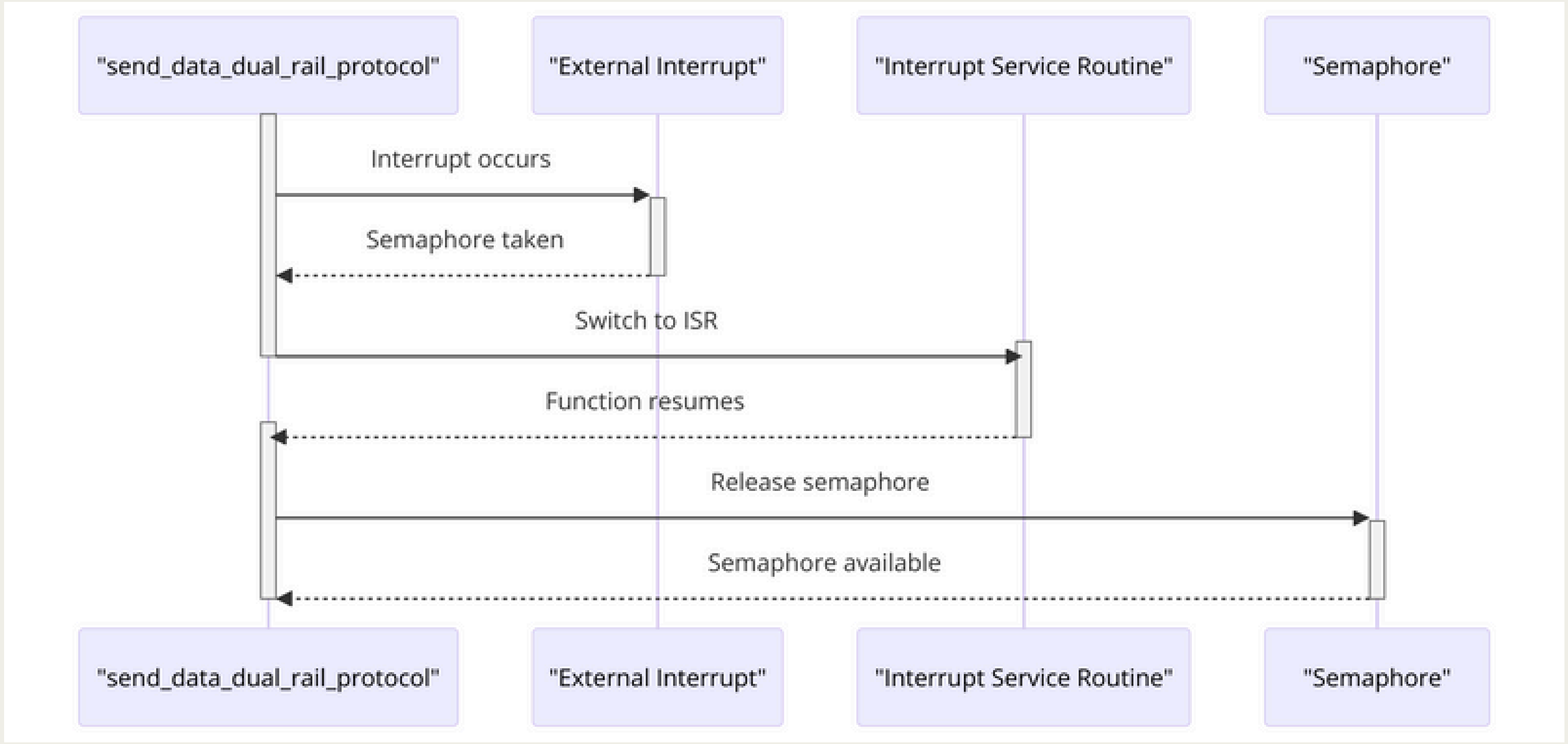
4-Phase Dual-Rail Protocol Implementation



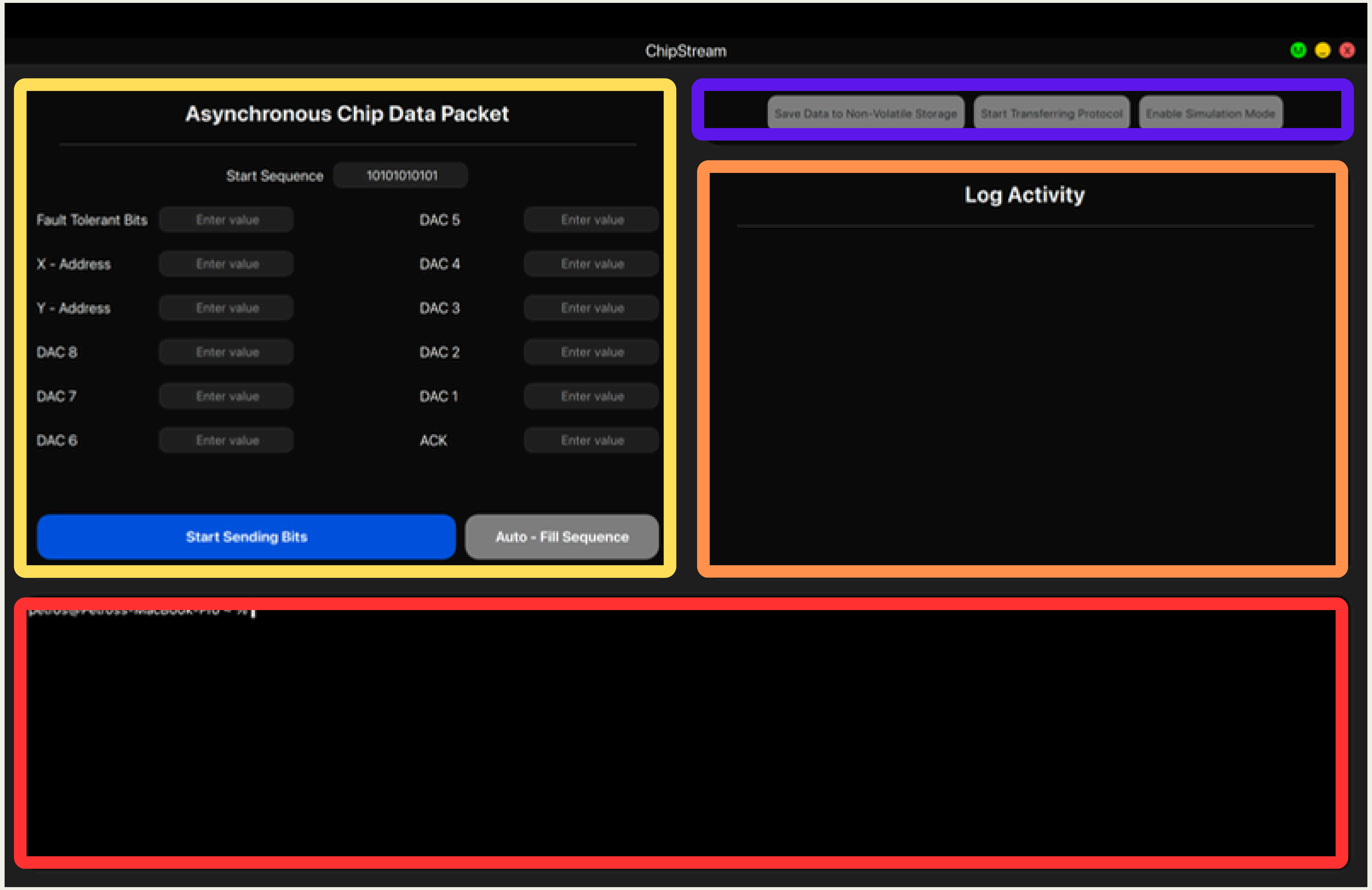


BRIDGE HARDWARE IMPLEMENTATION

Event Driven Mechanism for ACK



USER SOFTWARE IMPLEMENTATION



Asynchronous Chip Data Packet

Start Sequence: 10101010101

Fault Tolerant Bits	Enter value	DAC 5	Enter value
X - Address	Enter value	DAC 4	Enter value
Y - Address	Enter value	DAC 3	Enter value
DAC 8	Enter value	DAC 2	Enter value
DAC 7	Enter value	DAC 1	Enter value
DAC 6	Enter value	ACK	Enter value

```
petros@Petross-MacBook-Pro ~ %
```

SYSTEM TROUBLESHOOT

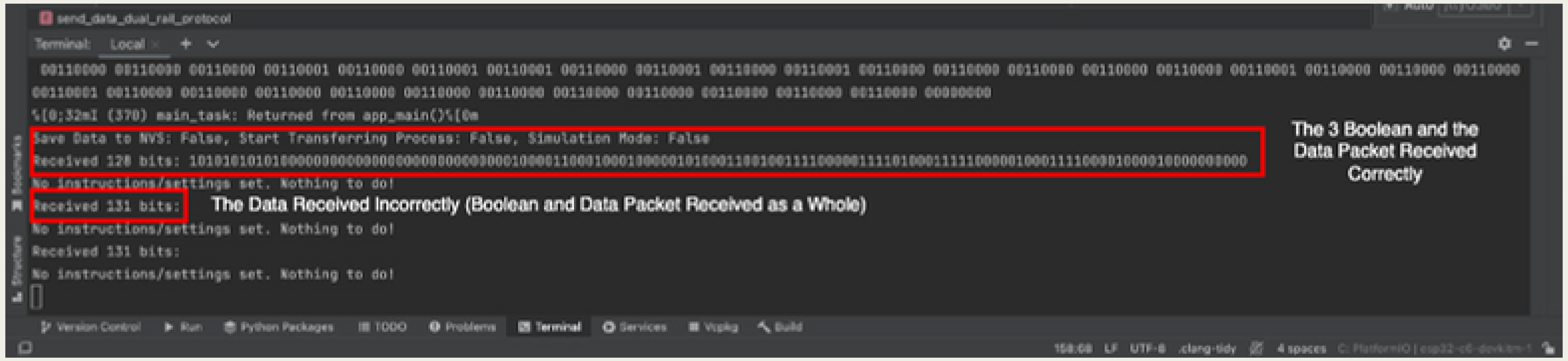
Improper Data Reception and Handling from Bridge Hardware

Correct Data Reception

- 3 Bits for Configuration Settings
- 128 Bits of Data

False Data Reception

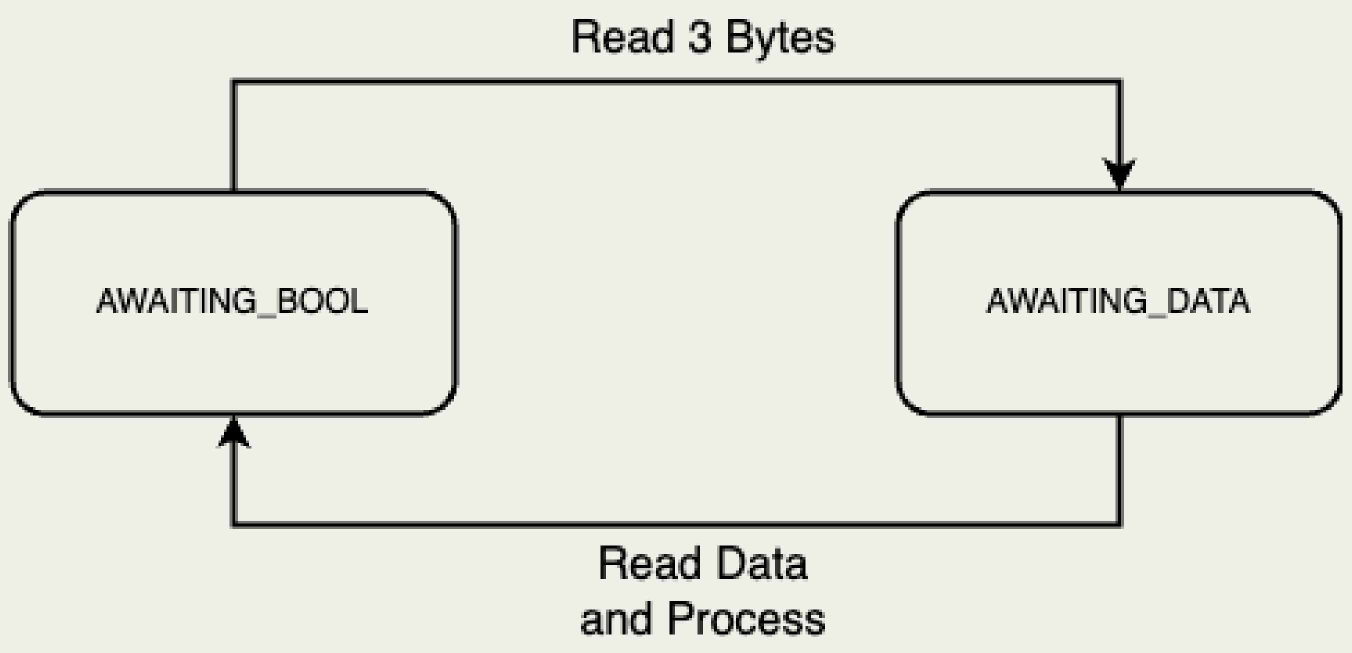
- 131 Bits as a whole



```
 00110000 00110000 00110000 00110001 00110000 00110001 00110001 00110000 00110001 00110000 00110001 00110000 00110000 00110000 00110000 00110000 00110001 00110000 00110000 00110000
00110001 00110000 00110000 00110000 00110000 00110000 00110000 00110000 00110000 00110000 00110000 00110000 00110000 00000000
%[0;32mI (370) main_task: Returned from app_main()%[0m
Save Data to NVS: False, Start Transferring Process: False, Simulation Mode: False
Received 128 bits: 10101010101000000000000000000000000000000100001100010001000001010001100100111100000111101000111110000010001111000010000100000000000
No instructions/settings set. Nothing to do!
Received 131 bits:
No instructions/settings set. Nothing to do!
Received 131 bits:
No instructions/settings set. Nothing to do!
```

send_data_dual_rail_protocol
Terminal: Local
The 3 Boolean and the Data Packet Received Correctly
The Data Received Incorrectly (Boolean and Data Packet Received as a Whole)
Version Control
Run
Python Packages
TODO
Problems
Terminal
Services
Vcpkg
Build
158:68 LF UTF-8 .clang-tidy 4 spaces

Solution



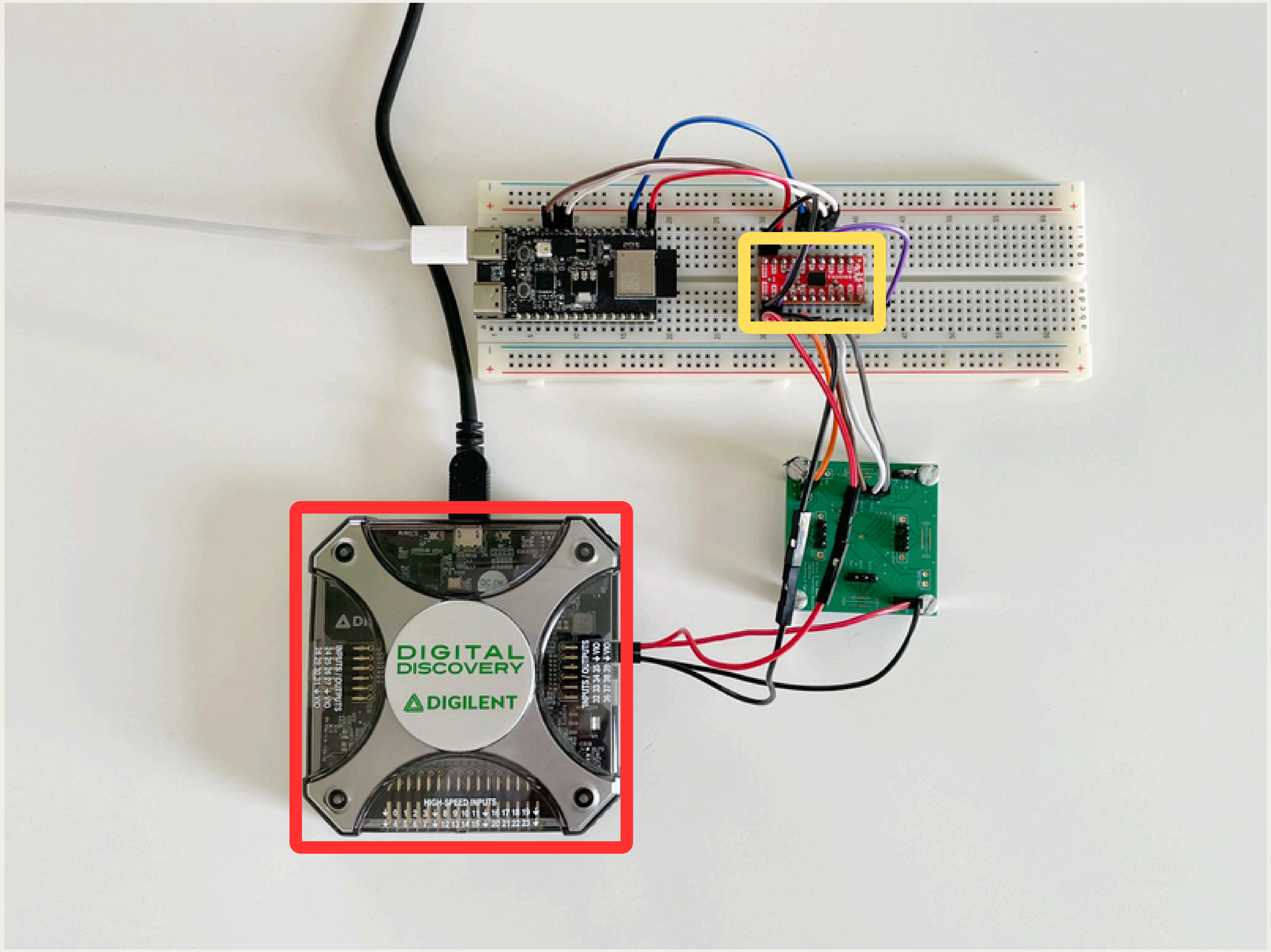
Transitions:

• Transition from AWAITING BOOL to AWAITING DATA after processing 3 bytes of boolean values.

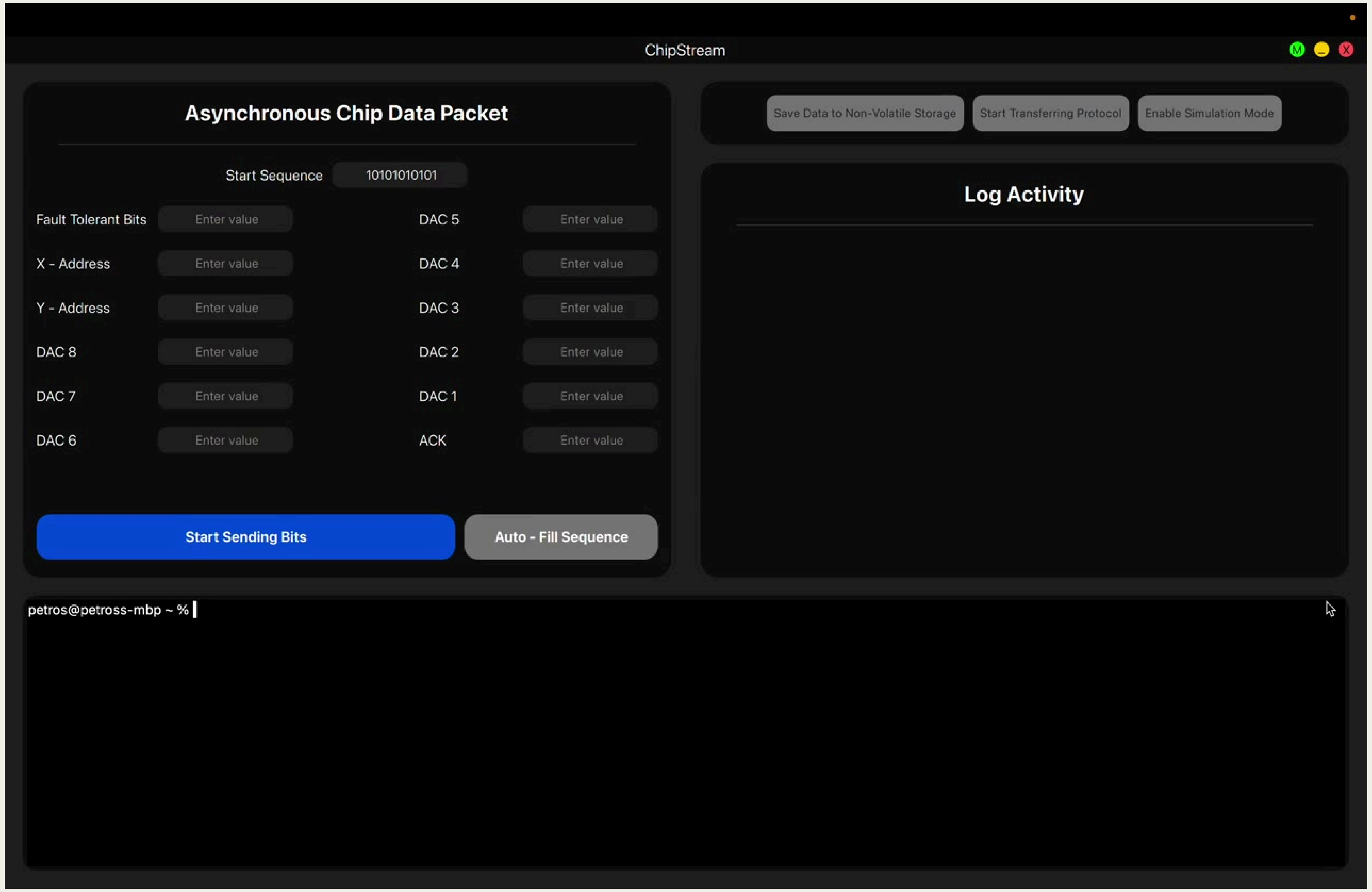
• Transition from AWAITING DATA to AWAITING BOOL after processing regular data bytes.

Note: Between each transition the UART Buffer cleans any leftover data

SYSTEM SETUP



SYSTEM DEMO



Asynchronous Chip Data Packet

Start Sequence: 10101010101

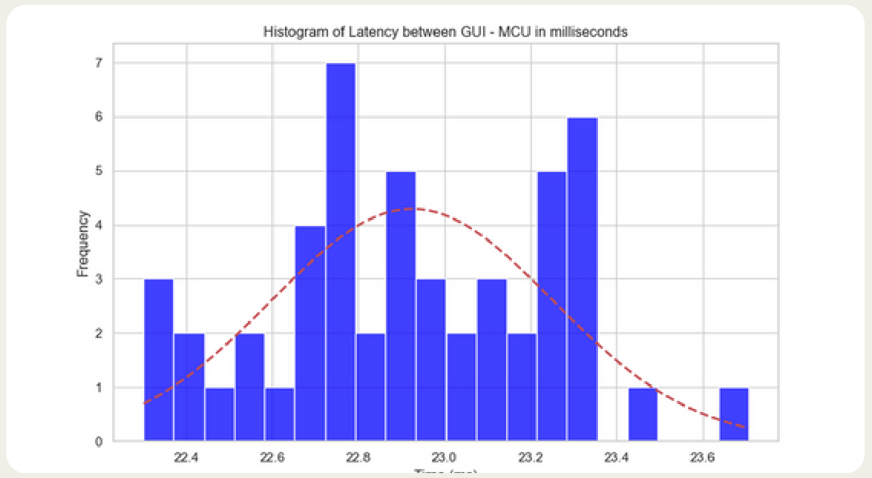
Fault Tolerant Bits	Enter value	DAC 5	Enter value
X - Address	Enter value	DAC 4	Enter value
Y - Address	Enter value	DAC 3	Enter value
DAC 8	Enter value	DAC 2	Enter value
DAC 7	Enter value	DAC 1	Enter value
DAC 6	Enter value	ACK	Enter value

Log Activity

```
petros@petross-mbp ~ %
```

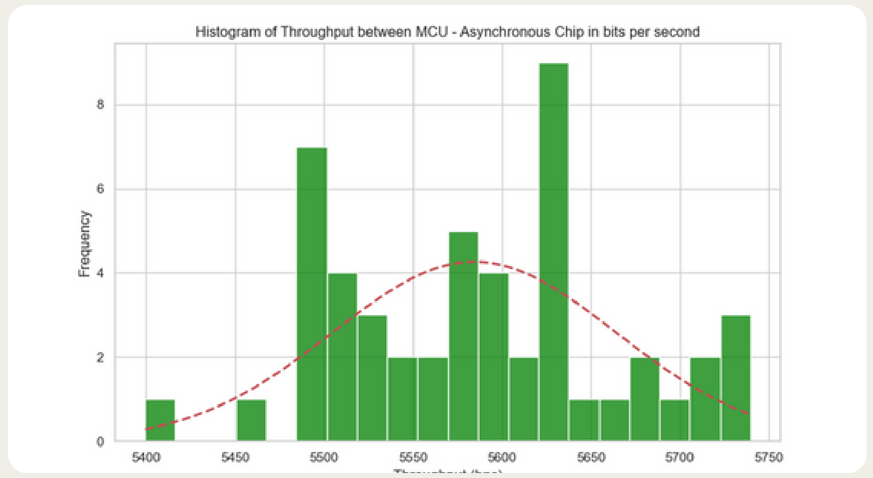
PERFORMANCE EVALUATION

Latency GUI - MCU



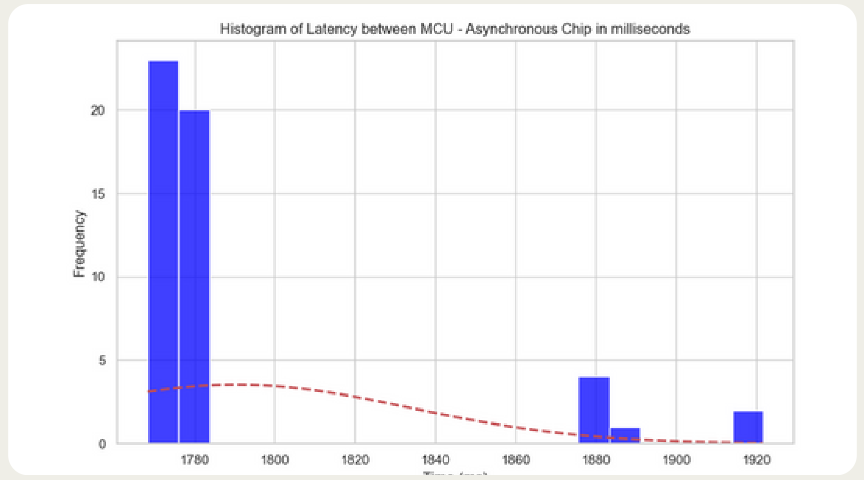
Clock Period: 1 ns
Min Value: 22.301 ms
Max Value: 23.706 ms
Mean Value: 22.925 ms
Std. Deviation: 0.326 ms

Throughput GUI - MCU



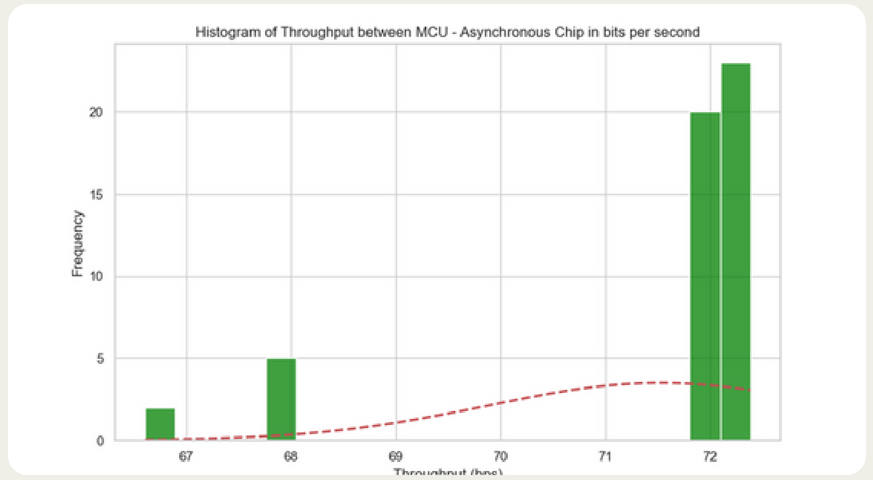
Throughput = Data / Time
Min Value: 5526.03 bps
Max Value: 5874 bps
Mean Value: 5715.41 bps
Std. Deviation: 81.44 bps

Latency MCU - Asynchronous Chip



Clock Period: 1 μs
Min Value: 1768.397 ms
Max Value: 1921.751 ms
Mean Value: 1790.476 ms
Std. Deviation: 43.274 ms

Throughput MCU - Asynchronous Chip



Throughput = Data / Time
Min Value: 66.61 bps
Max Value: 72.38 bps
Mean Value: 71.53 bps
Std. Deviation: 1.64 bps

Thank You!

Any Questions?

Petros Petrou
Undergraduate Student
Department of Electrical and Computer Engineering

University of Cyprus
Department of Electrical and Computer Engineering

CONTENTS

- Introduction

Part I: Foundational Concepts and Theoretical Background

- Evolution of Semiconductor Technology
- Synchronous versus Asynchronous Systems
- High Level Languages in Asynchronous Circuits

Part II: Comparative Technical Exploration and Hardware Selection

- Hardware Selection
- Communication Establishment
- Engineering of the Software

Part III: Practical Implementation, Evaluation and Synthesis

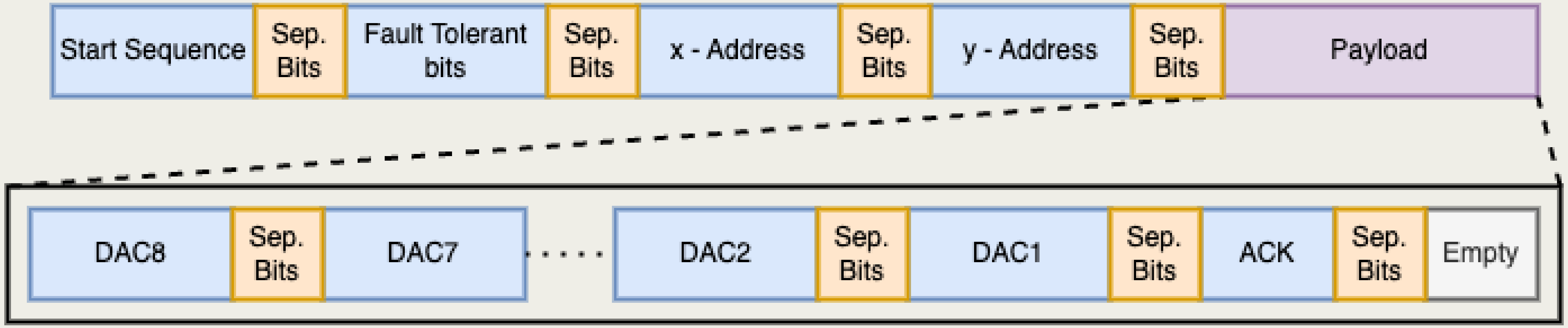
- Implementation, Debugging, Testing
- Performance Evaluation

SERIAL COMMUNICATION

Chosen communication Protocol: UART Protocol

	UART Protocol	**Other Protocols (SPI, I2C etc...)**
Communication Type:	Asynchronous	Synchronous
Protocol Complexity	Simple, no need for a shared clock signal, minimal wiring (2 lines)	More complicated implementation across devices, more wiring lines
Device Compatibility	Wide variety of MCUs and FPGAs support UART natively	Still a lot of MCUs and FPGAs support those protocols, yet the complexity is higher
Data transfer management	Straightforward	Requires more management and knowledge
Ease of Use	Generally simpler to program and integrate, ideal for specific tasks and beginners	More complex, suitable for advanced users and complex applications.

DATA PACKET FORMATION



Header Section

- Start Sequence (11)
- The Start Sequence is always initialised to the binary number(10101010101)2.
- Fault-Tolerant bits (2)
- x-Address Bits (9)
- y-Address Bits (9)

Separation bits (2)

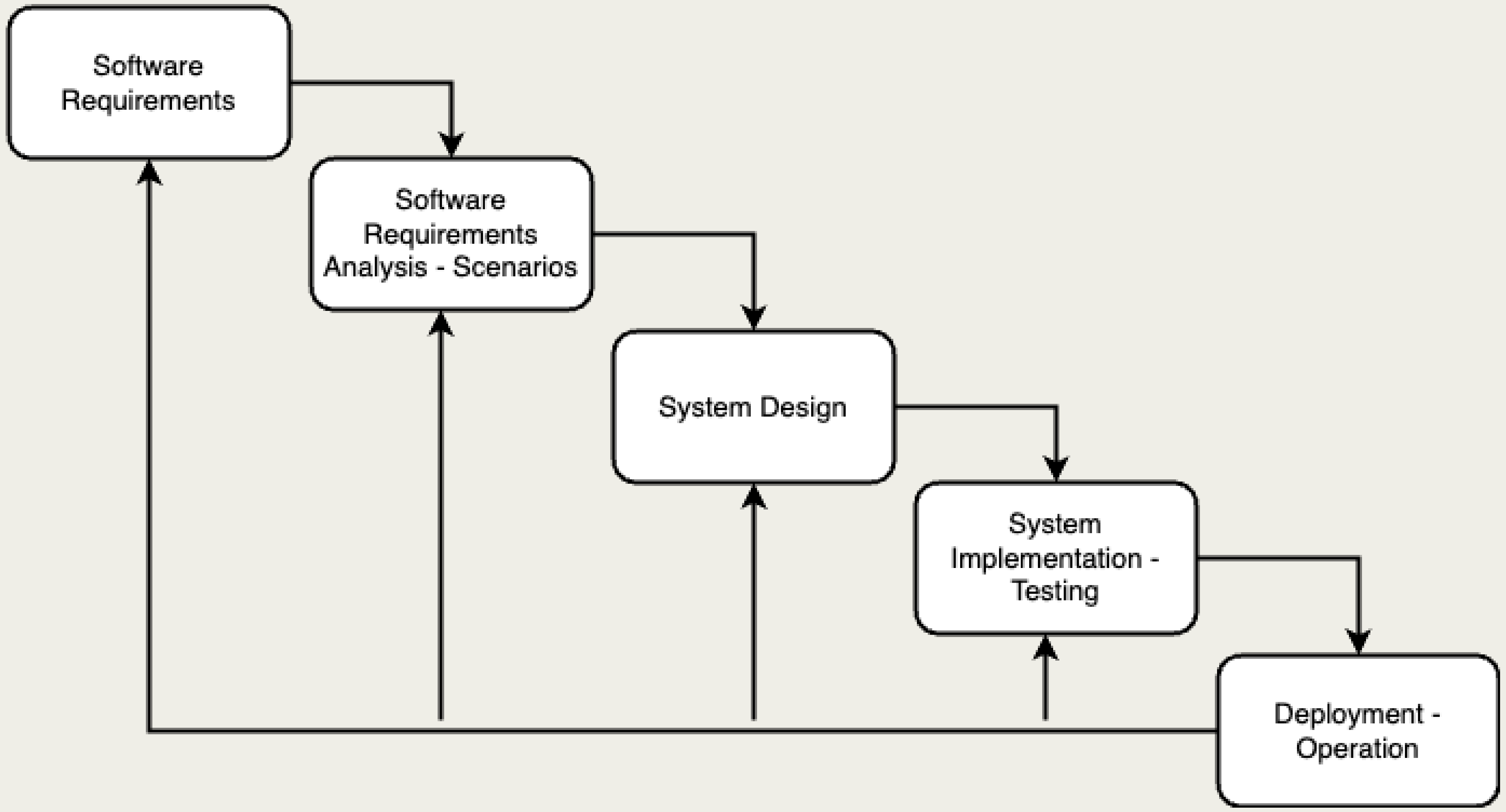
Null bits, always initialized as (00)2

Payload Section

- dac8 - dac3 (8)
- dac2 - dac1 (9)
- ACK (3)
- Empty (2)

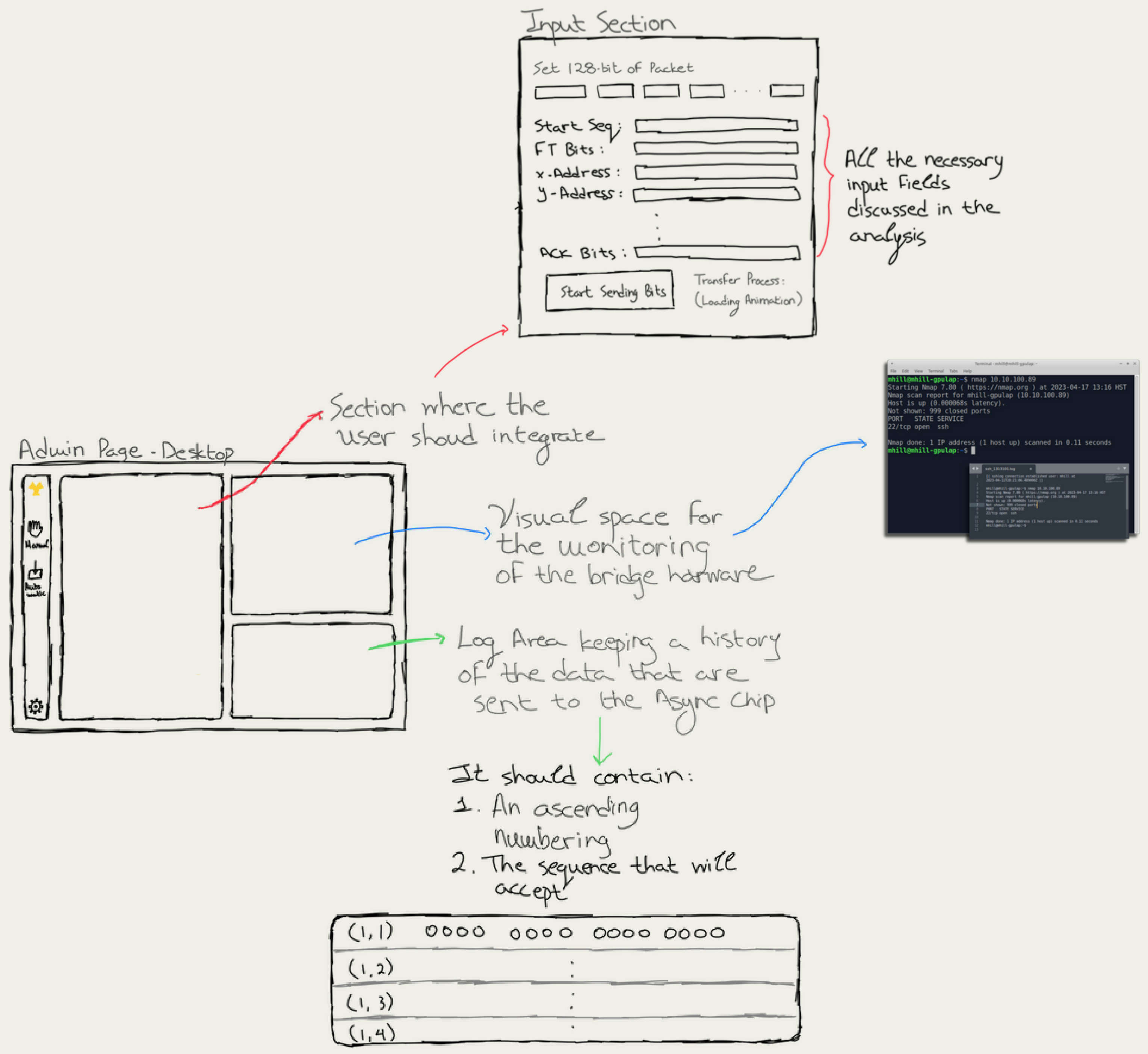
ENGINEERING OF THE SOFTWARE

Selected Software Development Model:

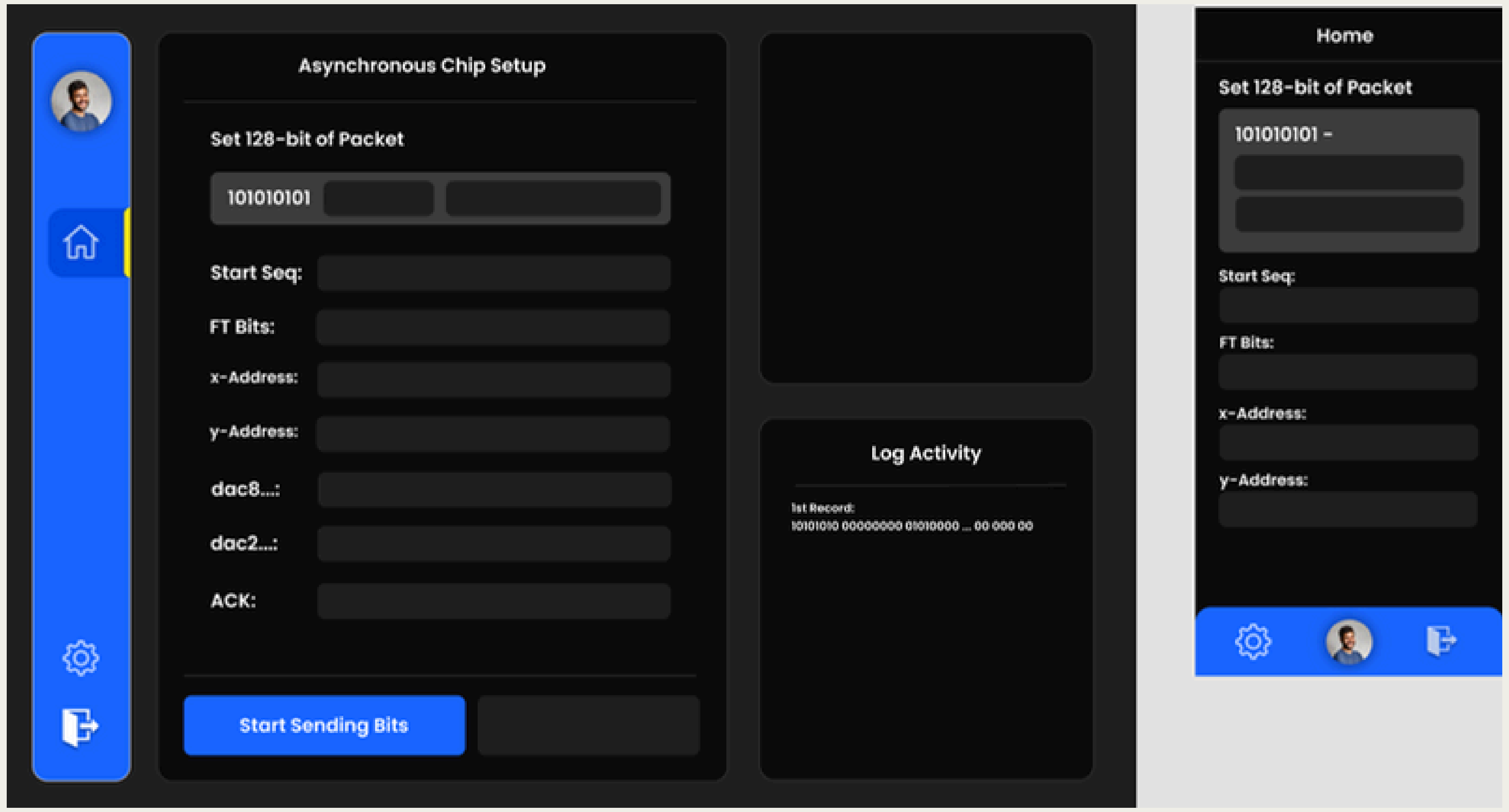

Software Requirements
Software Requirements Analysis - Scenarios
System Design
System Implementation - Testing
Deployment - Operation

ENGINEERING OF THE SOFTWARE

Software Wireframe



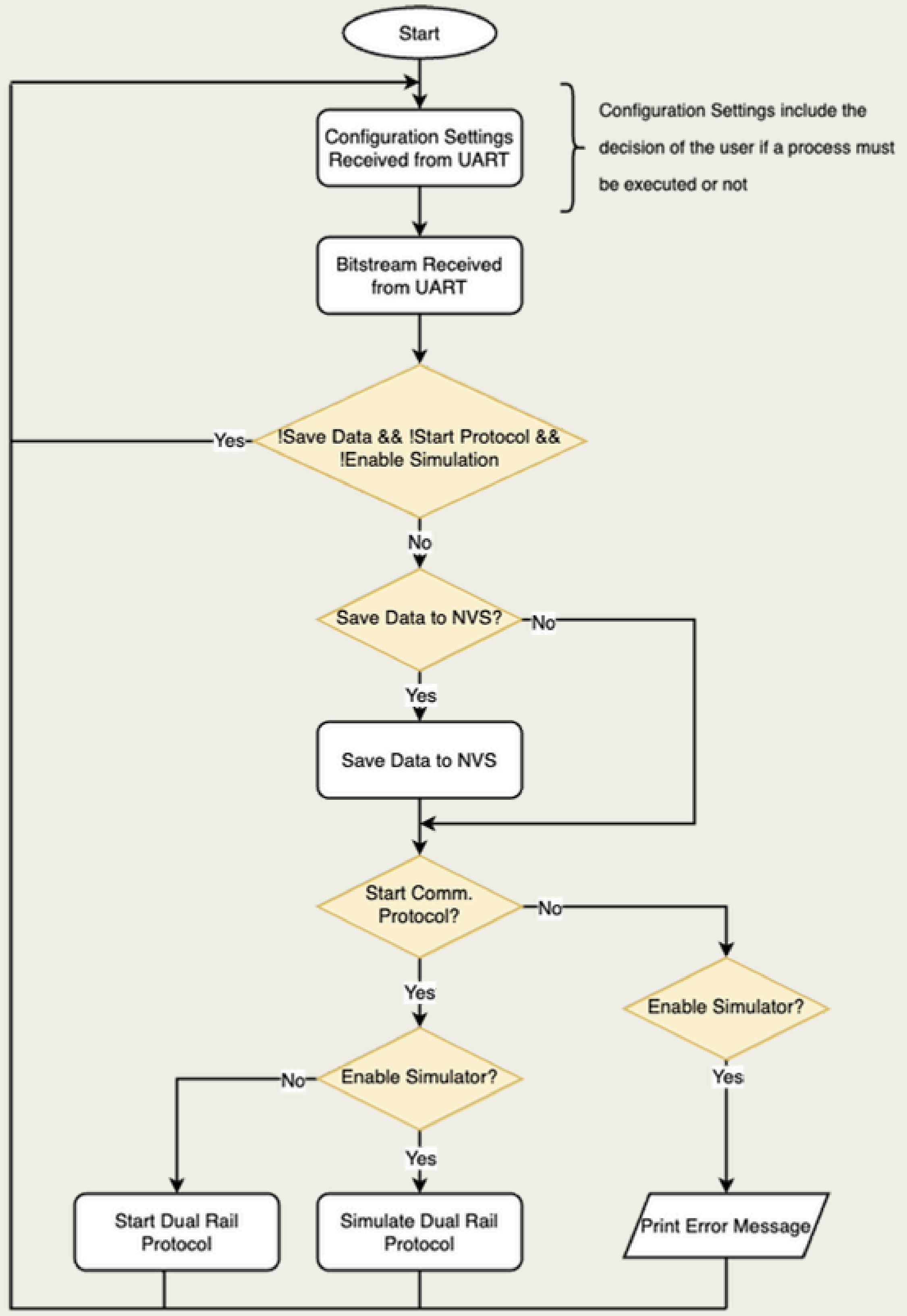
Input Section
Set 128-bit of Packet
Start Seq:
FT Bits:
x-Address:
y-Address:
ACK Bits:
Start Sending Bits
Transfer Process: (Loading Animation)
All the necessary input Fields discussed in the analysis
Section where the user shoud integrate
Admin Page - Desktop
Visual space for the monitoring of the bridge harware
Log Area keeping a history of the data that are sent to the Async Chip
It should contain:
1. An ascending numbering
2. The sequence that will accept
(1,1) 0000 0000 0000 0000
(1,2)
(1,3)
(1,4)



ENGINEERING OF THE SOFTWARE

User's Configuration Settings

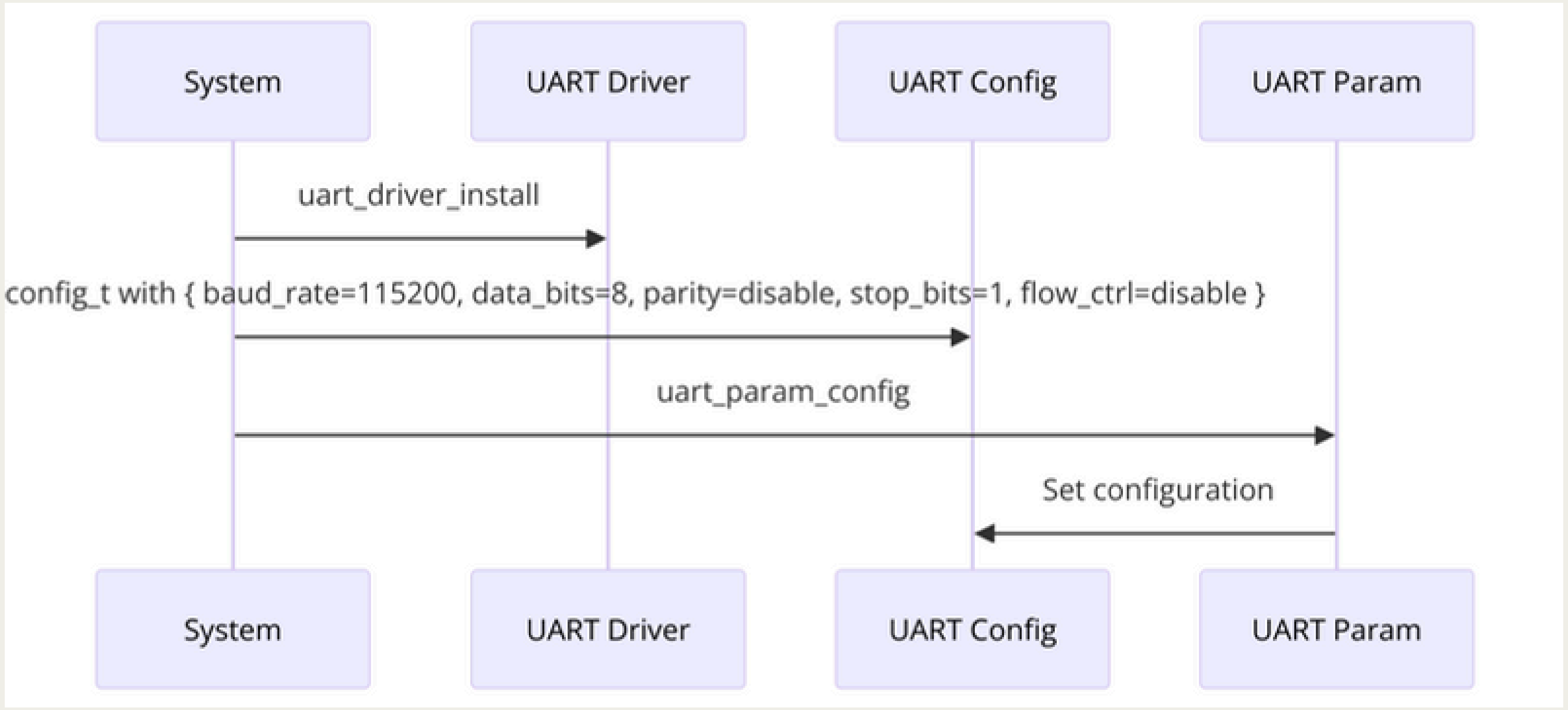
- Save Data to NVS
- Start Communication Protocol
- Enable Simulator



Start
Configuration Settings Received from UART
Configuration Settings include the decision of the user if a process must be executed or not
Bitstream Received from UART
!Save Data && !Start Protocol && !Enable Simulation
Yes
No
Save Data to NVS?
No
Yes
Save Data to NVS
Start Comm. Protocol?
No
Yes
Enable Simulator?
No
Yes
Enable Simulator?
Yes
Start Dual Rail Protocol
Simulate Dual Rail Protocol
Print Error Message

BRIDGE HARDWARE IMPLEMENTATION

UART Initialisation


System
UART Driver
UART Config
UART Param
uart_driver_install
config_t with { baud_rate=115200, data_bits=8, parity=disable, stop_bits=1, flow_ctrl=disable }
uart_param_config
Set configuration
System
UART Driver
UART Config
UART Param