University of Camerino

MASTER DEGREE OF COMPUTER SCIENCE (LM-18)

Course in Course in Complex System Design

Simulation Framework for Performance and Load Evaluation of Blockchain-integrated Embedded Systems

Supervisor
Corradini Flavio

Group
Lolli Andrea
Perticaroli Thomas
Draibine Adnane

External Supervisors
Culmone Rosario
Petracci Riccardo

A.A. 2023/2024

Abstract

The aim of this work is to develop a detailed simulation of a complex embedded system designed to manage blockchain transactions and certify the origin of products, as in the case of fishing, where it is essential to certify the provenance of fish to guarantee quality. This system is based on coroutines, allowing efficient and concurrent management of transactions, and includes probabilistic, non-deterministic and deterministic events to represent the variations and uncertainties that can occur during operation.

The main objective of simulation is to faithfully reproduce the behaviour of the system, thus allowing stress tests to be carried out at various workloads. This approach is crucial for identifying the 'breaking load' of the hardware specification, that is the point beyond which system performance starts to degrade. The results of these simulations are used not only to assess the current performance of the hardware in use, but also to provide clear and practical indications regarding the need for any hardware upgrades.

This work aims to provide an overview of the dynamics of a coroutine and blockchain-based embedded system, highlighting the importance of realistic simulations to ensure optimal and reliable operation. The information obtained from the testing of the system will help to improve operational efficiency and ensure the certification of the origin of products, thus meeting the growing demands for transparency and traceability in the modern market.

Index

1. Introduction

The integration of embedded systems with blockchain technology offers unique opportunities in a fast-moving technological environment, especially in the areas of data security, traceability and operational efficiency. One particularly relevant application that we took into account, consists in the evaluation of physical objects: thanks to embedded systems, it is in fact possible to perform scans that capture crucial information, such as geographical location, in a decentralised and immutable manner. This capability is crucial in many areas related to supply chain management, logistics and quality assurance, where accurate tracking and verification of assets are subject to strict regulations.

The increasing complexity and scale of these systems requires a thorough understanding of the behaviour they exhibit under different operating conditions. This paper addresses this need by proposing a sophisticated simulation of an embedded system designed to inspect containers, boxes or products and record metadata on a blockchain[HCL23]. Through this simulation, it is possible to derive the constraints, functionalities and possible limitations of the system, obtaining significant information to improve its performance.

In particular, the simulation makes it possible to evaluate variations in data complexity and reliability of the embedded system in response to increasing demand. Furthermore, the research work was able to establish the system's operational limits and thresholds, advancing useful knowledge and recommendations that could lead to improved efficiency and overall performance. These findings also provide valuable insights into the field of embedded systems and their integration with blockchain technology, thus assisting developers in creating more robust and reliable systems for practical applications.

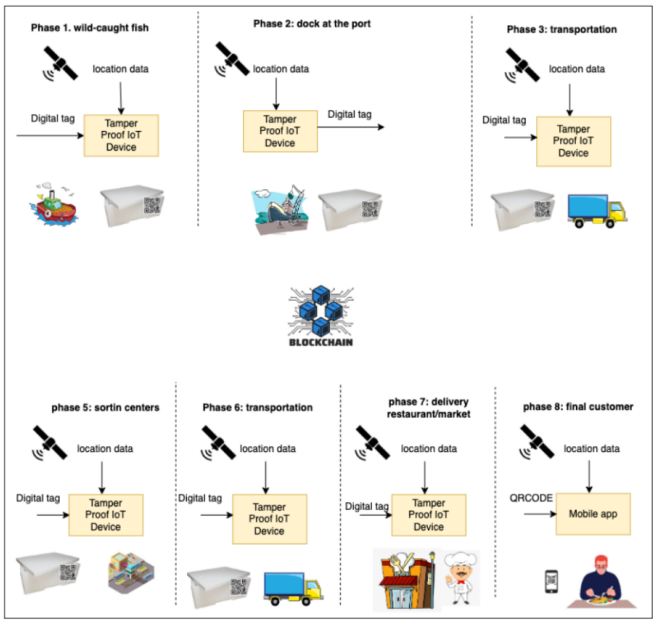


Figure 1.1: All phases of the system usage

1.1 Project Domain

This project covers the convergence of embedded systems, blockchain technology, and system simulation. Different domains are active in this area, which include:

1. **Design Study**: This research is related on the study of the design and development of processes running on embedded devices, aimed at scanning and analyzing physical objects. These processes manage the interaction between sensors, data collection systems, and processing units, handling data acquisition and analysis from the physical environment. The goal is to ensure efficient, real-time execution of tasks that support improved system functionality and overall performance.

2. **Simulation and Systems Engineering**: This field deals with the design and implementation of a simulation platform that is supposed to emulate the functionality of this particular embedded system (or other physical devices or systems). It will include data complexity analysis and its reliability under different loads. Simulation is a tool to understand system dynamics, find failure points, and optimize overall system efficiency.

3. **Benchmarking Performance and Identifying Bottlenecks**: Various stress tests will demonstrate the peak capacity at which the system operates, the exact points at which the bottlenecks occur, and from what level the performance is curtailed. The review confirms that the system will perform tasks effectively at peak loads and pinpoints the components that should be upgraded.

4. **Blockchain Data Recording**: Data would be recorded on the blockchain in a decentralized, immutable blockchain ledger for metadata, location, and inspection results. Integrity, traceability, and security are now guaranteed with this approach, which will be important in industries where verification and accountability constitute the very foundation.

1.2 Objectives

1. **Development of Advanced Simulation**: It should be designed to virtually recreate an embedded system targeted at box inspection and metadata recording on the blockchain. The simulation has to be performed in conditions as close to real as possible for extensive testing and analysis.

2. **Analyze system performance under different conditions**. Perform the performance evaluation of the whole embedded system based on varied conditions, such as data complexity and high demand for usage. Based on these evaluations, important conclusions can be obtained as to what is possible with this system and what is not.

3. **Delineate Where the System Breaks Down**: Find the maximum limit the system can bear before failure. This involves conducting a stress test on the system in order to find where it reaches its limit to fail, as this is pretty important to realize and understand the limit of the system for its real-world applications regarding reliability.

4. **Identification and Analysis of Bottlenecks**: Identify potential bottlenecks that could arise in the system, and how that would impact performance or scaling negatively. By understanding where the bottlenecks would occur and why they happen, enhancements of system architecture would be facilitated to enhance overall effectiveness.

5. **Define System Specifications and Operational Parameters**: Clearly spell out the exact performance specifications and bounds of operational parameters under which the system can operate effectively. This skeleton will then be used as a building block for further refinement of the system, which will enable the creation of newer and better versions that meet the necessary standards for real deployments.

6. **Enhancing Proximity in Embedded Systems**: The use of blockchain technology is presented in this research work, giving an overall description of key insights and developments related to integrating embedded systems with blockchain technology. The contribution hereby will add to the bigger discussion by underlining clear lines of how seamless integration can be achieved to ensure secure and reliable management of data in various domains.

7. **Preparedness for Practical Application**: This aspect in consideration, whatever inferences and improvements achieved with the simulated environment should be applicable in a real environment. This then gives the whole system feasibility of actual implementation in practical fields of supply chain management, logistics, and quality assurance among others.

1.3 Project Timeline and Work Distribution

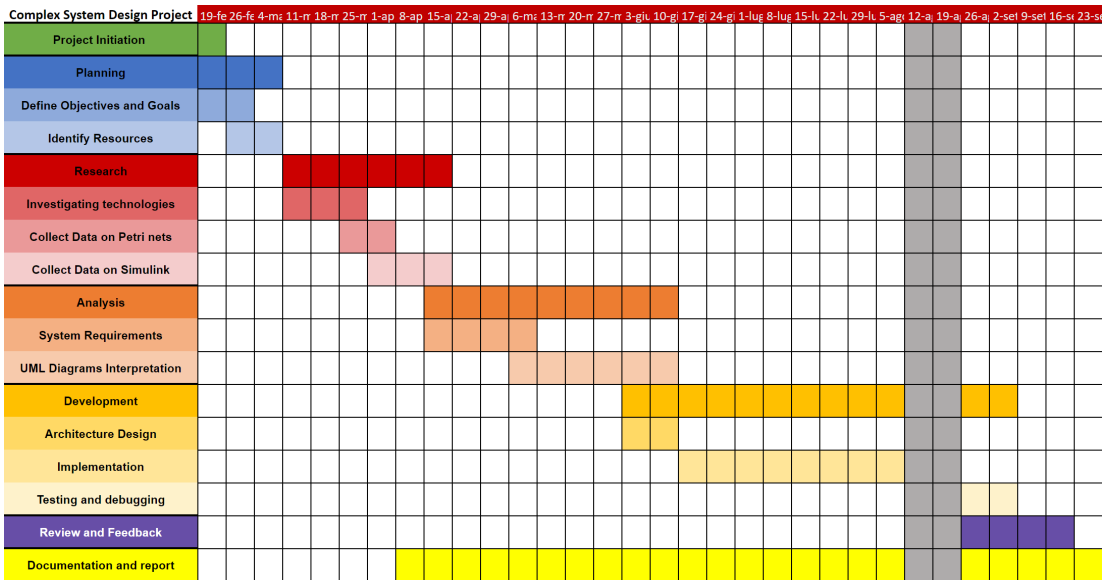


Figure 1.2: Gantt Diagram

The Gantt chart above is based on projected stages that we planned to reach. It is needed to take into consideration that the expected input from us would be for a maximum of only two hours per day, due to the fact that we work. This consequently impacted on the duration of the project.
The activities were divided in such a way that all the members would get a share of the activities involved; thus, everyone contributes to every step in knowledge sharing.

1.3.1 Phases explanation

1. **Project Initiation**: Building the basic structure through understanding the scope of the work and planning how to execute it by meeting professor Culmone and Riccardo Petracci.

2. **Planning**: Well-defined and precise goals are set out together with quantifiable objectives, considering the analytical framework and the methodologies designed for application on the integrated device.

3. **Research**: Conducted investigation into relevant technologies; gathered information related to the simulation tools. Following these, we examine the suitability of using each of them for the system represented by UML diagrams.

4. **Analysis**: The ongoing study regarding the needs of the system resulted in careful evaluation of the UML diagrams, concentrating on the actual processes running on an embedded system.

5. **Development**: In this stage, the main efforts are directed at architectural design and the many processes needed to be developed. Then, testing and debugging was carried out to find out whether the implementation was functional as expected.

6. **Review and Feedback**: Input was solicited and integrated into the system to refine the solution for the project's objectives.

7. **Documentation and report**: A lot of documentation was produced during the whole development process, which completed during the last quarter of the year and was reorganised in this report.

2. Theorical Background and Fundamentals

2.1 Embedded Systems

Embedded systems are dedicated computational architectures for specific functions of larger systems. Unlike general-purpose computers, which can run a wide range of applications, embedded systems are designed to implement only a limited set of operations with superior performance, power efficiency, and reliability. These systems span from everyday consumer electronics, industrial machinery, automotive, and medical devices. It generally has a microcontroller or microprocessor as its centerpiece, which mostly functions like the CPU of an embedded system. This CPU itself is integrated into a single chip with other elements such as memory-RAM, ROM, and flash-I/O interfaces, and peripherals like sensors, actuators, and communication modules. The hardware is also, in most cases, small and resource-limited; this naturally dictates that the software (or more correctly firmware) is of very high efficiency and optimized for low power consumption, limited processing resources, and real-time operation.

Embedded systems are designed to guarantee high reliability combined with real-time efficiency. Many applications, in particular those that are safety-critical, like automotive control systems and medical devices, require that the system respond to inputs in a known time so as to avoid intermittent failures or hazards. This need for real-time responsiveness requires that controlled systems have non-arbitrary timing behavior; the time at which a system reacts to its input signals must be predictable and repeatable. Software for embedded systems is typically developed in specific programming languages, like C or assembly language, which give direct access to and detailed control of all hardware components. However, since embedded systems are becoming more and more complex, higher-level programming languages and development environments are increasingly being used to manage this rise in complexity efficiently. Furthermore, a Real-Time Operating System (RTOS) is often used to manage the execution of tasks and ensure that critical activities meet their scheduled deadlines.

A defining feature of embedded systems is that they are part of a larger system. For instance, in a car, there is an embedded system controlling the engine, brakes, infotainment, and navigation. All these various sub-systems interact with each other to ensure not only the functional efficiency but also the safety of the vehicle as a whole.
With the advancement in technologies, the evolution of embedded systems allows them to embed more advanced functionalities, including connectivity through IoT, advanced data processing capabilities, and interaction with AI and ML algorithms. This is widening the circle of embedded systems, and their function shifts from simple control oper-

ations to higher-level complex autonomous decision-making processes. In other words, embedded systems form the basic architecture on which modern technologies depend to function properly, reliably, and for intended purposes. A proper balance in hardware-software optimization, real-time performance, and integration into larger systems must be struck while designing a system; therefore, this field inherently demands a niche for research and development in computing and electronics.

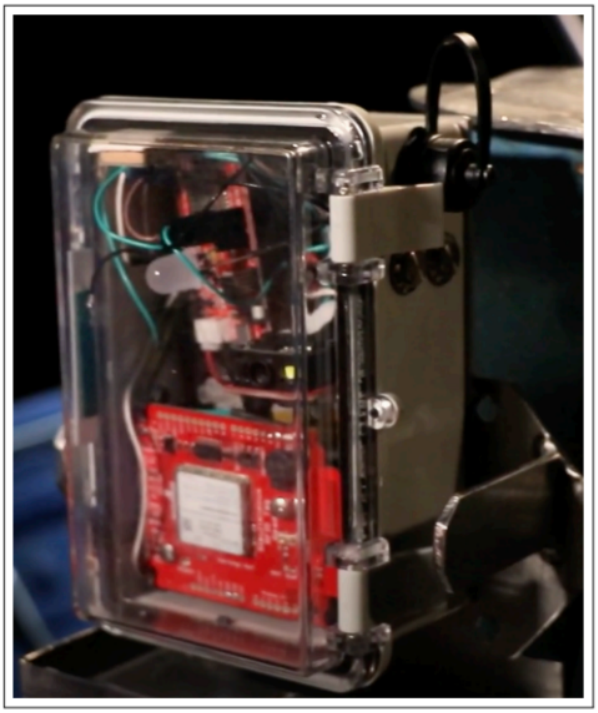


Figure 2.1: Image of the embedded system in question

2.1.1 ESP32

The ESP32 [Esp] is a very powerful, multi-capable controller that Espressif Systems design, that is used in the embedded systems we are studying. notable for its dual-core processing architecture, multi-interface connectivities, and extremely low power consumption. Thereby, this microcontroller gained huge interest among amateur developers, engineers, and programmers in applications such as but not limited to IoT devices, wearables, home automation solutions, and beyond.

Key Features

1. **Dual-Core Microprocessor**: Fundamentally, the ESP32 is based on a dual-core Tensilica Xtensa LX6 microprocessor; it runs at frequencies up to 240 MHz. Such a characteristic feature helps this device to perform many parallel tasks, which makes it in some situations fit for applications that require multitasking or simultaneous processing.

2. **Integration of Wi-Fi and Bluetooth Technologies**: One unique characteristic of ESP32 is its on-board connectivity: 2.4 GHz Wi-Fi-IEEE 802.11 b/g/n-and Bluetooth 4.2 BLE, or Bluetooth Low Energy. Thus, it is one of the best choices

for IoT projects where wireless communication will be necessary for devices to connect to the internet or with one another.

3. **Reduced Energy Consumption**: ESP32 was designed with an emphasis on energy efficiency. It integrates multiple active power-saving modes, deep sleep, light sleep, and modem sleep, which enable the device to reduce its power consumption down to mere microamps. This feature makes it very suitable for applications powered by a battery that require long operating times.

4. **Rich Set of Peripherals**: The ESP32 is a very powerful device with an array of peripherals. More than anything, it contains GPIOs, ADCs, DACs, PWM channels, timers, SPI, I2C, I2S, UART, and many more. These peripherals share the capability to interface with a wide array of sensors, actuators, and other hardware elements.

5. **Integrated Security Features**: Most of the embedded applications, especially those involving IoT, have several security concerns. For this purpose, ESP32 features hardware acceleration of encryption, such as AES and RSA, secure boot, and flash encryption that keep the data and firmware safe from unauthorized access.

6. **Development Ecosystem**: One of the advantages of ESP32 is its well-established and comprehensive development ecosystem. Espressif officially provides the ESP-IDF, which stands for the Espressif IoT Development Framework. More than this, compatibility with Arduino IDE, PlatformIO, and lots of other popular development platforms make the ESP32 much more user-friendly, both for beginners and for advanced developers.

Applications

The ESP32's flexibility positions it as a suitable platform for very many applications. Backed with it, IoT can enable intelligent devices to operate in homes, such as thermostats, burglar security systems, and lighting control devices, and transmit on either Wi-Fi or Bluetooth. Its low-power modes further make it an excellent choice for battery-operated devices like wearables, which need maximum energy efficiency. Due to the strong processing capability and support for various peripherals within the chip, the ESP32 can also run more complex functions. It enables real-time data processing, audio processing, and machine learning inference directly at edge devices. The ESP32 represents a powerful and affordable microcontroller; it sports excellent performance, extensive connectivity options, and very low power consumption. ESP32 is the best option when considering a broad spectrum of embedded applications in the fast-growing field of the IoT.

2.2 Static and Dynamic Views in System Design

In system design, a static and dynamic view of the system is important to gain full understanding of how components are configured and their functionality in operation. These two views will allow a designer to model the system correctly, analyze it, and also make improvements to the same.

Static View

The static aspect refers to a diagrammatic or structural relationship of elements that exist in the system and relationships among them. It focuses primarily on the structure of the constituents of the system, comprising modules, objects, tasks, or processes, together with their relationships. In a static view, a system's architecture, showing the location of different components, their relationships, and also any dependencies or hierarchies that might be in place, becomes the main concern.
For example, the UML diagrams utilized in our project provide a static perspective that delineates the distinct tasks, including GPS data retrieval, scanning operations, and blockchain transactions, along with their interconnections. This representation illustrates the overarching architecture of the system — highlighting elements such as coroutines and their role in managing system processes — while it does not yet encompass the dynamic behavior of these processes over time.
The static perspective proves particularly useful for understanding the overall structure of the system, and describing the architecture without concern for specific operational details. This viewpoint is essential for identifying how components interact with each other, ensuring modularity, as well as for determining each component's contribution to the overall architecture.

Dynamic View

In contrast, the dynamic perspective focuses on the behaviors and interactions of system components as they respond to a variety of inputs and events over time. It explores how actions by each component have an impact and how the entire system reacts to changes or operations in real time.
This view enables the investigation of the behavior of the system in a wide range of scenarios, such as the reaction of the embedded device to the freshly received GPS data, the way the device triggers a scan, and the way it elaborates a blockchain transaction. It traces the data flow and state transitions of the system, and the sequence of interactions among its parts.
Although the static perspective is essential for comprehending the structural relationships inherent within the system, the dynamic perspective is vital for assessing system performance, identifying bottlenecks, and guaranteeing that tasks are executed effectively and without errors such as deadlocks or data omissions. Such understanding and optimization of the behavior of complex systems require both static and dynamic viewpoints, especially when this includes an embedded device with real-time performance.

2.3 Asynchronous and Stochastic Events

Within general system modeling and analysis, especially those dealing with embedded systems and simulations, these concepts become more predominantly used in explaining the behaviors of systems when under changing conditions. These events represent two different types of happenings with respect to their influences upon a system in unpredictable or indeterminate ways.

Asynchronous Events

Asynchronous events are independent of the system control flow and any signals related to the system clock. They are intrinsically imprecise in their timing, and no pattern or schedule whatsoever can be ascribed to them. In other words, such events may pop up at rather unexpected times, triggered by external influences or certain conditions. Asynchronous events are a fact of life in embedded systems. The paradigmatic example of an asynchronous event is, of course, an external interrupt driven by a sensor. Whereas the system may know in advance that such events can happen, it does not have any prior knowledge about when they will, in fact happen. For this reason, a system should be developed to accommodate such events whenever it does happen.

Asynchronous events are usually handled by special methodologies, such as interrupts or event-driven processing, where the system is either in idle or executes other tasks until that sort of event happens. In our system, UML diagrams represent asynchronous events as the triggers of separate processes, themselves running in parallel with other activities. The importance of handling asynchronous events rests on the fact that the system ought to stay responsive to real-world events. As far as embedded systems are concerned, inefficiency and possible complete system failure could occur in cases where the asynchronous events are handled poorly or not at all.

Stochastic and Probabilistic Events

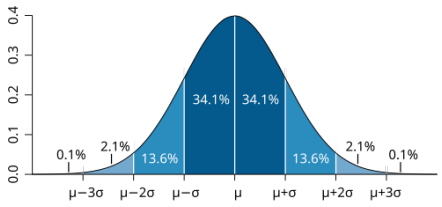


Figure 2.2: Probability distribution example

Stochastic events are those occurring at random, whose occurrence times are distributed in accordance with a probability distribution [Pro] of some type, without any a priori or deterministic cause. In fact, most events take place in conditions that feature stochasticity, especially for systems whose operation relies on incidents that, themselves, are driven by randomness or uncertainty, such as random processes that would emanate from sources of indeterminism, like traffic flow, weather, or environmental parameters, or user actions. Stochastic events are basic processes when systems interact with their environment - driven by random fluctuations. It provides the modeling of systems in a far more realistic manner, with the inherent indeterminism of real-life activities, which also allows increasing robustness in the design.

Stochastic modeling is widely regarded as a crucial approach within simulation methodologies to investigate and predict the dynamics of systems operating under conditions of uncertainty. This includes examples like random events, such as unexpected blockchain transaction delays because of network congestion, or random fluctuations in the reliability of GPS signals. Although these events or moments in time are impossible to forecast with complete certainty, the overall behavior is very often amenable to statistical modeling methods, such as probability theory.

2.4 Blockchain

The very idea of blockchain technology at first was designed and developed as an underlying technological platform on which Bitcoin was based, but it was introduced by a person - possibly a group of people - known as Satoshi Nakamoto. Since then, it has emerged as multi-faceted and a highly used asset in many fields. Basically, the blockchain is a decentralized and distributed ledger system that keeps track of transactions relatively fairly and transparently across its computing nodes. The most important points about it are:

1. **Decentralized**: The blockchain is inherently decentralized. Traditional systems involve various forms of banking and centralized databases that utilize a single authority or intermediary to validate the authenticity and authorization of transactions. On the other hand, a blockchain operates on a P2P network in which all participants are considered equal. Each node within this P2P network maintains a copy of the entire blockchain, wherein a blockchain is a continuously growing collection of blocks or records.
   The elimination of the need for intermediaries further decreases the risk of having a single point of failure, and this decentralization increases resilience and makes the system more transparent.

2. **Cryptographic Hash Functions and Immutability**: In essence, the very heart of blockchain security and integrity are founded on the concepts of cryptography. A block in a blockchain contains thousands of transactions, a timestamp, a nonce - a one-time random number - and a cryptographic hash related to the previous block. It is an example of a fixed-length string generated by a cryptographic hash function that ensures the uniqueness of the content of this particular block. Modifying the data in that block would result in a much different-looking hash, thereby breaking the blockchain.
   The linking with blocks through cryptographic hashes ensures immutability - the fact that once a block is added to the blockchain, it cannot be changed without changing all subsequent blocks and achieving consensus from the majority of the entire network. It is an inbuilt feature in maintaining data integrity in this blockchain and provides great enhancement of security against tampering and fraudulent activities.

3. **Blockchain Types**: Some blockchains are designed for different purposes:

   - *Public Blockchains*: Any user can access these blockchains. In most cases, they are decentralized and run on peer-to-peer networks. Familiar examples include Bitcoin and Ethereum. While public blockchains benefit from more security due to their large size and truly decentralized nature, they often expose reduced performance coming from power consumption and time to reach consensus.
   - *Private Blockchains*: These blockchains, operating within a confined group of users, are in fact the ones most businesses would use. They allow for a greater number of transactions per second and give greater control over the aspect of data privacy; this is at the expense of reduced decentralization.
   - *Consortium Blockchains*: This is the hybrid framework that amasses a big catch from both public and private blockchains. Consortium blockchains

are run by a group of organizations. They offer the best balance between transparency and security, and operational efficiency in industries where cooperation among more than one entity is important.

Fundamentally, blockchain technology is an advancement in storing and spreading data securely. The mixture of decentralization, methods of cryptography, and consensus mechanisms ensures that blockchain securely, transparently, and decentrally develops applications within diverse domains.

2.5 Embedded Systems Testing

The embedded system development life cycle assesses the most important steps, covering the testing of both hardware and software in different conditions.
Embedded systems have a wide variety of applications in safety-critical applications. Extensive testing cannot, therefore, be substituted for locating and fixing defects as well as confirming reliability. The nature of embedded systems - struck by high demands on processing time as a result of real-time processing and often severely limited resources - also calls for thorough testing as a normal standard procedure.

Categories of Testing in Embedded Systems

1. **Unit Testing**: Much emphasis is given on the testing of singular components or modules in software. In this case, each unit is tested separately to conclude if it functions as per set standards. Unit testing allows the developers to identify defects early enough, thereby reducing financial and temporal investments significantly in later stages of the testing process.

2. **Integration Testing**: Provides for effective interaction among the variety of modules or components that might exist within a system. Given the general context of embedded systems, it is generally expected that this encompasses the verification of interactions that involve an extensive range of both hardware and software elements, possibly including sensors, actuators, and communication interfaces.

3. **System Testing**: It includes the testing of the overall embedded system as a single entity, whether or not it meets the specifications laid down. The functional evaluation of the system, including whether or not it can achieve tasks required of it, and non-functional rating such as performance, security, and usability, among other factors, are also included.

4. **Stress Testing**: That means the operation of the system under conditions working towards stressing it to the limit, hence testing its performance under extreme limits - for instance, high load, temperature variation, or fluctuating power supply - thereby providing a measure for durability or reliability.

5. **Acceptance Testing**: The final stage of testing involves the implementation in the natural environment, which should ensure that the product meets users' needs and functions well in a natural setting.

2.5.1 Simulators

Generally speaking, simulators are important for developing and testing stages in embedded systems. A simulator can be defined as a software virtual testing tool that imitates an embedded system or parts of the embedded system in support of developers in order to assess and validate their designs before those are physically implemented in hardware.

Advantages of Employing Simulators

1. **Low-Cost Testing**: Such simulators help reduce the need for expensive prototype hardware and enable tests that can be conducted in a virtual environment. This is especially advantageous during early stages of development.

2. **Early Bug Detection**: A developer can find and fix such problems when they are small and tractable, rather than letting them grow into exponentially greater problems, by emulating the behavior of the system under development.

3. **Adaptive testing environment**: Simulators enable different test cases, such as the simulation of complex real-life scenarios that are very hard or impossible to reproduce in real conditions, among others.

4. **Progressive Development**: Simulators also make it easier for the iterative process of development to take place, by constant testing of the architecture and addition of new features.

2.5.2 Categories of Signals in Digital Signal Processing (DSP)

Digital Signal Processing[Dsp] forms a part of many embedded systems, especially those relating to communications, manipulation of audio, and control systems. Knowledge of the type of signal variety processed within a DSP is important in ensuring efficiency during the design of them.

Categories of Signals

1. **Analog Signals**: Continuous signals are those which are defined by their time variation and include such signals as audio waves, temperature records. Before being processed with a DSP, continuous signals are converted into digital signals using the ADC.

2. **Digital Signals**: Discrete signals are represented with respect to binary values or 0s and 1s. These are results from sampling and quantization of the analog signals and are processed through digital algorithms in DSP.

3. **Periodic Signals**: Signals that repeat at fixed intervals; examples are sine waves, and clock pulses. Periodic signals naturally occur in communications and in signal modulation applications.

4. **Aperiodic Signals**: One that does not exhibit periodic repetition. Examples of such signals include random noise, or transient signals such as speech. In general, non-periodic signals are more difficult for the DSP system to analyze and manipulate.

5. **Deterministic Signals**: These signals allow for a great degree of specificity in articulation through mathematical formulation and depict identifiable patterns and behaviors that can be easily recognized during processing and analysis.

6. **Randomized Signals**: It is quite common to encounter signals containing randomness and, hence, unpredictable detail. Familiar examples include thermal noise in electronic circuits. A very important part of stochastic signal processing inherently involves statistical considerations.

2.5.3 Petri Nets

The most general definition of a Petri net[LW09] is a mathematical framework and an analytical tool devised for the representation and investigation of distributed systems in embedded contexts. The applications of such systems are in particular appropriate when the processes in question run in parallel and include procedures that involve synchronization and resource allocation.

Application Domains of Petri Nets

- **Modeling Concurrency**: The Petri Nets have emerged to be an effective tool for representing systems containing concurrent processes. It can model the interactions and synchronizations between different processes.

- **Resource Allocation**: One of the most important properties of Petri Nets is their ability to model resource allocation and deallocation in a system and analyze these nets for deadlocks.

- **System Verification**: Systems are analyzed using the Petri Nets for verifying whether they will behave as expected for various kinds of scenarios. Analyzing the Petri Net model, developers can identify potential problems such as race conditions, deadlocks, or resource contention.

- **Performance Analysis**: The Petri Nets provide a good framework for performance analysis of embedded systems concerning process throughput and utilization of resources.

3. Analysis

3.1 System Workflow using UML

This section presents the analysis of the system through UML diagrams, provided by Riccardo Petracci in cooperation with Prof. Culmone, representing the workflow of the embedded system. The principal operations are scanning, acquiring GPS data, and processing blockchain transactions. These diagrams give a more detailed illustration of how such a system operate, from the initial setup to handling the specifics of the job at hand, like GPS data retrieval and recording transactions on the blockchain. This analysis will elaborate on the system architecture, interactions across different tasks, and their contributions to the overall functionality. This step is vital in order not to miss any possible bottlenecks, ensuring the efficiency and reliability of the system. The subsections that follow will elaborate on each diagram to show the flow, timing, and coordination between the different modules of the system.

3.1.1 Flowchart

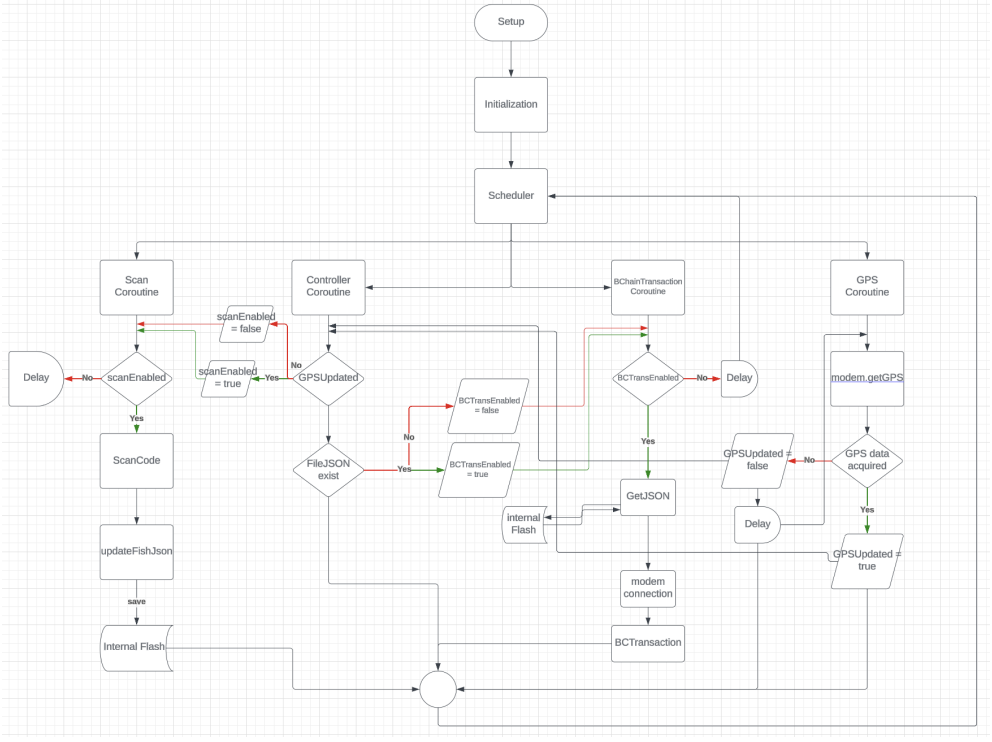


Figure 3.1: Flowchart

The operation of the system starts with the Setup stage, and it is followed by an Initialization phase in which it is supposed to configure all the default hardware and software settings. Similarly, as a conductor indicating when each coroutine should play based on specific conditions or events, the Scheduler should perform the activation of proper coroutines of the system.

Key Coroutines and their Responsibilities

- **Scan Coroutine**: The Scan Coroutine has the responsibility of keeping, internally, a JSON flash memory file updated. This coroutine is enabled by using a flag, namely the usage of a flag called scanEnabled; in case this flag is true, the system moves on with the scanning of the ScanCode block, which takes place here.
  It updates the JSON file, by which these results are persisted. This step is necessary for ensuing steps, especially with regard to blockchain transactions. In case the condition of scanEnabled is not met, then it enters into the delay state, whereby it stops the scanning.

- **Controller Coroutine**: The Controller Coroutine plays a supervisory role in the program, controlling the flow of data between various coroutines. It has a very significant role of checking the status of the GPS data update, GPSUpdated. Once the GPS is updated, it then checks if a JSON file exists. This is essential because the system needs a JSON file to trigger a blockchain transaction.
  It then sets BCTransEnabled to true once the file is found, allowing the blockchain transaction coroutine to proceed. That is, it will not attempt to start a transaction unless and until it finds out that the JSON has relevant data.

- **GPS Coroutine**: The purpose of the GPS Coroutine is to deal with the geographic location data gathering. In this regard, it activates the modem to acquire GPS information. It continuously checks when the GPS information is acquired and on that basis, the GPSUpdated flag is set to true.
  In case there is an inability to update GPS, the coroutine will fall back into delay and retry somewhat later.

- **BChainTransaction Coroutine**: The Blockchain Transaction Coroutine is responsible for the concrete processing of the blockchain transaction, including the delivery of metadata from scanning to the blockchain network, together with the GPS data. The coroutine can only turn active in such a way that, if the BCTransEnabled flag is set to true, the transaction can proceed only if the required preconditions are met.
  The system will then read the JSON file after being triggered through internal flash and will connect with the modem, initiating the blockchain transaction. It ensures that data is inscribed in the blockchain in a non-repudiable and non-perishable way, improving integrity and traceability for the information scanned.

Critical Conditions and States

The flowchart emphasizes a variety of critical factors that govern the flow of the system:

- **scanEnabled**: Controls whether the system performs scanning.
- **GPSUpdated**: Ensures that the collection of the GPS data happens before further steps like the blockchain transaction can be executed.
- **BCTransEnabled**: A flag indicating whether the system is ready to execute a blockchain transaction, true in case a valid JSON file exists or not.

Data Storage and Communication

Among these features is that scanned data in JSON format are stored in the internal flash memory. It works as a buffer storage that holds data until the system is well ready to forward them across the blockchain network. When the conditions set forth are met, it initiates a modem connection for the upload of the data, integral to a blockchain transaction.
This setup ensures that once information is documented in internal flash, it finds its way into a blockchain to securely maintain its integrity and be tamper-proof.

Error Handling and Delays

This system has several delay states that assist in the control of processes, which cannot move forward since their conditions are not met. For example, the system enters a delay state if scanning is disabled, other than incessant checking of the condition, which would lead to inefficiency. Delays are used also on the GPS and blockchain transaction coroutines such that the system can wait efficiently for the satisfaction of conditions such that the processor is not overloaded or resources wasted.

3.1.2 Sequence Diagram

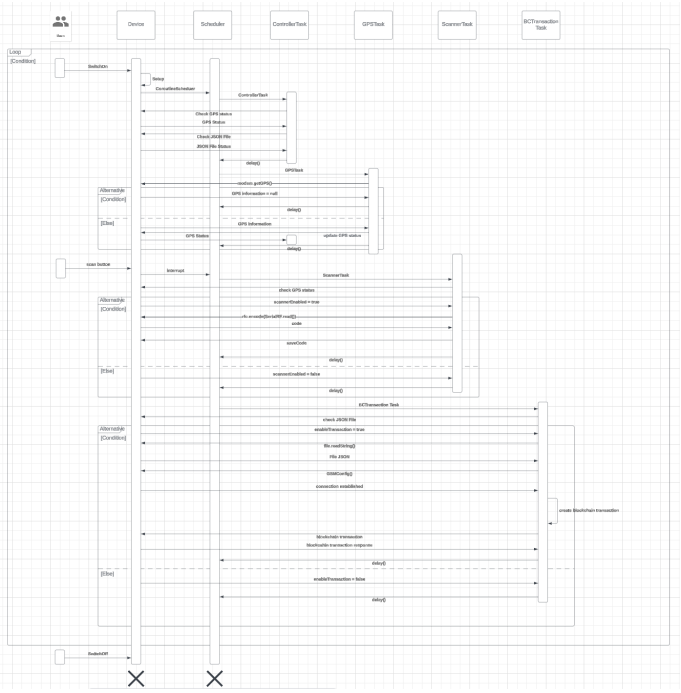


Figure 3.2: Sequence diagram

This UML sequence diagram shows the interaction of various details of the system. The intended functionalities will involve such operations as scanning, GPS data capture, and blockchain transactions. In this diagram, these elements expose their temporal interaction, which, upon analysis of sequences and flow of events, will detail activities like scanning, data capture, and blockchain documentation.

Components and Actors

Within the sequence diagram, there are several key elements:

- **Device**: It represents the whole hardware, or the embedded device.
- **Scheduler**: Ensures that different activities are scheduled and coordinated.
- **ControllerTask**: Handles the logic to check conditions and triggers actions in other tasks.
- **GPSTask**: Oversees the procedure of obtaining GPS data.
- **ScannerTask**: Provides basic methods for scanning operations.
- **BCTransactionTask**: Runs the blockchain transaction after processing the data.

Flow of Operations

1. **SwitchOn**: The system starts with the SwitchOn event, wherein the device is powered up and it starts its operation. Then the scheduler carries out some conditions and starts to execute tasks.
   First, the ControllerTask checks the state of the GPS system and its JSON file. Then, the scheduler instructs the GPSTask to progress provided that enabling conditions are met. In cases where GPS data is required, the ControllerTask adapts the execution of further steps based on this requirement. The GPSTask takes the responsibility of acquiring GPS data. It calls modem.getGPS(), which should return the location information. If GPS information are null, there is a bit of delay while returning to the scheduler. Otherwise, if the GPS data has been retrieved, the status is updated, including passing on this information to the ControllerTask. As the diagram shows, the GPS data is required for further processing, because other tasks rely on that data before they can proceed.

2. **Scan Button**: This event starts a check on the status of GPS, making sure that a scan can only occur when valid GPS data is available. When the GPS data is back, enable the ScannerTask, do a scan, and process the scan code. Encode the data that was scanned, and save it out as a JSON file. If the scann is not enabled, it enters a delay state in anticipation of the next activation.
   When the scan is complete and JSON has been generated, it is time for the BC-TransmissionTask to kick in. It first checks if the enableTransaction flag is true, which means that the blockchain transaction feature is enabled. The BCTransmissionTask also verifies if the JSON file exists and tries to connect to execute the blockchain transaction.
   With a connection established, it proceeds to create the blockchain transaction, making sure to capture the information that was scanned onto the blockchain in a correct and secure manner. If the enableTransaction flag is disabled, the task enters a delay state until such a time that the next transaction is allowed.

3. **SwitchOff**: The process terminates with the system powering down via the SwitchOff event.

3.1.3 Activity Diagram

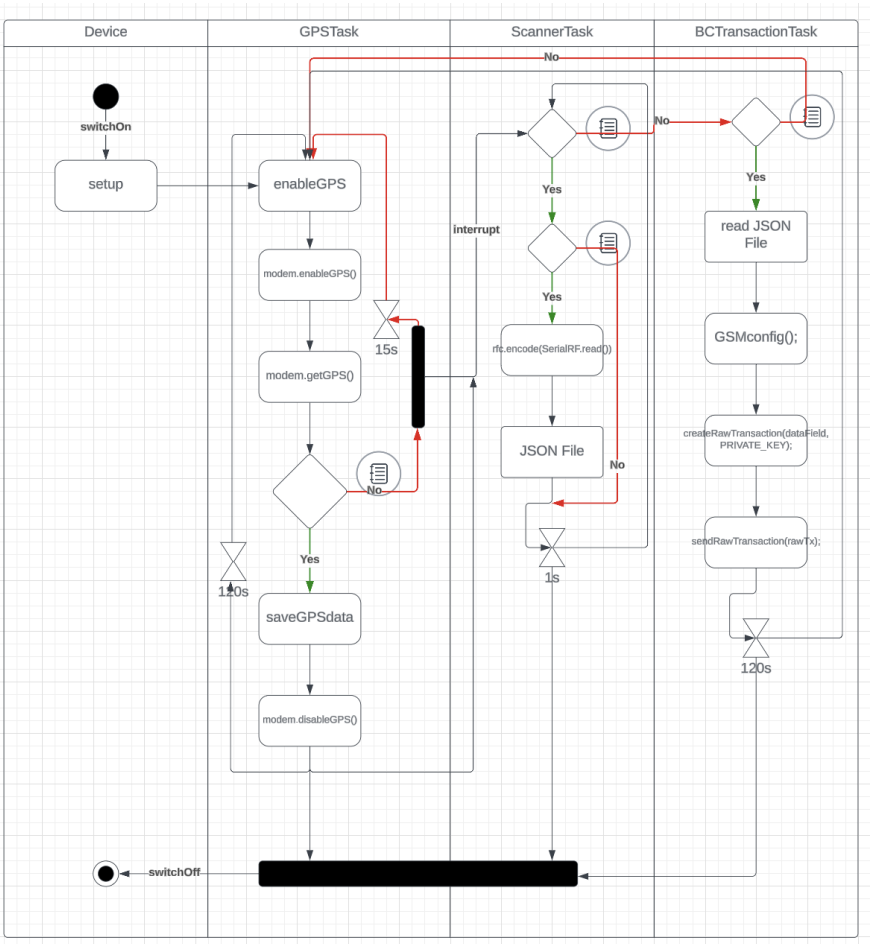


Figure 3.3: Activity diagram

This diagram, similar to the previous two shown, illustrates that very same embedded system, exposing the interaction between major tasks: Device, GPSTask, ScannerTask, and BCTransactionTask. It will illustrate, step by step, major system operations - from device power-on to blockchain recording. Let us proceed to explain this diagram and its implications in detail.

Major components and flow

- **Device Setup**: It starts with the event switchOn, which means switching on the device. Once it is activated, the setup process is launched.
  The setup process involves configuring necessary hardware and software components for proper operation of the embedded system.

- **GPSTask**: Once configured, the GPSTask is triggered, which activates the GPS module and records location data.
  The diagram shows that the modem GPS is turned on through modem.enableGPS(), whereafter the system waits for 15 seconds to elapse; this is to allow the GPS to lock onto satellites. GPS data is retrieved via modem.getGPS(); a check will

be performed to ensure valid GPS data was received. If valid GPS data is available it will be saved via saveGPSdata(), then the GPS will be disabled via modem.disableGPS().
Adding an 120-seconds delay after disabling GPS is an effort to save resources. GPS modules are generally power hungry. The system adds resiliency to the process by retrying when GPS data is not available.

- **ScannerTask**: The ScannerTask initiates by way of an interrupt; generated by an external event: when a scan is detected.
  Upon triggering the scanner, the program reads information from the serial port, encodes it by using the function rf.encode(SerialRF.read()), and it generates a JSON file including both Scanning and GPS data. There is a 1-second delay between completing the scan and generating the JSON file, because the system needs to process data and ensure smooth transitions between tasks.

- **BCTransactionTask**: It starts by confirming whether the JSON file exists. Once this has been affirmatively confirmed, the BCTransactionTask goes into the phase of reading.
  The procedure initializes GSM configurations (GSMconfig()) to facilitate a modem connection for the purpose of data transmission. After the modem has been configured, a blockchain transaction is generated through the createRawTransaction() using the JSON data and a private key. sendRawTransaction() attempts to send the transaction, followed by wait of 120 seconds - the confirmation that the blockchain transaction has been declared or the network is in sync.

- **Turn Off**: The whole process terminates when, after all the tasks are executed or the system reaches an idle state, the system switches off via the switchOff event.

3.1.4 BPMN Diagram

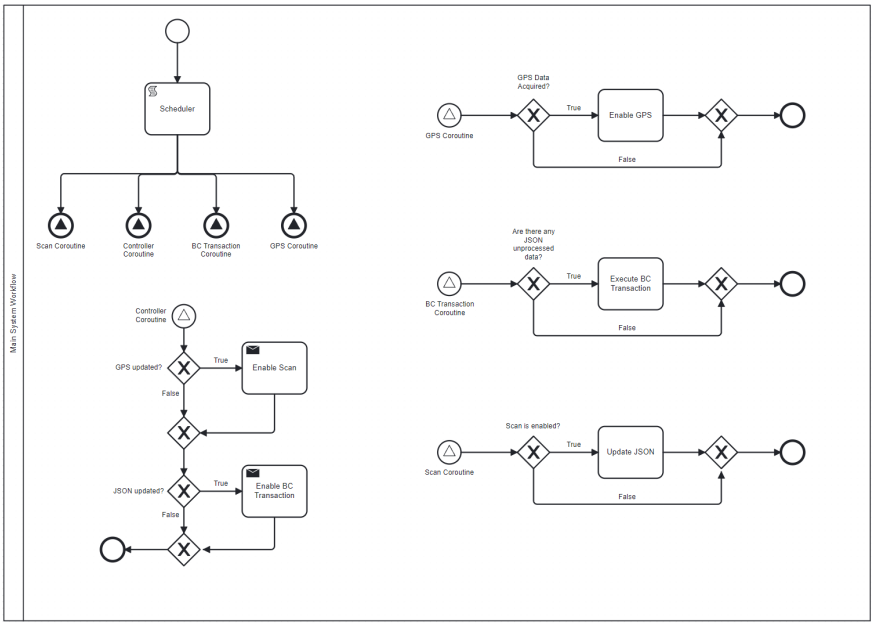


Figure 3.4: Workflow with BPMN Diagram

The image represents the dynamic view of the workflow of the embedded system that we used as a first step towards the implementation of the system based on a coroutine execution model. This type of diagram (BPMN - Business Process Model and Notation) is used to visualise the workflow of the system's key components and describe the behaviour and interaction of the various entities that make up the system.

Scheduler e Coroutine

At the top of the diagram, we can see that the system is managed by a central scheduler, represented by a box, which coordinates the activation of the different coroutines. In this case, four main coroutines are shown as subdivisions managed by the scheduler:

1. Scan Coroutine: Responsible for activating the scan of the product or object to certify its origin.

2. Controller Coroutine: Controls the flow of data and verifies that updates, such as GPS or transaction updates, are performed correctly.

3. BC Transaction Coroutine: Manages blockchain transactions, ensuring that traceability information is recorded correctly in the blockchain.

4. GPS Coroutine: Collects GPS information, necessary to determine the location of the object or product at a given time.

The scheduler then activates each coroutine when appropriate, ensuring that operations take place in a coordinated and conflict-free manner, respecting the priorities dictated by system conditions.

Workflow of Individual Coroutines (key aspects)

Each coroutine has a specific workflow that is displayed separately in the diagram.

GPS Coroutine: The workflow starts by checking whether GPS data have been acquired. If the condition is true, the GPS functionality is activated, otherwise the workflow ends. This step is an important phase of the system, as the correct acquisition of GPS data is crucial for the certification of product origin.

BC Transaction Coroutine: After activation of the coroutine, the system checks whether unprocessed JSON data exists. If such data is present, a blockchain transaction is executed, otherwise the flow stops. This process ensures that product data is tracked and recorded securely and unchangeably in the blockchain.

Controller Coroutine: This coroutine has a role in supervising and activating the other processes. In particular, it checks whether the GPS data is up-to-date; if the condition is met, it enables scanning, otherwise it checks whether the JSON has been updated to activate the blockchain transaction. This allows optimised workflow management, preventing unnecessary processes from being activated.

Scan Coroutine: The scan coroutine checks whether the scan is enabled; if yes, it updates the JSON file with the scan data, which can then be processed by the blockchain.

System Dynamics and Implementation

The presented model is particularly interesting because it emphasises the dynamism of the system and the parallel between the coroutines, which can be executed concurrently or conditionally depending on the events occurring. This design enables efficient resource management through the use of coroutines, which allow tasks to be executed in a non-blocking manner. The use of conditional events and logical checks such as those described in the diagram reduces the possibility of conflicts and optimises the utilisation of the system under high loads. We decided to represent the system in this way before proceeding to the implementation for two main reasons, the first is to simplify the whole thing, eliminating parts of the system and operations that did not interest us from an implementation point of view, the second is for the ability to represent the behaviour of the system even if we wanted to update the system to allow the coroutines to run in parallel. The embedeed system in question is in fact designed to be able to run coroutines from different hardware communicating via signals.

3.2 Simulators for Emebedded System Validation

A set of simulators may be applied for the purpose of comprehensive evaluation and assessment of the embedded system represented by the foregoing UML diagrams. Each of these simulators has different advantages with respect to flexibility, ease of use, and suitability for embedded system-related workflows. This chapter explores the capabilities of a number of simulators, including the opportunity of custom simulator implementation, together with the already established HIPS, Simulink, and PIPE. We will also compare Simulink and PIPE in the strengths and weaknesses these tools might have with respect to the specific requirements of the system to be analyzed.

3.2.1 Developing a Custom Simulator

The development of a proprietary simulator would allow customization of the analysis and evaluation of specific processes operating in an embedded system. This is because with an internally developed simulator, one could go ahead and represent dimensions of the system not well taken care of by commercially available simulators.

This methodology allows easily implementing several metrics from performance evaluation: real-time monitoring, resource utilization, and workload impact on system performance. One of the biggest disadvantages of this approach is that developing a simulator from scratch is a very time-consuming job with high complexity; it requires dual experience in simulation modeling and operation of embedded systems.

3.2.2 HIPS (Hardware In the Loop Simulation)

HIPS[Hip] is a powerful tool that's often used in the testing process of an embedded system. It bridges the gap between a simulated environment and real-world applications. This tool allows for actual test hardware to be introduced into a simulated environment, enabling real systems to be tested under controlled conditions.

It proves particularly useful for the process of testing those applications that require real-time interaction between the embedded device and its immediate physical environment, such as data input through sensors or GPS devices. On the other hand, HIPS setups are complex and can be quite resource-intensive to set up with both hardware and software components.

3.2.3 Simulink

Simulink [Simb] is a design, simulation, and analysis tool for dynamic systems that can be used in a wide variety of applications within the sphere of embedded systems and control theory. As a software package, it offers a graphical user interface for modelling processes using block diagrams, and it is particularly useful when trying to show or investigate complex workflows, as those present in the current system. Simulink supports real-time simulations and code generation, which makes it a very powerful tool for use during the system design validation phase before implementation. Moreover, since it is integrated with the MATLAB package, this provides full support for advanced mathematical analysis and scripting, making it even more adaptable to a wide range of simulations.

3.2.4 PIPE (Platform Independent Petri Net Editor)

PIPE [Pip] is a software application designed for the modeling and simulation of what are known as Petri Nets. These mechanisms are commonly employed to illustrate flow control within various systems, particularly when concurrent processes are occurring. This instrument proves effective in the analysis of such systems characterized by distributed tasks, such those used in ours: scanning, GPS data acquisition, and blockchain transactions. Petri Nets are particularly adept at identifying potential bottlenecks, deadlocks, or even synchronization challenges within the system. PIPE offers visual modeling and deep analysis, but it might lack some of the real-time simulation capabilities that may be available in tools like Simulink.

3.2.5 Comparison of Simulink and PIPE

Both of the platforms provide important insights into the system dynamics; however, their applications differ significantly. Simulink is especially efficient for simulations conducted in continuous time and for analyses performed in real time, rendering it highly appropriate for embedded systems that necessitate control logic and engagement with dynamic inputs, including sensors and external devices. On the contrary, PIPE specializes in discrete event systems, equipping users with robust tools for the exploration of concurrency, synchronization, and possible conflicts that may arise during process execution.
In our view, though PIPE is a good tool to analyze concurrency and task interaction, Simulink seems to be more adequate for our needs. It allows a comprehensive environment to test process performance and operational characteristics, very relevant for real-time applications such as GPS data acquisition and blockchain transaction execution. While PIPE can be useful in deadlock and resource conflicts avoidance, when it comes to the real-time performance guarantee and dimensional dynamic process analysis that our system requires, it is bounded. Therefore, Simulink will be more appropriate for this evaluation.

4. Implementation

This chapter will present the implementation of the simulation system developed, analysing in detail the different phases and design choices. It will start from the initial attempts to implement the simulation using PIPE and Petri nets, illustrating the problems encountered and the limitations that led us to consider other solutions. Next, the motivations that led to the choice of Simulink as the main simulation tool will be described, highlighting the advantages offered in terms of flexibility and integration with the system model. Finally, the implementation of the system within Simulink will be presented, showing how the main components were modelled and simulated to analyse the system's behaviour.

4.1 PIPE and Petri nets modeling

The idea of using Petri nets to model and simulate the system in question has deep roots in the engineering of complex systems. Petri nets, as discussed in the previous theoretical section, are powerful tools for modelling systems characterised by concurrency, synchronisation and non-determinism. This makes them particularly suitable for representing systems consisting of several processes (or coroutines). Furthermore, their ability to model stochastic behaviour through the use of probabilistic transitions initially made them a promising choice for simulating our embedded system.

Our idea of applying Petri nets came from the intuition that we could approximate the system architecture as an analogy of an industrial assembly line. Each coroutine in the system could be seen as a 'machine' in a factory, taking inputs (data or requests), processing them and passing them on to the next coroutine. This model seemed appropriate for evaluating performance and identifying potential bottlenecks, just as one would do with a physical production chain. Finding various articles [LW09][CL09][CL18][SS07] proposing the use of Petri nets to simulate assembly lines and analyse the efficiency of each machine, we assumed that such a structure was applicable to our context, allowing us to identify the breaking point of our system.

A further point in favour of using Petri nets is their non-deterministic nature, an inherent characteristic that reflects well on the behaviour of our embedded system. The system's coroutines are in fact subject to stochastic variables, which necessitated an approach that could take into account uncertainty and probability in the execution times and interactions between the coroutines. The PIPE (Platform Independent Petri net Editor) software is a well-known open source tool for simulating Petri nets, which includes support for stochastic nets, making it a natural choice for our purposes. PIPE allowed us to model Petri nets graphically and run simulations that take into account temporal and stochastic transitions, thus providing a solid basis for studying the be-

haviour of our system under conditions of uncertainty.

However, despite the initial enthusiasm for Petri nets and their potential, some significant difficulties emerged during the implementation process. One of the main problems encountered was the complexity of correctly representing the logic and execution flow of the coroutines in the system. While Petri nets are effective in modelling sequential and concurrent processes, the specificity of our system, in which coroutines have variable execution times and must respond dynamically to external events, highlighted the limitations of this approach. We will discuss later in this chapter, how managing precise time transitions and modelling the system in real time presented difficulties that PIPE was unable to resolve and other reasons that brought us to Simulink.

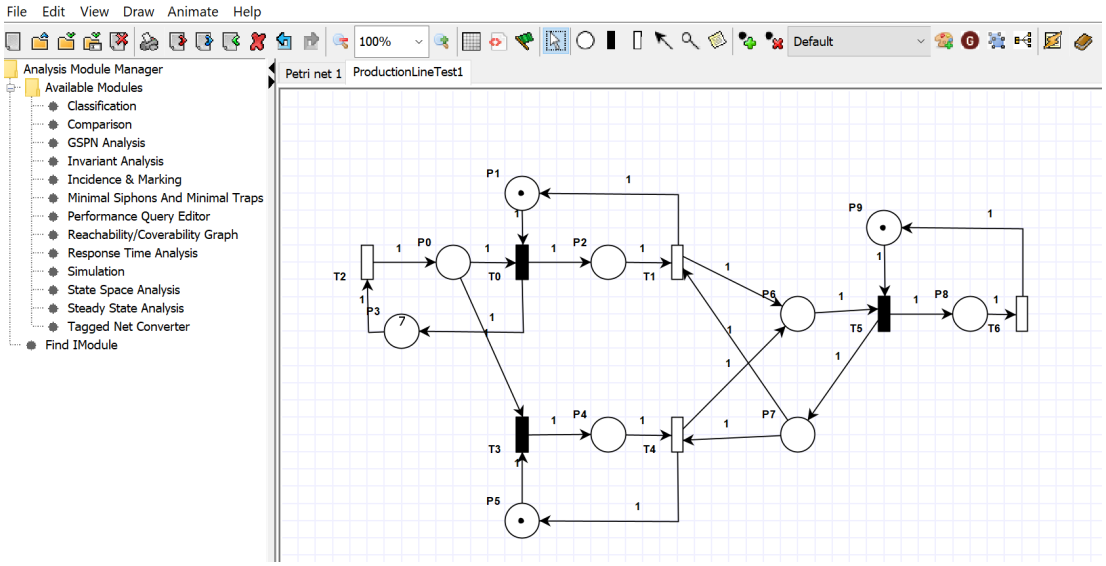


Figure 4.1: PIPE interface

The image above shows PIPE's intuitive graphic interface used to create and analyse complex systems. In the image provided, the interface shows a grid drawing environment in which it is possible to construct models of Petri nets using graphical elements such as places (circles), transitions (rectangles) and arcs connecting these elements. On the left-hand side of the screen, there is a menu listing the various analysis modules available, including simulation, state-space analysis, and tools for calculating structural properties such as failure loads. In the toolbar at the top, there are options for drawing and editing Petri nets, such as inserting transitions, places, arcs, and configuring network parameters. The software allows simulations of the created nets, verifying the dynamic behaviour of the system. In addition, detailed analysis of network performance and coverage can be performed.

The picture in the creation grid also shows an example of two processes, or coroutines, connected in series within a Petri net. Following the analogy used above, we can think of each coroutine as a machine in an assembly line. The first machine (or coroutine) processes a certain number of 'parts' and, once it has completed its processing cycle, passes the results on to the second machine. The second coroutine can then only start its process when it has the parts produced by the first. In other words, the execution flow of the second coroutine depends entirely on the output generated by the first. This chain structure shows a sequential dependency relationship between the

coroutines, which is typical of many complex embedded systems. The efficiency of the entire system, as well as the time required to complete a complete cycle of operations, depends on the correct operation and balance between the various coroutines. If one of the coroutines operates at a slower pace or takes too long to process its inputs, the entire system can suffer delays, slowing down overall production or, in the worst case, stopping completely.

In order to analyse the breaking load of a simple system like this, i.e. the point at which the system is no longer able to function properly due to overloading or slowing down, we need to take a close look at the components that make up the system. Using tools such as Petri nets and simulation software such as PIPE, it is possible to monitor the flow of tokens (representing the processed 'pieces') between the various transitions and identify any bottlenecks. By analysing the time it takes for each coroutine to complete its cycle and comparing it to the required production rate, we can predict when the system will overload and begin to fail, providing a key indicator for improving the efficiency of the system or redesigning it.

4.1.1 The system as a supply chain

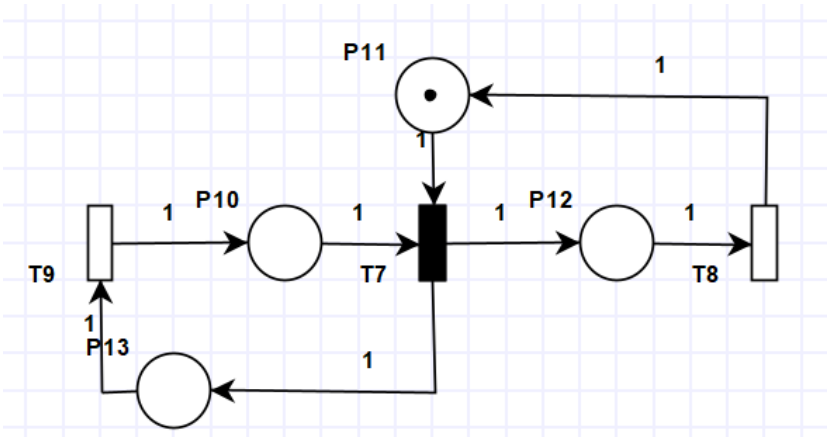


Figure 4.2: Single machine petri net

This Petri net represents a central component of the simulation, modelling the operation of a single machine capable of producing a part in time t. The production cycle begins with transition T9, which simulates the entry of new parts into the system, generating them every t units of time. The part then passes into place P10, which represents the initialisation of the production process.

The machine has a state called idle, represented by place P11, which contains a token when the machine is idle and ready to start production. The T7 transition can only be triggered when P11 is in the idle state, indicating that the machine is not currently occupied. When T7 is triggered, the production process begins and the idle state is momentarily suspended.

The part then proceeds to P12, where the actual processing takes place. The final transition, T8, represents the output of the machine: after a processing time t1, the workpiece is produced and can be 'picked up' or used for further downstream process-

ing. The cycle continues as long as the machine has parts to process, thus simulating the behaviour of a real production chain.

This network can effectively model the operation of a single machine in a production context, with the aim of analysing the processing time and operational efficiency of the system, however, as we continued with the development on PIPE, we soon realised that the complexity of the system we intended to recreate, far exceeded our initial expectations. Although PIPE proved to be a simple and intuitive tool for representing one or two machines, it became extremely complex when we tried to model several coroutines running simultaneously. Furthermore, the stochastic aspect of PIPE, while useful for handling stochastic processes, is not controllable in any way. This posed a problem for the parts of the system where we needed to guarantee determinism, as it was not possible to switch off this randomness where it was not needed.

4.1.2 PIPE disadvantages

Although PIPE offers a formal and useful approach to modelling systems using Petri nets, after a thorough analysis we concluded that Simulink is superior in terms of practical application for the simulation of our embedded system. One of the main reasons for this is accuracy. Simulink's continuous-time simulation and its ability to handle complex mathematical models allow the dynamic behaviour of the system to be represented more accurately than PIPE's discrete-event approach. This is particularly relevant when dealing with real-time systems, where it is crucial to accurately capture the timing and interaction between the various components.

In addition, Simulink offers a more intuitive and user-friendly interface for creating simulations, which is particularly suitable for users who may not have extensive experience with formal modelling languages such as Petri nets. Simulink's block diagram approach, in which each system component is visually represented as a block and connections represent the flow of data or control signals, simplifies the construction and understanding of complex systems. In contrast, PIPE requires an in-depth understanding of Petri net theory and its specific elements (places, transitions and tokens), introducing a steeper learning curve.

Another additional advantage of Simulink over PIPE is the presence of a large and active user community, as well as extensive documentation and technical support provided by MathWorks witch turned out essential to this project. This allowed us to easily find tutorials, troubleshooting guides and ready-made models to build on, speeding up the development process and facilitating problem solving. On the other hand, although PIPE is an open-source tool, its community and support base are more limited, making it more difficult to find useful resources for complex simulations or handling advanced problems.

In light of these factors, Simulink not only offers a more detailed and accurate representation of the system, but also provides a more fluid, scalable and supported platform for real-world simulation and validation of embedded systems. In the next section we will discuss how we continued the implementation of the model in Simulink in details.

4.2 Simulink implementation

Before analysing the implementation process, it is important to discuss some technical aspects of the model. As mentioned in the analysis chapter of this report, in representing the system we have to deal with some deterministic parts, some non-deterministic parts, and others that are probabilistic or stochastic in nature. Our simulation must take into account all these variables and represent each aspect appropriately.

First of all, we decided not to limit ourselves to representing the system only from a logical point of view, but to create a simulation that also considers the architecture of the embedded system. Simulink greatly facilitated us in this, as it allows us to break down the various parts of the system using separate blocks[Sima]. We felt that the best approach was to encapsulate each coroutine as a block in the simulation. These coroutines are then handled by a scheduler block, which invokes the coroutines and receives a completion signal from each. Once the coroutine has signalled the completion of its execution, the scheduler starts the next one.

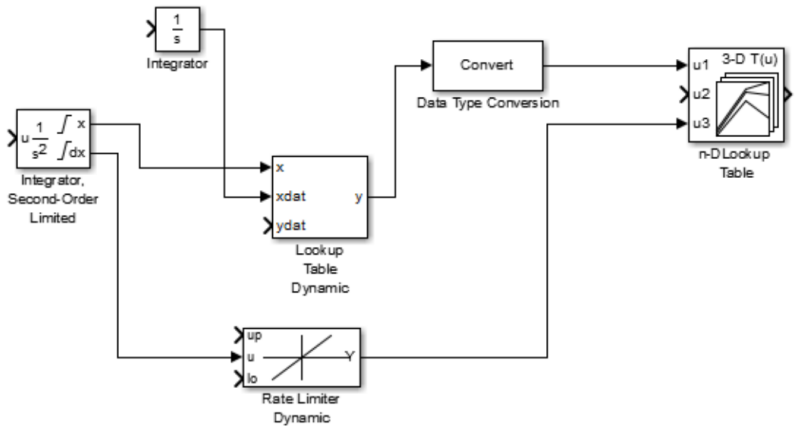


Figure 4.3: Example of Simulink block diagram approach

Of the many blocks offered by Simulink, the one we have mainly used for our system is the Matlab Function Block [Mat], a block that allows Matlab code to be inserted directly into the Simulink model. This block is extremely versatile, as it allows custom functions to be written to handle more complex operations that cannot be easily represented using standard Simulink blocks. In practice, the Matlab Function Block allows the capabilities of the model to be extended by incorporating the power and flexibility of the Matlab language, especially when dealing with complex control logic or advanced mathematical calculations.

Arguably, the same behaviour of the system we have implemented could be replicated using other specific Simulink blocks, such as those dedicated to flow control or handling conditional logic. However, the use of the Matlab Function Block allowed us greater flexibility in writing code and handling coroutines. Thanks to this block, we were able to implement the necessary logics more quickly and efficiently, without having to rely completely on Simulink's default blocks, which can sometimes be more complex or less intuitive to configure for certain types of operations. Furthermore, the use of the Matlab Function Block allowed us to easily modify the behaviour of the system, testing

different solutions without having to completely restructure the model.

Figure 4.4: Matlab function block

Thanks to this block, we were able to write specific code that matches the system requirements. As said before, we intended to reproduce the exact architecture of the system and create a simulation of the device behaviour based on coroutines. The UML described bfore served as a blueprint and as a reference to represent the architecture of the system, thanks to the UML diagrams we implemented the scheduler and the 4 coroutines (Scan coroutine, Controller coroutuine, GPS Coroutine, Blockchain Transaction coroutine) with Matlab functions.

4.2.1 Scheduler and initialization

In this part of the system, we use a combination of MATLAB blocks in Simulink to manage the execution sequence of coroutines, a key concept in the simulation of our embedded system. The image below represents the Simulink model interface, which includes the blocks Initializator, schedulerControl, and scheduler, each of which plays a specific role in managing the time flow and control of coroutines. Below we analyse the behaviour of each block.

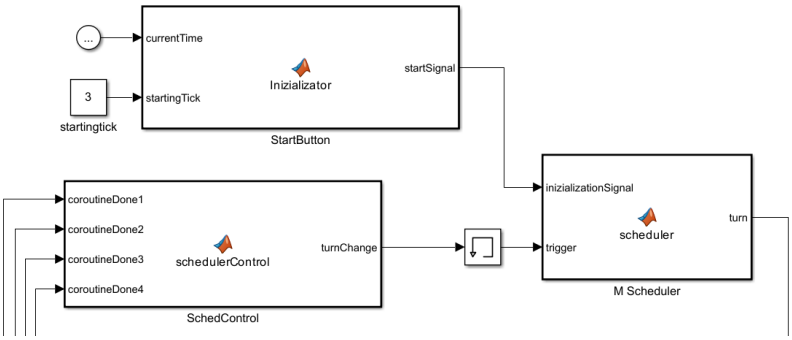


Figure 4.5: Scheduler and initializator block àù

Initializator

The Initialiser block is responsible for initialising the system. The Initializator function, which takes the current time (currentTime) and the starting tick (startingTick) as input, checks whether the simulation time has reached the value defined for the start of execution. If the currentTime is equal to the startingTick, the function generates a start signal by setting the variable startSignal to 1, otherwise it remains at 0. This allows precise control over when to start the simulation, synchronising all other blocks with respect to this start signal.

schedulerControl

The schedulerControl block handles the switching from one coroutine to another. This function monitors the status of the four coroutines (indicated by the signals coroutineDone1, coroutineDone2, coroutineDone3, coroutineDone4) and detects when a coroutine has finished execution, using an edge detection mechanism to detect when one of the 'coroutine completed' signals goes from 0 to 1.

The edge detection logic checks whether a coroutine has gone from the 'not completed' (0) to 'completed' (1) state. When this happens for any of the coroutines, the function generates a change signal (turnChange) that indicates to the scheduler block that the control should switch to the next coroutine. Once the signal has been generated, turnInProgress is reset to ensure that the change occurs only once per tick.

scheduler

The scheduler block controls which coroutine is to be executed. Each time a turnChange signal is received, the block updates the current index (currentIndex), which indicates which coroutine is to be executed. The process is cyclic, which means that once the last coroutine is completed, the system returns to activate the first one.

This block uses the initialisation signal (initialisationSignal) to reset the cycle to the beginning and the trigger signal to advance from one coroutine to the next. At each tick where a shift change is detected (rising edge of the trigger signal), the index is incremented cyclically, maintaining control over the running coroutines. The image below shows the signal produced by the scheduler block during the simulation.

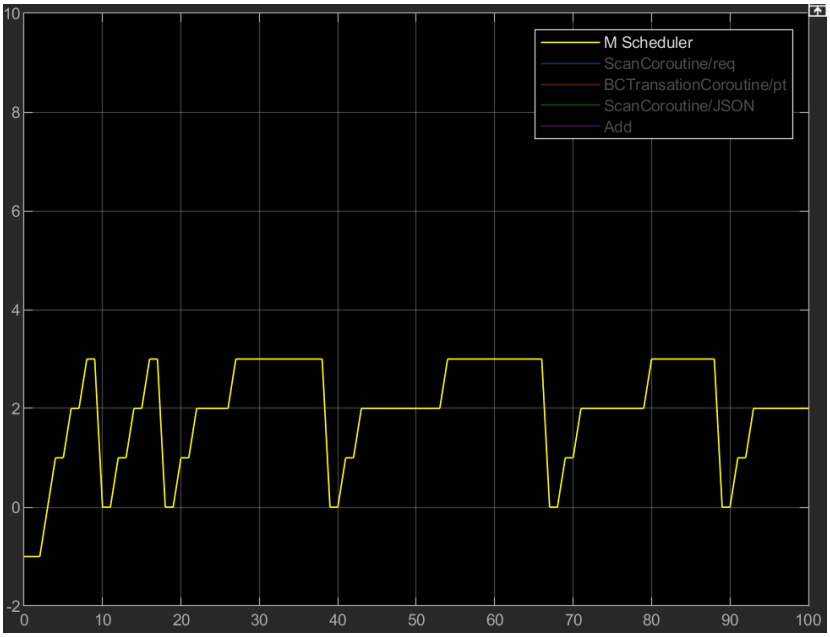


Figure 4.6: Scheduler signal scope

4.2.2 GPS Coroutine

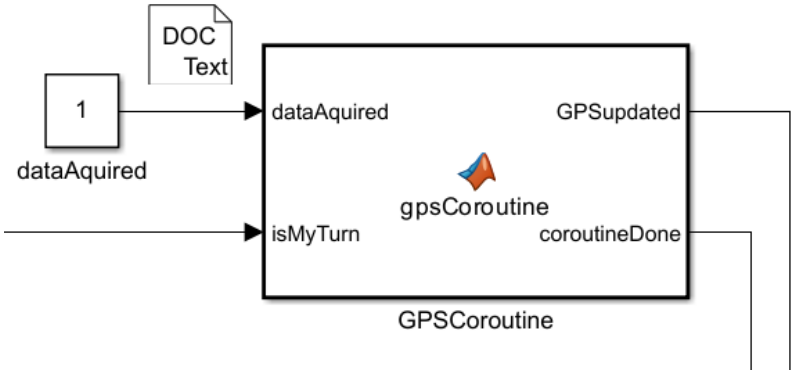


Figure 4.7: GPS coroutine block

The gpsCoroutine function simulates the GPS update process within our system. This process is controlled by the scheduler that signal when the coroutine has to perform its task. The main objective of this coroutine is to monitor the acquisition of new GPS data and to signal when the update has been completed. The coroutine uses a persistent variable called state, which maintains its value between different executions of the function. Initially, this variable is set to 0, indicating that no GPS update has been performed yet. Like in other coroutines, the use of a persistent variable allows coroutine to 'remember' its state even after its execution has been suspended by the scheduler, resuming the process exactly where it left off. This state is than the output of the function as a binary signal (GPSupdated), if this signal is equal to 0, it means the GPS is not updated.

The coroutine is designed to execute only when it is its turn, determined by the isMyTurn input provided by the scheduler signal. If it is its turn, the function checks whether new GPS data has been acquired via the dataAquired parameter. If data is available (dataAquired == 1), the status of the coroutine is updated to 1, indicating that the GPS update is complete. In this case, the coroutine signals that it has completed its task by setting the coroutineDone value to 1. Finally, the function signals that it has not yet completed its execution by setting coroutineDone to 0 until the task it needs to perform is completed. This wait cycle is necessary to synchronise execution with the global scheduler, and it is used in every other coroutine.

Here below we present the image of the scope of the output of the GPScoroutine, it is clear how, when it is the coroutine's turn (indicated by the scheduler signal in the scope in yellow), the output of the GPSupdated is also set to 1, and because the DataAquired is always true (1) in the simulation, when this coroutine is called again, it does not change the value of the output.

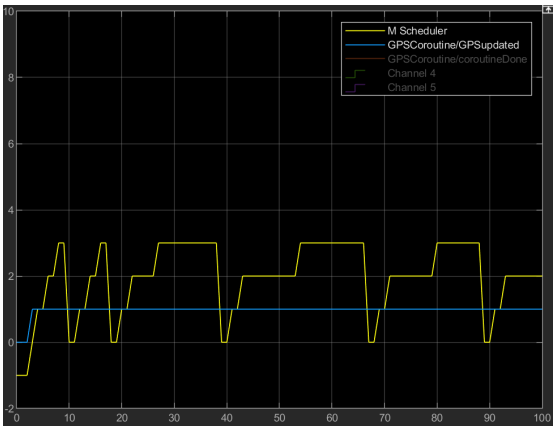


Figure 4.8: GPS updated signal (in blue)

We would like to also show the signal produced by coroutineDone, as it is a mechanism used by every coroutine in the simulation, the image below shows that whenever a coroutine ends his tasks, it produces a binary signal ture (1) that is than detected by the scheduler and used to start another coroutine.

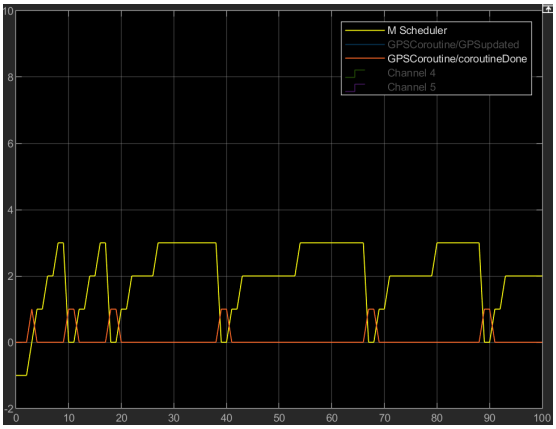


Figure 4.9: Coroutine Done signal (in red)

A final aspect to consider is the DataAquired input. In the simulations we ran, DataAquired is always set to 1, which indicates that the GPS is always able to acquire the data and supply it to the rest of the system. This choice was made because if the GPS were unable to obtain the data, the entire system could not perform its function properly. It is possible that, in some cases, the GPS cannot communicate due to the poor coverage of the area in which the system is located. However, although the data acquired by the GPS can be considered probabilistic (in 95% of cases it is equal to 1), we believe that introducing this variability, although feasible, would add an unnecessary level of complexity to our simulation.

In other words, we decided to keep DataAquired fixed in order to simplify the simulation, as our focus is not on handling any GPS failures. Adding a probabilistic component to simulate the failure of GPS to acquire data could be interesting in contexts where this variable plays a key role in the functionality of the system. However, in our specific case, this choice would have unnecessarily complicated the model without providing any real added value to the final simulation results. If an user of our

simulation might be intrested in this behaviour, it can easily change the variable we set to 1, with a function that produces the DataAquired signal as it pleases.

4.2.3 Scan Coroutine

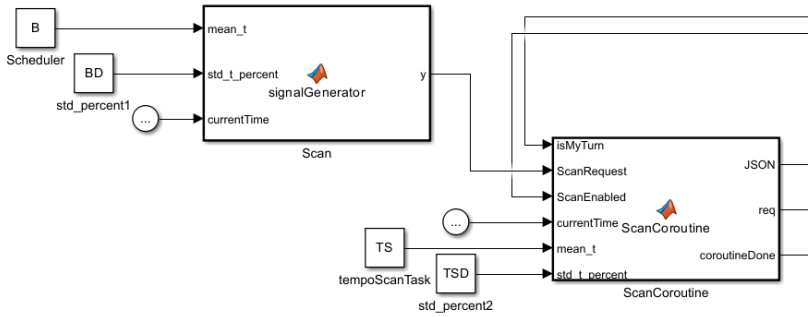


Figure 4.10: Enter Caption

Integration of Stochastic Events into the System

In a system such as the one we have designed, as explained before, some events occur deterministically, while others may be influenced by random or stochastic factors. Stochastic events represent behaviour that cannot be predicted exactly but only modelled through probabilistic distributions. This aspect was first integrated into our system with the introduction of the signalGenerator and the ScanCoroutine, which together simulate the random arrival of scans in the system and their subsequent processing.

signalGenerator

The block of code relating to the signalGenerator has the task of simulating the stochastic behaviour of the scan in the system's scanner. In other words, it represents the moment when a user scans a QR code in the system, with an average time interval between scans determined by the variable mean_t and a variability around this value expressed as a percentage by the variable std_t_percent.

Each time the system is interrogated, it is calculated whether or not to generate a scan signal based on the current time and the next scheduled event. The next event is determined by the addition of a stochastic interval, which is influenced by a mean_t centred normal distribution with proportionate standard deviation (std_t_percent). This makes the generation of the signal not precisely predictable, but statistically adjusted. If the current time reaches or exceeds the nextTriggerTime, a signal is emitted and the next event is scheduled. Otherwise, the signal remains null.

We can observe from the image that the input for this block are two variables called B and BD, this is because, to test the system and find eventual bottlenecks we run the simulation multiple times and each time we vary some aspects, we will shol later how we can set this variables before launching a simulation via Matlab, for now we would

like to point out the effect that these variable can have on the output of the signal generation, here below the 2 images show how we can render the output of this block less predictable by adjusting these variables.

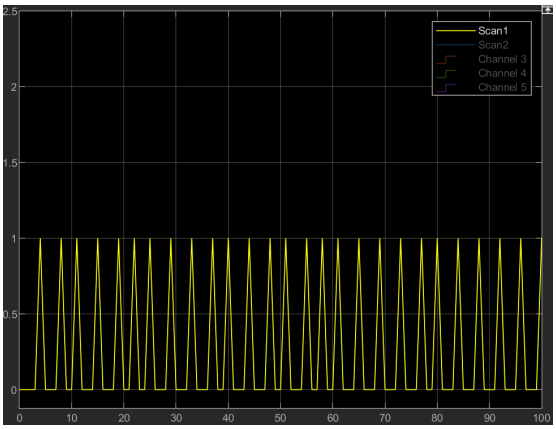


Figure 4.11: Scan scope B = 3, BD = 5

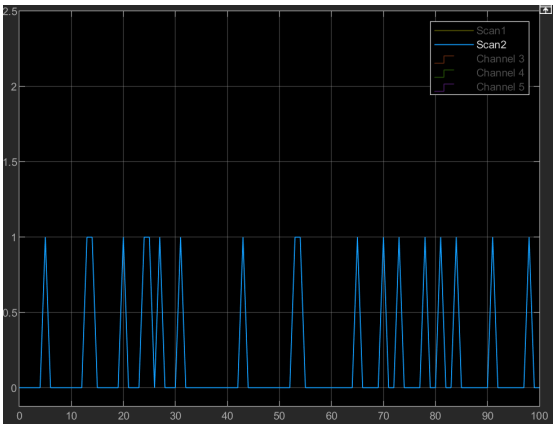


Figure 4.12: Scan scope B = 3, BD = 80

ScanCoroutine

The ScanCoroutine is responsible for processing scans received by the system. When a new scan request is generated (indicated by the ScanRequest variable discussed before), it is queued, even if the coroutine is not currently active. The system stores the number of pending requests and, when it is the coroutine's turn to process a scan (indicated by isMyTurn), it proceeds to process a single request at a time.

Again, the time required to process a scan is not fixed, but follows a stochastic behaviour. At the start of processing, a random time interval is calculated based on the mean mean_t and the standard deviation std_t_percent, as in the signalGenerator. During this time, the coroutine simulates the scan processing. When processing is complete, the coroutine updates the internal state, indicating that a scan has been processed and signalling to the system that the coroutine has completed its task for that cycle.Again the mean_t and std_t_percent are set as variables assigned each run of the simulation through Matlab. Additionally, if there are no pending requests at the

time of the coroutine's turn, the system returns immediately without performing any further operations, signalling that the coroutine is finished but without processing new scans.

These two code blocks represent an implementation of stochastic events in the system, simulating the variability of the arrival and processing times of scans. By using a probabilistic distribution, in this way it is possible to realistically model the behaviour of the system in a non-deterministic context, while still maintaining statistical control over the events. The signalGenerator manages the random generation of scans, while the ScanCoroutine takes care of processing them according to system availability and load. These elements offer greater realism to the simulation, introducing dynamics that are more in keeping with real situations in which events can occur randomly.

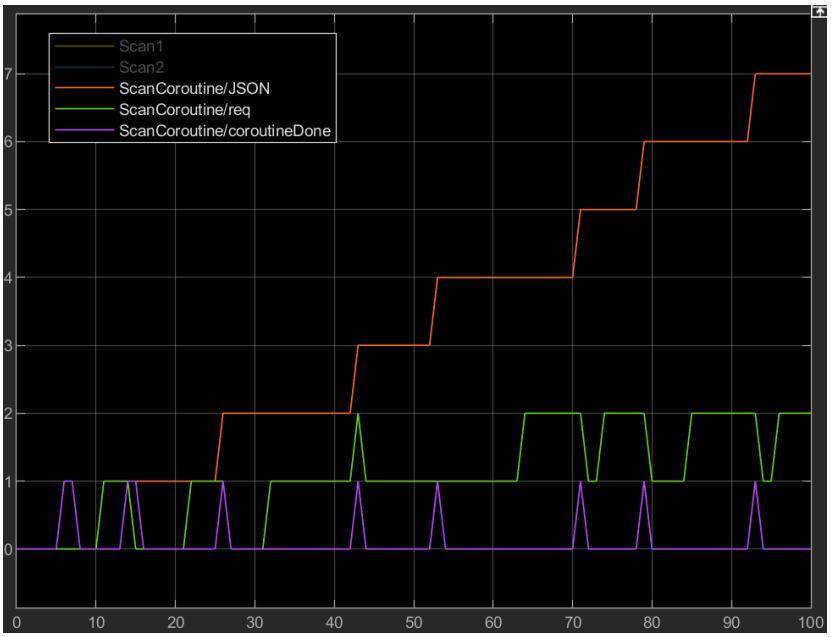


Figure 4.13: Scope of the scan coroutine output

In the image we can observe 3 signal, in purple the coroutineDone signal that indicates each time the coroutine is completed; in green the requests queue, indicating how many scans have been performed that needs to be elaborated into a JSON; finally in red, the JSON signal, this signal is an integer that represents the number of scans information that have been added to the JSON file that will than be used by the Blockchain Transaction Coroutine to perform a blockchain transaction based on this information. Some key aspects to observe are the fact that we can see, each time the coroutine ends (signaled by a purple spike in the scope), a JSON is created, and the queue diminishes by 1.

4.2.4 Controller Coroutine

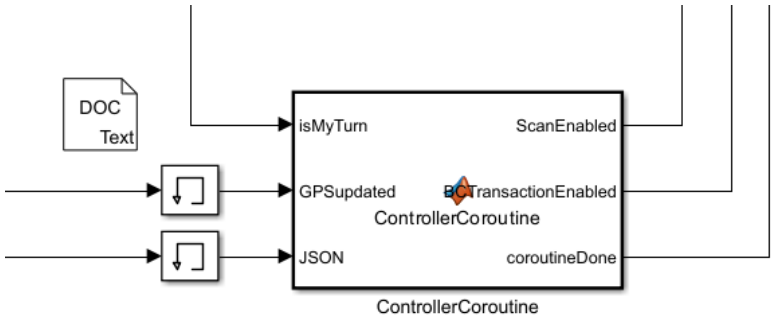


Figure 4.14: Controller Coroutine function block

This coroutine is one of the simplest in the system, yet it could be considered the core of the whole embeded system. The role of this coroutine is to eneble binary flags that tells the other coroutine is the scan is enebled and if the blockchain transations is anabled and can be performed, these flags also ensure that, in case the other coroutine are called without the flags being positive, they end immediately without wasting time.

In detail this function checks if the GPS has been upodated by the GPS Coroutine, if so, it enables the scanEnabled flag, than it cheks if there are any JSON that have not been processed already, if so it enables the Blockchain transaction. One last thing to take into account, before the inputs of GPSupdated and JSON we can observe from the image that there are 2 memory blocks, this is because in simulink the input of a block in one tick cannot depend from the output of the same block in the same tick, so because our system creates a loop between this coroutine, the scheduler (through coroutineDone output) and the other coroutines, the memory block is necessary so that the input used to produces the output is referring to the tick before and not the same one of the output.

4.2.5 Blockchain Transaction Coroutine

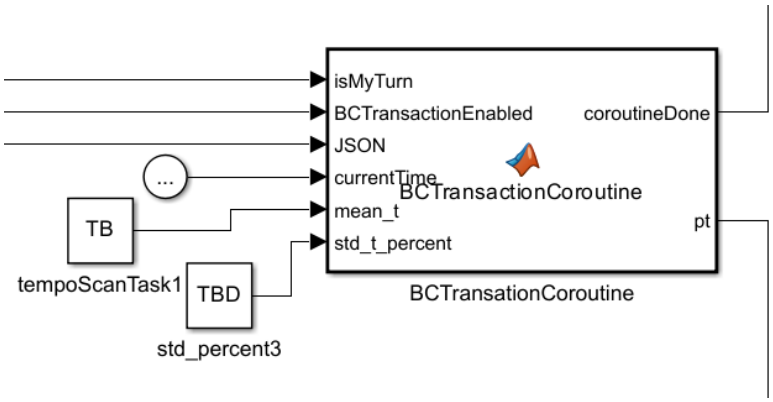


Figure 4.15: Blockchain Transaction Coroutine function block

The BCTransactionCoroutine plays a central role in transaction processing within a system simulating a blockchain-type process. The main function of this coroutine is to manage and process stochastically (with random variability) the transactions received, represented by the JSON parameter. In practice, it processes one transaction at a time, respecting the time constraints dictated by the average duration and percentage variability provided as input. The coroutine is designed to trigger only when it is its turn and if there are new transactions to process (JSON > processedTransactions). Processing follows a cycle that simulates the time required to complete each transaction, generated stochastically using the mean_t parameter (average processing time) and a standard deviation (std_t_percent).

This coroutine is in some ways, very similar to the ScanCoroutine, as the main input is the JSON produced by the Scan, we can say that this two coroutines are 'in series', and the blockchain transaction will fire only after the Scan has produced a JSON. The output of this coroutine is a variable called pt (processed transaction), in the scope of our simulation this variable is used to compute how many transaction are still needed to be performed by simply subtracting JSON - pt we get the number of unprocessed JSON, this will then be used to find whether or not, the breaking point of the system if this coroutine (Transaction too slow) or the Scan coroutine (JSON creation too slow). This specific aspects will be analized in depth in the next chapter.

5. Tool Execution and Results Analysis

In this chapter, the results obtained from the simulation will be analysed and an overview of the final product will be provided, together with instructions for running the developed tool. The following figure shows a complete block diagram view of the simulation realised in Simulink. Please note that this simulation was designed to be architecturally faithful to the reference embedded system, which is based on coroutines. In the remainder of the chapter, we will show how, through the execution of the simulation, it is possible to obtain a detailed analysis of the system's performance, which makes it possible to determine, thanks to the specifications provided, what hardware is needed to meet the system's operational requirements.

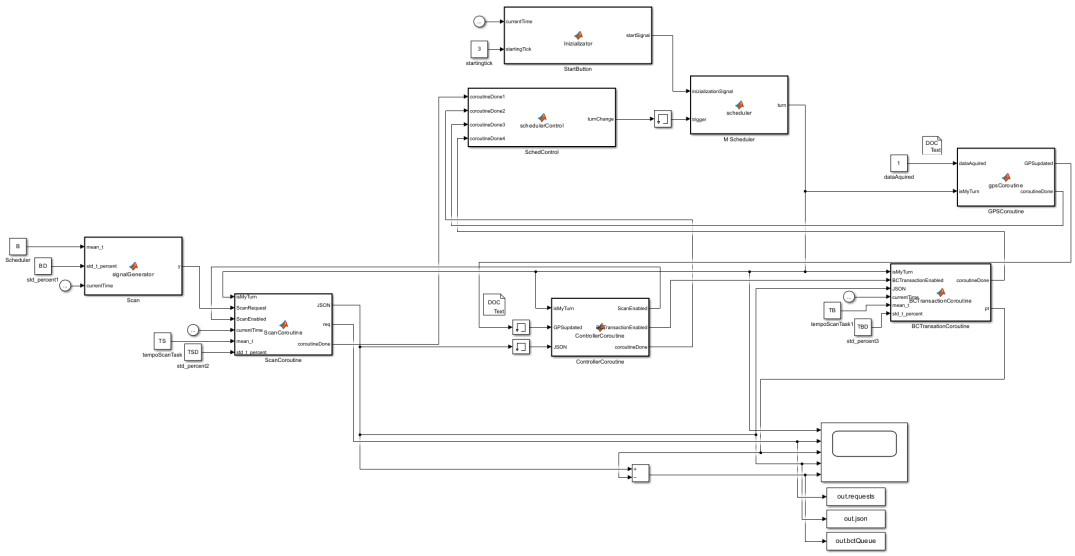


Figure 5.1: Simulink complete view of the simulation

5.1 Simulation execution

First of all, discerning readers will have noticed the presence of a few blocks in the simulation that have not been described above, namely the four blocks located in the bottom right-hand corner of the image. The largest block is the scope, a tool used extensively during the development of the simulation. The scope allows the values of input signals to be displayed, making it an essential component for debugging and monitoring the system. Even more relevant, however, are the three underlying blocks, as they represent the heart of the analysis process for determining the system's failure load.

The simulation, as described and implemented so far, can be executed directly from Simulink by simply pressing the 'play' button and setting an end time. Simulink's intuitive graphic interface makes it very easy to start the simulation, especially if you already know the exact specifications of the system and have a large collection of reliable data. Although simulation can be used in this way, its real potential lies in the fact that it can be run before the hardware is manufactured. This is particularly useful in the industrial environment, where an embedded system can cost thousands of euros. By simulating the system in advance, it is possible to understand precisely what specifications are needed, avoiding over-dimensioning or unnecessary purchases of hardware components.

A question may arise at this point: how can we define the simulation parameters if we do not yet have the hardware to test them? The answer lies in adopting a probabilistic approach. We can establish plausible ranges for the parameters and then run the simulation considering all combinations of these ranges. This process allows us to obtain an accurate mapping of the conditions that lead to the system's failure load. Once these results are obtained, it will be possible to purchase hardware that exactly matches the performance required by the simulation, thereby optimising the investment.

In our specific case, the simulation is based on three main variables: B (the frequency of scans), TS (the average scan coroutine time), and TB (the average transaction coroutine time of the blockchain). As mentioned, the number of simulations to be performed will be equal to the product nB $\times$ nTS $\times$ nTB, where nB is the number of possible values of B, nTS the number of possible values of TS, and nTB that of TB. Clearly, it is unthinkable to manually configure all the simulation parameters each time, making an automated system to explore all possible combinations essential.

5.1.1 Matlab script

As mentioned above, Simulink can be controlled directly from MATLAB, allowing the simulation file (.slnx) to be executed via MATLAB commands. During simulation execution, the constant blocks within the Simulink diagram, which do not have fixed values but use labels such as B, TS and TB, do not contain predefined numerical values. Instead, these blocks acquire their values from the corresponding variables defined in the MATLAB workspace [Simc]. This means that each time the simulation is started, the values for the scan frequency (B), the average scan coroutine execution time (TS) and the average blockchain transaction coroutine execution time (TB) are taken directly from the workspace variables. This approach is extremely useful because it allows simulation parameters to be changed dynamically by simply updating the variables in the workspace, without having to intervene directly in the Simulink model.

To solve the problem presented in the previous paragraphs, it is possible to automate the process using a MATLAB script. The three simulation blocks, mentioned in the previous paragraphs, are known as the toWorkspace blocks. These blocks play a crucial role: they link the signals generated within the simulation and automatically export them to MATLAB's workspace once the simulation is finished. This flow of data between Simulink and MATLAB is made possible by the fact that Simulink operates within MATLAB, allowing for close integration between the two environments. In practice, the toWorkspace blocks capture simulation results in real time and transfer them to the workspace, where they can be analysed and used for further calculations.

This configuration allows a MATLAB script to be written that runs the simulation iteratively, varying the parameters B, TS and TB through all possible values within defined ranges. The script can run the simulation multiple times, exploring all combinations of the specified parameters. Thanks to the toWorkspace blocks, the data resulting from each execution is saved and stored in the workspace. Subsequently, this data can be analysed to determine how different configurations affect system performance, thus enabling the identification of the hardware specifications needed to meet the required performance. In this way, automation not only saves time, but also provides a comprehensive analysis of possible system configurations.

```
% data structure for each combination of TS and TB
dataMatrix = cell(maxLength, 1 + 2 * length(
   scannerTimeInterval) * length(bcTimeInterval));

% loop to acquire the value for each combination of parameters
for i = 1 : length(scannerTimeInterval)
    for j = 1 : length(bcTimeInterval)
        % assign simulation parameters
        TS = scannerTimeInterval(i);
        TB = bcTimeInterval(j);

        % run the simulation for this set of parameters
        sim('provaUML');

        % extract data from simulation
        timeData = ans.requests.Time;
        requestsData = ans.requests.Data;
        bctQueueData = ans.bctQueue.Data;

        % store data
        for tIdx = 1:length(timeData)
            dataMatrix{tIdx, 1} = timeData(tIdx);  % Store
               time data only once
            dataMatrix{tIdx, 2 * (i-1) * length(bcTimeInterval
               ) + 2*j} = requestsData(tIdx);
            dataMatrix{tIdx, 2 * (i-1) * length(bcTimeInterval
               ) + 2*j + 1} = bctQueueData(tIdx);
        end

        maxLength = max(maxLength, length(timeData));
    end
end
```

Here we provided an example of the code we used to run simulations iterating over various combinations of time parameters, defined by two intervals: one for the average time of scan requests (TS, represented by scannerTimeInterval) and one for the average time of blockchain transactions (TB, represented by bcTimeInterval). For each combination of these parameters, a simulation of the system is performed by invoking the Simulink provaUML model. During each simulation, time data (timeData), requests (requestsData), and the blockchain transaction queue (bctQueueData) are collected.

This data is stored in the dataMatrix, where each row represents an instant of time and the columns contain the corresponding data for each combination of TS and TB parameters. The matrix is sized to handle all possible combinations, and for each simulation cycle the maximum length (maxLength) is updated to ensure that the matrix can hold all the data collected. In summary, the code automates the collection of data from multiple simulations, simplifying the analysis of system performance for different configurations.

5.2 Results analysis

In the following section, we will proceed with an in-depth analysis of the data collected during the simulation, highlighting the main results and trends that emerged. To facilitate understanding, we will present a concrete example of the data obtained, which will serve as a practical illustration of the dynamics at play. By analysing this data, we aim not only to clarify the results, but also to provide a context for their use. This example will serve to demonstrate how the information derived from the simulation can be interpreted and employed in real-world scenarios, offering insights for further study and application. Understanding these results is crucial to assessing the effectiveness of simulation and its potential in representing complex behaviour of real systems.

Simulation Data Considerations: It is crucial to emphasise that the data used for our simulation is essentially indicative and could represent that of a real execution in certain contexts. However, the simulation design is intended to be a highly customisable tool, allowing the user to adapt the parameters to the specific needs of the system they wish to represent. This means that instead of relying solely on pre-defined data, users can enter more relevant values and conditions, making the simulation a flexible and versatile tool. Furthermore, the unit of measurement adopted in Simulink for simulation is the 'simulation tick'. Although this unit does not correspond to a precise time measurement, it can be interpreted in various ways depending on the needs of the model. Therefore, the parameters and variables used also follow this same unit of measurement, making the association of real times with simulation results complex. This characteristic implies that users must be particularly careful when interpreting the results, as the unit of measurement may influence the perception of the timing and dynamics of the simulated system.

Regardless of this facts, we will now present some ways we offered to the user, using our mock data as an example, to see the simulation results and make predictions on the system, based on the data. Some quick note for the users that would like to change the view on the data we provided, it is always possible to access the whole simulation data through the Matlab script we provided, making it extremely easy to create a new view on the data gathered.

5.2.1 Analysis through simulink

The first and easyest way to see the data of a simulation is through Simulink directly, expecially through the scope provided in the simulation, or by configuring a new one in the block diagram interface.

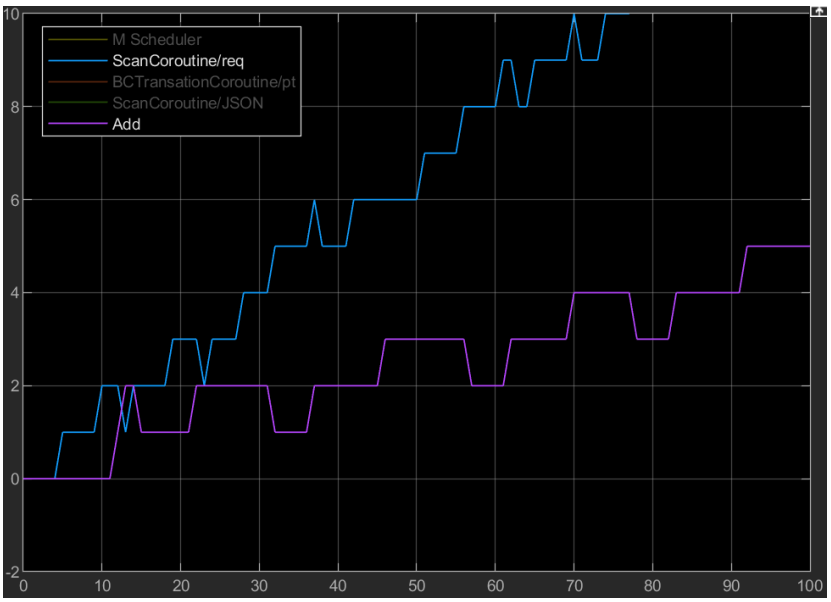


Figure 5.2: Execution results 1

Here above, we provided an example output of a simulation performed for 100 ticks, using the values B = 4, TS = 7 and TB = 8, with a standard deviation for TBD and TSD equal to 10%. From the analysis of the diagram, we can see that the system fails to properly handle scan requests, which accumulate in both the scan coroutine and the blockchain transaction coroutine. Examining the diagram more closely, we notice that, depicted in blue, there is a steady increase in the queue of scan requests from the start of the simulation. After a certain period, the purple line clearly indicates that there are JSONs that fail to be processed on time, as the number of blockchain transaction requests increases. This highlights that, in a system with this architecture, where a request needs to be processed every 4 ticks, it is imperative to have hardware with higher specifications than the current ones, i.e. with a TS of 7 and a TB of 8. This result underlines the importance of carefully evaluating hardware capabilities in relation to the processing requirements of the system.

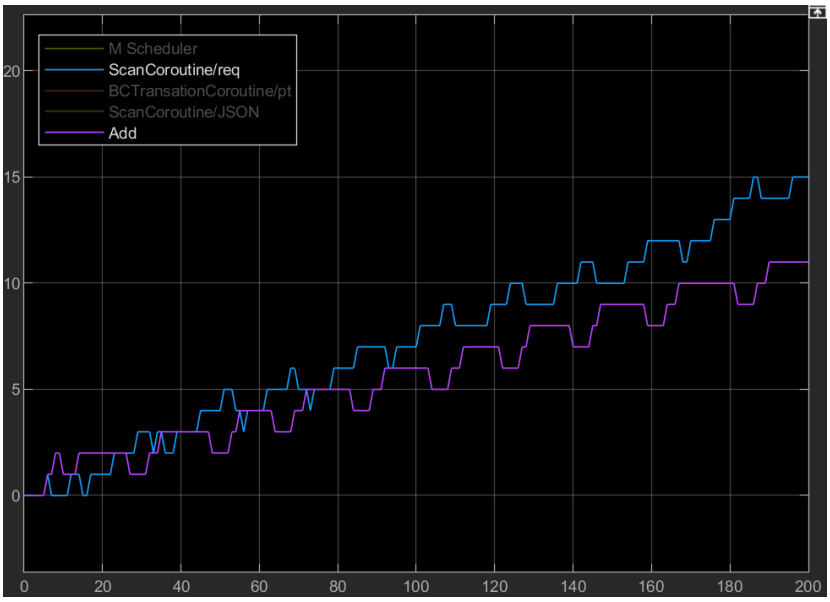


Figure 5.3: Execution results 2

In this second example, the parameters used are B = 5, TS = 3 and TB = 10. Analysing the diagram, we can see that the two lines, the blue one and the purple one, tend to come closer together. This phenomenon indicates that one of the two main coroutines is limiting the other. From the graphical representation, we can see that the scanning system is able to read the data correctly, but is slowed down by blockchain transactions, which take too long to process. This situation clearly suggests that it is crucial to choose hardware with an adequate specification to ensure the smooth operation of the system. It is essential to consider the performance of each component to optimise overall efficiency and prevent bottlenecks in the processing of requests.

.

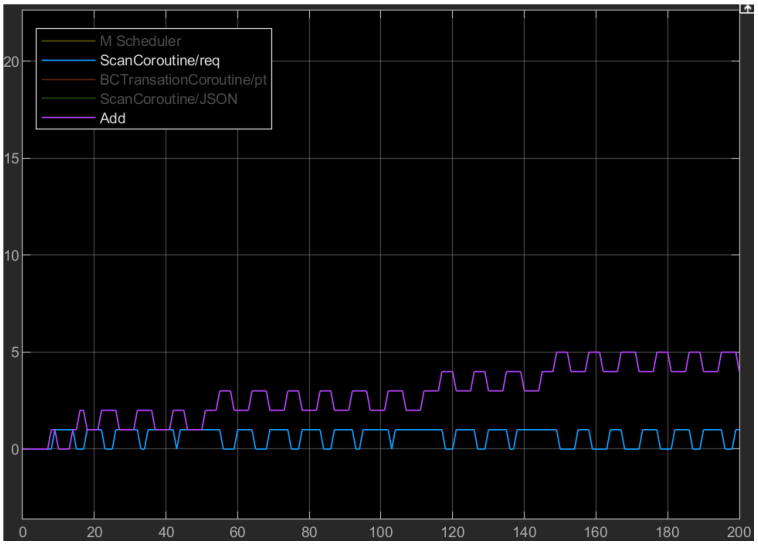


Figure 5.4: Enter Caption

This last simulation example analysed by the scope presents the parameters B = 8, TS = 3 and TB = 2, which are the most suitable for the specifications required for optimal system operation. In the simulation performed, the image clearly shows that the scanning requests are practically zero, which indicates an effective management of scanning operations. However, we also note the presence of some unprocessed JSON within the blockchain transaction coroutine. This may suggest that hardware is required to further reduce the processing time of the blockchain coroutine to ensure that all requests are handled in a timely manner and without delay. Interestingly, although the parameters in this simulation show a better ability to process scan requests, the convergence of data occurs much slower than in the previous examples. This phenomenon indicate that, although positive results have been achieved, there is still room for improvement and optimisation for the system, which could be explored by adjusting the hardware specifications and optimising the coroutines involved. Ultimately, the data suggest a thorough reflection on how to balance the different components of the system to maximise overall efficiency.

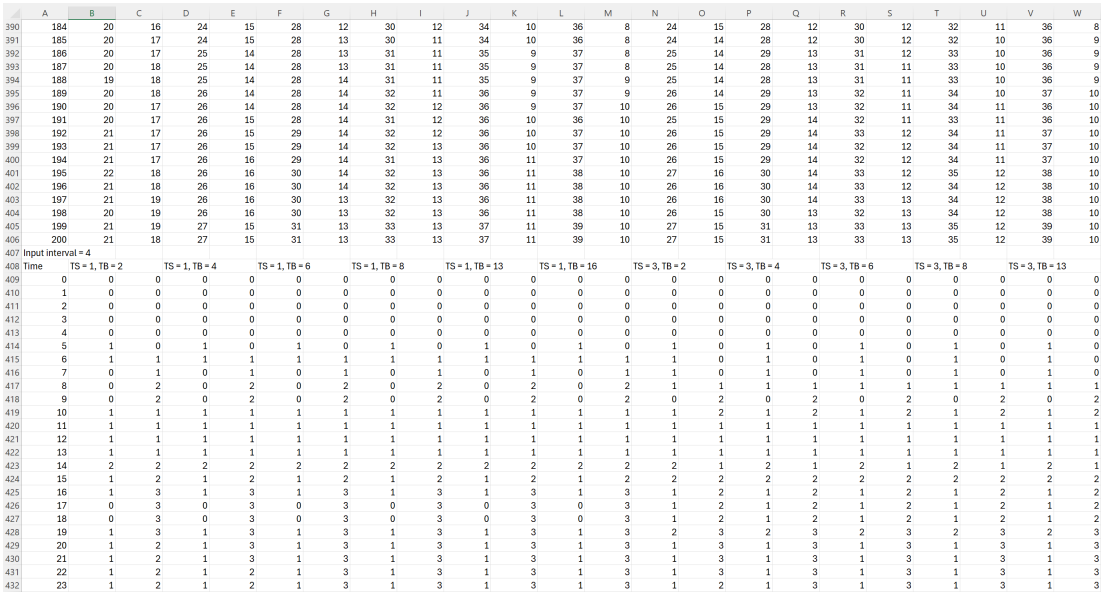
5.2.2 Output data file

From the graphs shown in the figures above, it is possible to clearly identify bottlenecks and possible slowdowns in the system. However, it is important to note that Simulink only allows data from one simulation to be displayed at a time, thus limiting the ability to compare different parameter sets directly. This limitation makes it difficult to obtain an overall view of the system's performance.

To address this problem, we have implemented a function in the Matlab scripts, used to run the simulation for each parameter combination, dedicated to collecting and processing data. This function uses specific libraries to save the results in a file with the extension .xlsx, a format that can be easily opened with any spreadsheet editor. This approach offers numerous advantages: not only does it allow a large amount of data

to be visualised in a structured manner, it also facilitates the processing and graphical representation of the information. Users can, therefore, generate graphs, statistics and other visualisations directly from the spreadsheet, enabling deeper analysis and a better understanding of the relationships between different parameters. In this way, a more systematic and integrated approach to the analysis of simulated data is fostered, improving the effectiveness of system evaluation. Because of this, we decided to arrange the simulation data in 3 different views:

Mass Data View:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
390	184	20	16	24	15	28	12	30	12	34	10	36	8	24	15	28	12	30	12	32	11	36	8
391	185	20	17	24	15	28	13	30	11	34	10	36	8	24	14	28	12	30	12	32	10	36	9
392	186	20	17	25	14	28	13	31	11	35	9	37	8	25	14	29	13	31	12	33	10	36	9
393	187	20	18	25	14	28	13	31	11	35	9	37	8	25	14	28	13	31	11	33	10	36	9
394	188	19	18	25	14	28	14	31	11	35	9	37	9	25	14	28	13	31	11	33	10	36	9
395	189	20	18	26	14	28	14	32	11	36	9	37	9	26	14	29	13	32	11	34	10	37	10
396	190	20	17	26	14	28	14	32	12	36	9	37	10	26	15	29	13	32	11	34	11	36	10
397	191	20	17	26	15	28	14	31	12	36	10	36	10	25	15	29	14	32	11	33	11	36	10
398	192	21	17	26	15	29	14	32	12	36	10	37	10	26	15	29	14	33	12	34	11	37	10
399	193	21	17	26	15	29	14	32	13	36	10	37	10	26	15	29	14	32	12	34	11	37	10
400	194	21	17	26	16	29	14	31	13	36	11	37	10	26	15	29	14	32	12	34	11	37	10
401	195	22	18	26	16	30	14	32	13	36	11	38	10	27	16	30	14	33	12	35	12	38	10
402	196	21	18	26	16	30	14	32	13	36	11	38	10	26	16	30	14	33	12	34	12	38	10
403	197	21	19	26	16	30	13	32	13	36	11	38	10	26	16	30	14	33	13	34	12	38	10
404	198	20	19	26	16	30	13	32	13	36	11	38	10	26	15	30	13	32	13	34	12	38	10
405	199	21	19	27	15	31	13	33	13	37	11	39	10	27	15	31	13	33	13	35	12	39	10
406	200	21	18	27	15	31	13	33	13	37	11	39	10	27	15	31	13	33	13	35	12	39	10
407	Input interval = 4																						
408	Time	TS = 1, TB = 2		TS = 1, TB = 4		TS = 1, TB = 6		TS = 1, TB = 8		TS = 1, TB = 13		TS = 1, TB = 16		TS = 3, TB = 2		TS = 3, TB = 4		TS = 3, TB = 6		TS = 3, TB = 8		TS = 3, TB = 13	
409	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
410	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
411	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
412	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
413	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
414	5	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
415	6	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0
416	7	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	1	0
417	8	0	2	0	2	0	2	0	2	0	2	0	2	1	1	1	1	1	1	1	1	1	1
418	9	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2	0	2
419	10	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	2	1	2	1	2	1	2
420	11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
421	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
422	13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
423	14	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	1	2	1	2	1	2	1
424	15	1	2	1	2	1	2	1	2	1	2	1	2	2	2	2	2	2	2	2	2	2	2
425	16	1	3	1	3	1	3	1	3	1	3	1	3	1	2	1	2	1	2	1	2	1	2
426	17	0	3	0	3	0	3	0	3	0	3	0	3	1	2	1	2	1	2	1	2	1	2
427	18	0	3	0	3	0	3	0	3	0	3	0	3	1	2	1	2	1	2	1	2	1	2
428	19	1	3	1	3	1	3	1	3	1	3	1	3	2	3	2	3	2	3	2	3	2	3
429	20	1	2	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3
430	21	1	2	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3
431	22	1	2	1	2	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3	1	3
432	23	1	2	1	2	1	3	1	3	1	3	1	3	1	2	1	3	1	3	1	3	1	3

Figure 5.5: 1st data view

This view represents the most comprehensive collection of simulation data, as all results obtained for each time instant and for each parameter combination are stored in this sheet. Consequently, it is the largest sheet, as no data is overlooked during the entire simulation process. This breadth of information allows a detailed evaluation of the system's performance, which is essential for analysis and optimisation.

Looking at the image, we can see how the data is organised in a systematic and clear manner. Each section of the sheet is introduced by a line specifying the value of B for that set of simulations, marked 'input interval = B'. This makes it immediate for the user to identify to which combination of parameters the subsequent data belong.

Below this header, the first column on the left indicates the time tick of the simulation, providing a precise time reference for each piece of data collected. Next, the columns are arranged in pairs: on one side, we find the request values, representing the scan requests that have not yet been processed; on the other, there are the JSONs to be processed. Thanks to these two types of data, it is possible to make in-depth predictions and analyses, as shown in the data shown in the previous scopes. This structure thus allows every possible combination of parameters to be explored, facilitating the identification of trends and bottlenecks in the system. In summary, this sheet not only collects data, but also serves as a crucial tool for analysing and evaluating the simulated

system's performance.

Here below is an example of some graphs we drew using the built in features of a spreadsheet software

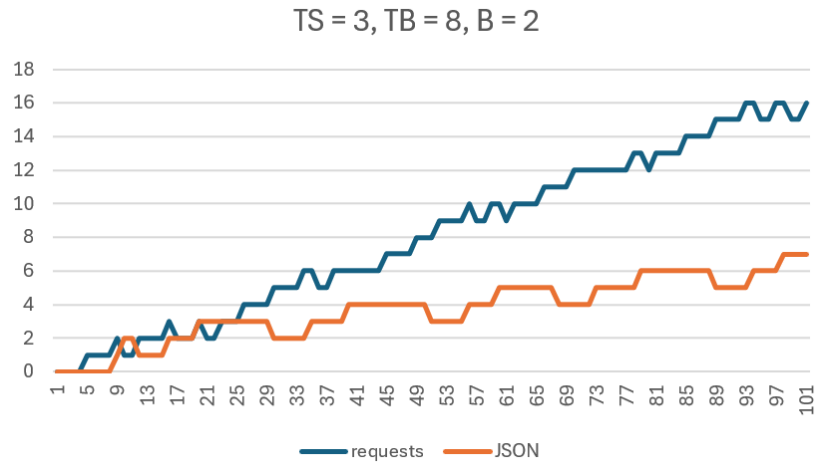


Figure 5.6: diagram from 1st data view

Summary Data View:

	A	B	C	D	E	F
1	B	TS = 1, TB = 2 (Requests)	TS = 1, TB = 2 (BTCqueue)	TS = 1, TB = 4 (Requests)	TS = 1, TB = 4 (BTCqueue)	TS = 1, TB = 6 (Requests)
2	2	43	19	49	16	53
3	3	21	18	27	15	31
4	4	8	18	14	15	18
5	5	1	17	7	14	11
6	6	1	11	2	14	6
7	7	1	7	2	10	2
8	8	1	4	1	8	2
9	9	0	1	0	6	0
10	10	4	0	0	3	0

Figure 5.7: 2nd data view

This view represents a reworking of some of the data in the first view. It was introduced to address the problem of the first view, where the selection of all simulation data can make it difficult to maintain focus on the overall objective. In fact, the first view is excessively scattered, making the analysis of the results complicated. Therefore, only the final values of the scanner requests and the JSON to be processed are collected in the second sheet. Although this approach does not provide a complete overview of the process that led to those results, it does allow us to identify which parts of the system present difficulties in handling operations.

In the image below, we can observe two graphs generated from this view, in which the value of B is positioned on the X-axis. In the first graph, we note that the simulation does not present any problems when the value of B is greater than or equal to 5, since the requests tend to converge towards zero. However, we also show that for values of B below 5, the system struggles to maintain optimal performance. In such cases, it is suggested to consider a hardware upgrade to improve scanner performance.

From the second graph, on the other hand, we can analyse the number of unprocessed JSONsons related to pending blockchain transactions. In this situation, the results are even more worrying: the system only works properly when the incoming scan requests do not exceed 1 in every 9 ticks of the simulation. These observations show that the system can only effectively handle an amount of requests (B) not exceeding 1 every 9 seconds or more. These data clearly highlight the limitations of the system and the need for action to optimise its performance.

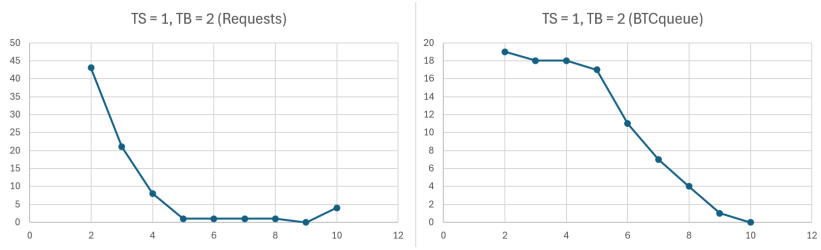


Figure 5.8: diagram from 1st data view

Matrix Data View:

	A	B	C	D	E	F	G
1	B = 2 Requests						
2	TS/TB	2	4	6	8	13	16
3	1	43	49	53	55	59	61
4	3	50	54	56	58	62	64
5	5	54	56	58	60	64	64
6	7	58	60	61	63	65	67
7	12	66	66	66	66	68	69
8	17	69	69	69	69	70	70
9							
10	B = 2 BctQueue						
11	TS/TB	2	4	6	8	13	16
12	1	19	16	14	14	12	11
13	3	15	13	13	12	10	8
14	5	13	13	12	11	9	9
15	7	8	10	10	8	8	6
16	12	1	2	5	6	6	5
17	17	2	2	2	3	5	5
18							
19							
20	B = 3 Requests						
21	TS/TB	2	4	6	8	13	16
22	1	21	27	31	33	37	39
23	3	27	31	33	35	39	41
24	5	31	33	35	37	41	41
25	7	35	37	38	40	42	44
26	12	43	43	43	43	45	46
27	17	46	46	46	46	47	47

Figure 5.9: 3rd data view

This is the latest view we have made available for our simulator. This view is designed to present the data in a similar way to the previous view, but on a three-dimensional plane, thus offering a new level of understanding. The data is organised within a matrix, with TS and TB values positioned on the horizontal axes. Recall that TS represents the average time of the scan coroutine, while TB represents the average time of the blockchain transaction coroutine. On the vertical axis, or Y-axis, the number of requests or JSONs to be processed in blockchain transactions at the conclusion of the simulation is instead displayed.

Similar to the previous case, it is not possible to observe the trend of the signals during the execution of the simulation. However, the final value is a good indication of the system's performance, providing useful information on how the system performs overall. This three-dimensional representation allows the interactions between the different parameters and their influences on the system's performance to be analysed. In the section below, we show an example of this matrix view, accompanied by a three-dimensional representation that clearly illustrates the relationship between the various factors involved. In this way, users can get a clearer idea of how different configurations affect performance and identify possible areas for improvement.

B = 7 Requests						
TS/TB	2	4	6	8	13	16
1	1	2	2	4	8	9
3	1	2	4	6	9	11
5	2	4	6	7	11	12
7	5	7	8	10	12	14
12	13	13	13	13	15	16
17	16	16	16	16	17	17

Figure 5.10: example of matrix data

In this image we show how the data are collected in a matrix to represent the performance of the system in handling requests for the scanner when B = 7, at all the possible combination of the TS and TB parameters, in fact each cell of the matrix contains the number of final requests when TS and TB are the respective value on the axis of the matrix. We can than use this to plot a 3D diagram of this matrix as a plane, when the plane is near the 0 value, we can assume that the system is able to perform correctly, however when the plane start to converge upward, we can assume that there is a bottleneck in the respective component, for the parameters TS and TB at the intersection with the axis in that point. The image below, shows an example of one plane generated by a spreadsheet software.

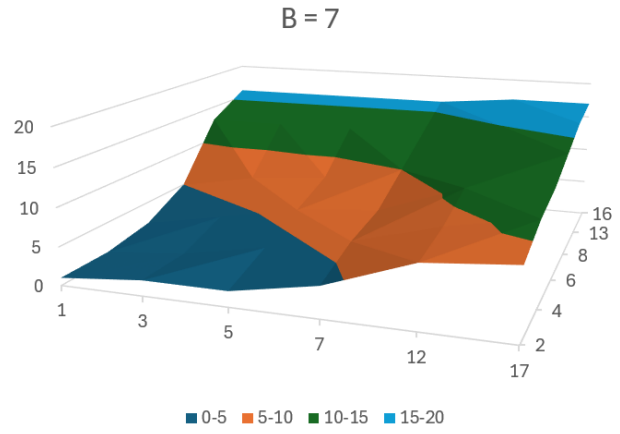


Figure 5.11: example of diagram 3D view from matrix

6. Conclusion

In this work, we presented the development of an advanced simulator capable of analysing the performance of a complex system, such as an embedded system intended to manage blockchain transactions, through the use of different sets of parameters. The main objective was to identify possible bottlenecks and assess the system's ability to handle a high workload and multiple requests. Through the use of advanced tools such as Simulink and Matlab, we collected and organised the data in different formats, providing a solid basis for a thorough and comparative analysis of the simulations. This made it possible to accurately identify failure loads, hardware slowdowns and system performance limits. It is important to emphasise that the value of B, unlike TS and TB, is not an intrinsic parameter of the system, but an external input that was deliberately set to low values to simulate a high workload, thus testing the performance of the system under stress conditions.

One of the main challenges faced during this work was the correct simulation of the embedded system architecture, based on a coroutine model. Thanks to the accuracy of the simulator, we were able to faithfully represent the behaviour of the real system, confirming its reliability in assessing performance and resource requirements. The simulation proved to be able to replicate the embedded system architecture with a high degree of fidelity, providing detailed information on the system's behaviour under different operating conditions.

One of the simulator's main advantages is its flexibility and customisability. Users can modify simulation parameters to suit a wide range of operating scenarios, making it an extremely versatile tool. The three result visualisation modes - a detailed view of the entire simulation, a final summary and a three-dimensional representation - allow users to explore data from different angles and better understand the interactions between complex parameters such as TS (mean scan coroutine time) and TB (mean blockchain transaction coroutine time). This visual approach helps to quickly identify critical areas and opportunities for optimisation.

The data obtained shows that the developed simulator has proven to be an effective tool for performance analysis and the design of more efficient embedded systems. The collected data provide a solid basis for optimising hardware resources and operating parameters, improving system processing capacity and reducing response times to demands. However, the work does not stop there: further developments could further improve the accuracy and usefulness of the simulator, such as the integration of predictive models based on machine learning, which could suggest optimal configurations based on historical simulation data and expected operating conditions.

The experience gained in the development of this simulator can be extended to other industrial and technological contexts, where the management of complex processes requires an in-depth and precise analysis of available resources. This tool can help prevent inefficiencies, reduce costs and improve the overall robustness of the system.

6.1 Future works

Despite the positive results of the developed simulator, there are several directions in which the project can be further improved and expanded. Future developments can not only strengthen the analysis capabilities, but also adapt the simulator to new technological contexts. In this section, we will explore some of the potential areas for development.

1. **Using the R language for data analysis:** One of the most promising future developments concerns the integration of languages commonly used in the field of data science, such as R or Python, to improve the visualisation and analysis of the data collected by the simulation. Matlab already offers powerful analysis tools, but languages such as R are particularly suitable for manipulating large datasets and creating advanced visualisations. The use of R would allow the exploitation of libraries dedicated to statistics and data analysis, such as ggplot2 for the creation of customised and interactive graphs, or dplyr for efficient data management. This approach could provide an even more detailed and dynamic view of the interactions between the various simulation parameters, facilitating the identification of complex patterns and performance analysis in a more intuitive manner. Furthermore, the creation of automated reports with R Markdown or Shiny Dashboards would allow for easier sharing and interpretation of results with other interested parties.

2. **Parallelisation of coroutines:** Another interesting future development concerns the possibility of modifying the coroutine scheduler to support parallel execution, especially in the context of multicore systems. Currently, coroutines are run sequentially, but as mentioned in the initial part of the report, a future system model could implement a multicore architecture, where each coroutine is executed on a separate core or hardware. Implementing a system capable of executing coroutines in parallel would allow simulating scenarios much closer to the operational reality of complex embedded systems, where processing occurs in a distributed manner. Modifying the scheduler to allow this is extremely simple, however, this development entails significant challenges, such as managing the mutual exclusion of shared resources, the need for synchronisation between coroutines and the risk of race conditions. The simulator could be updated to include synchronisation mechanisms and tools to detect and solve these problems, making the system more robust and realistic.

3. **Integration of machine learning techniques for parameter optimisation:** Another possible future development is the integration of machine learning for the automatic optimisation of system parameters. Currently, performance analysis and identification of bottlenecks is done by running multiple simulations with different parameter sets, we believe that with some more work, this could be switched with a machine learning model.

Bibliography

[CL09] Y. Chiu and M.-H. Li. "Petri-Net Modelling of an Assembly Process System". In: *ResearchGate* (2009). Accessed: 2024-09-13. URL: https://www.researchgate.net/publication/237258243_PETRI-NET_MODELLING_OF_AN_ASSEMBLY_PROCESS_SYSTEM.

[CL18] C.-Y. Cheng and Y.-M. Lee. "A Petri Net-based Supply Chain System". In: *ResearchGate* (2018). Accessed: 2024-09-13. URL: https://www.researchgate.net/publication/328859579_A_Petri_Net-based_Supply_Chain_System.

[Dsp] *DSP Signal theory*. https://it.wikipedia.org/wiki/Elaborazione_numerica_dei_segnali. Accessed: 2024-09-13.

[Esp] *ESP32 manufacturer website*. https://www.espressif.com/en/products/socs/esp32.

[HCL23] Y. Hu, Z. Chen, and H. Liu. "Blockchain Application for Fish Origin Certification". In: *Innovations for Community Services (I4CS 2023)*. Springer, 2023, pp. 167–181. DOI: 10.1007/978-3-031-46784-4_13. URL: https://link.springer.com/chapter/10.1007/978-3-031-46784-4_13.

[Hip] *HIPS Software for Petri Nets*. https://www.oris-tool.org/tutorial. Accessed: 2024-05-7.

[LW09] P. Lee and X. Wang. "Petri-net based application for supply chain management". In: *2009 IEEE International Conference on Industrial Engineering and Engineering Management*. Accessed: 2024-09-13. 2009, pp. 728–732. DOI: 10.1109/IEEM.2009.5373050. URL: https://ieeexplore.ieee.org/document/5373050.

[Mat] *MATLAB Function Block*. https://it.mathworks.com/help/simulink/ug/what-is-a-matlab-function-block.html.

[Pip] *Software PIPE*. https://sarahtattersall.github.io/PIPE/.

[Pro] *Probability distribution theory*. https://it.mathworks.com/help/stats/probability-distributions-1.html.

[Sima] *Simulink Blok Diagram*. https://it.mathworks.com/help/simulink/blocks.html.

[Simb] *Simulink mathworks website*. https://it.mathworks.com/products/simulink.html.

[Simc] *Using Simulink Data in MATLAB to Run Multiple Simulations*. https://it.mathworks.com/help/simulink/slref/simulink.simulationoutput.html.

[SS07] M. Silva and A. Solanas. “Structural Modelling With Petri Nets”. In: *IFAC Proceedings Volumes* 40.3 (2007). Accessed: 2024-09-13, pp. 126–132. DOI: 10.3182/20070711-3-FR-2916.00135. URL: https://www.sciencedirect.com/science/article/pii/S1474667017446088/pdf.

Acknowledgments

We would like to express our sincere gratitude to Professor Rosario Culmone and PhD student Riccardo Petracci for their invaluable support during the development of this project. Their guidance, advice and availability were crucial to the success of the work. Our special thanks go above all for the unique learning opportunity offered, which represented an extraordinary opportunity for personal and professional growth, contributing significantly to our academic development and understanding of the topics addressed.