People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdès

Institute of Electrical and Electronic Engineering
Department of Electronics

Project Report Presented in Partial Fulfilment of

the Requirements of the Degree of

'MASTER'

In Electrical and Electronic Engineering

Title:

Emulating Multi-node Embedded Systems Using Renode

Presented By:

- **HACHEMANE Abderrahemane**
- **KHOUAS Aness Mohamed**

Supervisor:

Dr. MAACHE Ahmed

Registration Number:……../2014

Abstract

Emulation is crucial prior to high-scale, complicated embedded systems, particularly for pedagogical and prototype utilization. The emulation process offers an ideal and cost-effective approach, where we can be in a position to analyze and evaluate system designs with the assumptions and some flaws that can be worked out before implementing them in the actual system. This practice helps greatly in teaching since students can train on such concepts without the necessity of having costly hardware.

This project introduces Renode, which is an effective emulation tool, to emulate an entire embedded system including interconnected-node systems. Fast prototyping and the emulation of various hardware elements, including their functionality, flexibility, and interactive communications, make Renode a versatile platform for development.

Our work starts with implementing a single-node embedded system using ZedBoard and FreeRTOS, demonstrating the basic setup and functionality. We then expanded this implementation to accurately emulate a multi-node system, highlighting Renode's ability to handle complex, interconnected environments.

In fact, as our findings indicate, Renode is a valuable and efficient tool for modeling embedded systems and IoT networks. The emulations were effective and the approach proved to be sound in practice despite several difficulties that were met along the way.

Dedications

”In the Name of Allah, the Most Merciful, the Most Compassionate All praise be to Allah, the Lord of the worlds. May prayers and peace be upon the Prophet Muhammad, His servant and messenger. This work is dedicated wholeheartedly to my beloved parents, who have been a constant source of inspiration, providing me with hope and encouragement when I faced moments of doubt and despair. They have always been there, guiding and supporting me.

Acknowledgments

First and foremost, we would like to extend our deepest gratitude to God Almighty who provided us with His blessing and the opportunity to successfully conclude our project.
In the successful accomplishment of our final year project titled ”Emulating Multi-node Embedded Systems Using Renode” we would like to express our deepest gratitude to our supervisor, **Maache Ahmed**. We are immensely grateful for his valuable advice regarding our project and future career prospects. His willingness to assist us, steadfast encouragement, and continuous support were instrumental in making this project a reality. We deeply appreciate his patience, dedication, and unwavering belief in our abilities.
Lastly, we are immensely thankful to our parents and friends for their unwavering support in completing this report. We would like to extend our gratitude to all the individuals who have supported us throughout this journey.

Contents

List of Figures

List of Abbreviations

Abbreviation	**Meaning**
SoC	System on Chip
AXI	Advanced eXtensible Interface
ADC	Analog to Digital Converter
IDE	Integrated Development Environment
RTOS	Real-Time Operating System
GPIO	General-Purpose Input/Output
IP	Intellectual Property
CPU	Central Processing Unit
FPGA	Field Programmable Gate Array
PS	Processing System
PL	Programmable Logic
DDR	Double Data Rate
DMA	Direct Memory Access
UART	Universal Asynchronous Receiver-Transmitter
IoT	Internet of Things

General introduction

The growth of embedded systems and the popularity of IoT devices have impacted many fields ranging from consumer electronics to industrial processes and much more. As these systems grow more sophisticated and integrated, so has the demand for proper tools for the development and testing of these systems. As a result of these changes, emulation, a practice of simulating the behavior of hardware using software, has become practically indispensable to the lifecycle of embedded systems. Through emulation, developers and educators can explore, assess, and optimize system designs at a relatively low cost without physically implementing them on system hardware.

Emulation is especially useful in educational environments by giving students actual experience in designing and debugging embedded systems without having to invest in a lot of actual hardware. Nonetheless, there is a lack of well-documented and versatile emulation tools suitable for dealing with the complexity and large numbers of nodes needed in today's multi-node embedded systems.

The objectives of this work are to develop the topic of Renode as a strong and multipurpose emulation environment to explain the emulation idea. Renode allows emulating entire systems, including complex multi-node systems, providing developers with a versatile environment for rapid prototyping and testing. Leveraging various types of hardware components and supporting interactive communication offers extensive adaptability, making Renode a versatile tool that would effectively serve both educational and professional settings.

In this report, the first chapter will introduce the theoretical concepts behind the work, defining embedded systems, emulation, and multi-node systems. Next, in the second chapter, the reader will get familiar with the tools used in the project such as ZedBoard, Renode, etc. Then, the third chapter will discuss the implementation of a single-node embedded system using the ZedBoard. At the end, the fourth chapter will show the multi-node emulation.

Chapter 1

Introduction to Multi-Node Embedded Systems & Emulation

1.1 Introduction

This is a brief chapter that provides the reader with a point of entry into the understanding of what embedded systems, emulation, and multi-noding mean. These three subjects are considered the basis of the current embedded system design and development to equip them with suitable approaches and enabling technologies for developing highly complex, dependable, and portable systems.

Starting with the definition of the first topic, it is necessary to define embedded systems, which can be described as specialized computer systems intended for the completion of particular functions in parallel with mechanical or electrical systems[1]. These systems are almost a standard for current technologies, being integrated with all sorts of devices, from home appliances to complex industrial equipment. Knowing the basics of embedded systems and how they can be implemented requires comprehending the nature and make-up of embedded systems.

Next, emulation is described in detail; it is the process of providing a software model for emulating the behavior of hardware components. Emulation permits designers to verify their new system using the models without the utilization of physical hardware, thereby reducing the issues in creating the physical embedded system. In this section, three aspects will be discussed because they are the key motives for emulation: the emulation gives a controlled environment for testing; it helps to debug the application; and finally, it is cheaper than developing using devices.

Next, we assess multi-noding, a procedure through which several complex nodes are connected with each other to accomplish one task. Multi-node systems are gradually being incorporated in many applications like sensor networks, self-driving cars, and smart industry production lines. This section will explain how systems with multiple nodes are designed, the issues faced during the synchronization and communication of multiple systems, and the strategies that can be adopted to solve these issues.

By the end of this chapter, the reader will be well informed about what embedded systems are and their requirements through emulation, and the issues associated with multi-noding. This knowledge will form a base and build their understanding of the advanced topics covered in the following chapters which cover detailed descriptions of all these topics and their interaction.

1.2 Embedded Systems

Embedded systems are stand-alone and tailored computing systems whose application is limited to a specific function. Such systems are typically defined by their functionality that includes working in real-time, low power consumption, and a resource-constrained nature. They can be observed in numerous applications, from electronic devices to manufacturing equipment.

An embedded system, in its simplest form, consists of the following major components: a microcontroller or microprocessor, memory, input/output ports, and firmware. The microcontroller or microprocessor serves as the brain of the system, executing programmed instructions to perform desired tasks. Memory is classified into ROM and RAM. ROM is usually used to store the firmware while RAM is used to store the temporary data generated in real-time execution. The third part is the I/O devices such as ADCs, and simple GPIOs which are used to communicate with the outside world, facilitating interconnection and data sharing.

The key feature of embedded systems is that the system must interact with the physical world on a real-time basis, thereby producing or responding within a resultant time. This is important as it enables the embedded system to effectively execute the intended function in a real-time manner. Moreover, embedded systems are developed to operate on low power and occupy minimum physical space, thereby being power and size-efficient. They are also made for the purpose of performing certain functions and are usually designed for efficiency, dependability, and affordability; hence, they cannot be underestimated in various innovative technological uses. According to their usage, embedded systems can be classified into different categories. Single-node embedded systems are most common in the industry, multi-node embedded systems are commonly used in home automation systems and interconnected sensor networks, portable systems like smartphones, and real-time embedded systems.

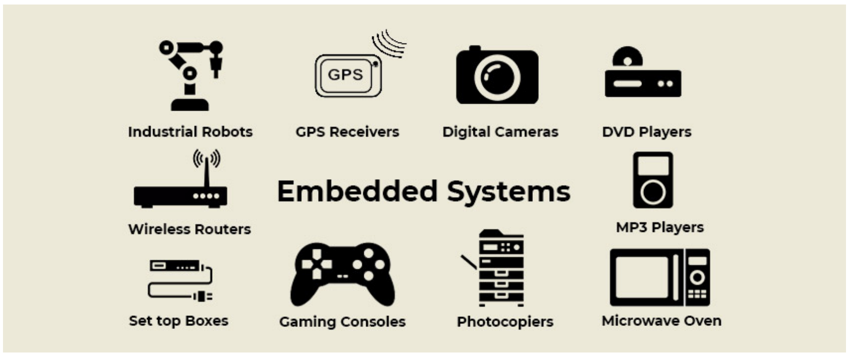


Figure 1.1: Embedded systems examples[2]

There are many substantial obstacles that engineers who are involved in the design and implementation of the embedded system need to overcome in order to achieve the desired result. The first problem is that of real-time performance because usually embedded applications have strict performance requirements and the system should be capable of processing data and reacting to events in a particular time[3]. This means that schedulability analysis and optimization depend on precise timing analysis. Real-Time Operating Systems should be thanked for offering rather decent solutions to this issue. Another key issue is the utilization of limited resources like CPU, memory, and power where proper coding methodology and the choice of the right industry standard boards play a vital role. Moreover, reliability and fault tolerance are important attributes of embedded systems because the devices are used in various important applications that have strict requirements for device performance. This involves extensive testing and data validation processes to guarantee severe stability under different scenarios. Communicating with a large number of peripherals and various external devices imposes additional challenges with regard to the protocols and interfaces used to interact with the devices. Finally, the technology is growing at a very fast rate, and this implies that the embedded system needs to be designed with the ability to be modified in the future but not redesigned[1]. Addressing these challenges requires a combination of innovative design, thorough testing, and a deep understanding of both hardware and software aspects of embedded systems.

In embedded systems, it is not a very rare practice to execute applications on the bare metal that is the hardware layer. But when it comes to real-time jobs that require precise efficient solutions, employing special operating systems known as real-time operating systems or RTOS are more effective. These systems are expected to deal with real-time applications, thus being responsive to events promptly. There is a very famous RTOS called FreeRTOS. In the further sections of this report, the reader will be introduced to FreeRTOS software: its functionality, possibilities, and applications. The integration of Multi-node technology into embedded systems has revolutionized the way devices and systems interact and communicate with each other. Multi-node integration allows embedded systems to connect to each other, enabling them to collect, analyze, and exchange data in real-time. This connectivity opens up a wide range of possibilities for applications in various industries, from smart home devices to industrial automation and highway systems[4].

1.3 Hardware Emulation

This section will give a detailed discussion of the several features that have made emulation significant in the implementation phase. This will involve an analysis of the issues and characteristics of today's implementation environments and expectations in areas such as risk, cost, time-to-market, and resource availability to the business. It will serve as a theoretical framework to ensure that everyone involved in this study has a good grasp of emulation's importance in handling issues of the implementation phase before their appearance.

To put it in simple terms, emulation may be understood within the context of implementation preparation as the imitation of the functionalities and behaviors of a system, it can be a computer, an SoC, or a piece of hardware. This means developing or working on a model that can mimic the conditions and parameters of the original system by emulating tests before it is released to the real environment. Emulation can be helpful in recreating a real environment, testing for various problems, and estimating how changes or upgrades will affect production without affecting the actual environment. It makes it easier for organizations to understand the behaviors of various systems in order to predict potential problems that can arise and find ways to effectively address problems during their implementation[5]. Additionally, emulation plays an important role in improving control over risks to achieve better system tests under various conditions and situations. This makes it possible for organizations and individuals to prepare and possibly look for ways of avoiding disruptions in case they happen during the implementation of the new process. Further, emulation helps in cost control by isolating problems that can be corrected in the early stage without having to spend large amounts of money on rectifications later during system implementation.

By promoting emulation, learning can be enhanced through the provision of cheaper software and hardware that cannot be afforded by any learner. This makes the use of emulation-based experiments effective in enabling real-like learning without having to spend very large amounts of money such as costs of space, rent, maintenance, and occupational health and safety measures[6]. Moreover, in emulation-based virtual laboratories, students are capable of performing the experiments on the same platform and without the actual hardware in place, thus there is likely to be even better understanding and knowledge retention on the subject content and skills being taught.

Renode is an emulation framework available in the sphere of embedded systems that has been created by Antmicro[7]. Despite the fact that this is open-source, Renode can efficiently emulate a variety of processors, boards, and even devices. Renode is also an efficient emulator for developers to model, experiment, and debug their embedded systems, enabling rapid prototyping in pre-silicon configurations[7]. The subsequent chapter will thus aim at describing Renode in detail, in terms of the tools it consists of, its potential uses in the context of embedded system development, its advantages, and more.

The topic of emulation in embedded systems can be a challenging and broad one that has many

Chapter 2

Hardware & Software Overview

2.1 Introduction:

In the previous chapter, the reader was introduced to the topic of multi-node embedded systems and emulation. In this chapter, which forms the bridge between the theoretical background and the practical elements of the work, the reader will be provided with a guide to the equipments -both hardware and software- which are vitally involved in the designing and the implementing of these systems.

First of all, we are going to familiarize with the Zedboard Development Kit, a full-featured platform which is implemented around Zynq-7000 SoC from Xilinx. Next, we will explore the AMD Design Suite: Vivado and Vitis are the two software suitable for designing hardware and runtime software applications respectively. Vivado Design Suite is a software tool developed by AMD to perform the analysis and synthesis of the HDL design and incorporating IP cores. On the other hand, Vitis provides a unified software development environment for creating embedded applications for FPGAs and SoCs. Then, we will explore the Renode Emulator, an all-in-one platform that emulate the design, debug and test the phases of embedded system. Renode makes it possible to test software on an emulated environment before its integration with the physical hardware and this make the process of hardware development less expensive, time-consuming, and with less risk factors.

2.2 Zedboard Development kit

The ZedBoard is a powerful and versatile development platform based on the Xilinx Zynq-7000 All Programmable System-on-Chip (SoC). This board is designed to facilitate the development and prototyping of embedded systems by integrating a dual-core ARM Cortex-A9 processor with FPGA fabric, providing a comprehensive environment for both hardware and software design. Its extensive feature set makes it an ideal choice for a wide range of applications, from academic research and development to industrial automation and consumer electronics[8].

2.2.1 ZedBoard Key Features

As mentioned before, Zedboard has plenty of features and characteristics that makes it exciting platform to develop both hardware and software on. One of the most important aspects is that the ZedBoard delivers considerable processing power and distinctive memory design as well as various peripherals. The introduced Zynq-7000 SoC is incorporating FPGA and dual core ARM processor which can be targeted for implementing custom hardware accelerators and can also efficiently perform the processing. Memory resources available with the board are DDR3 SDRAM memory, QSPI Flash and available facilities for SD card interface provide the adequate storage and high data access rate.

More so, the connectivity features supported by the ZedBoard include Gigabit Ethernet, USB, a HDMI port, a VGA port and other I/O interfaces including the PMOD connectors which are general purpose I/O. These characteristics allow to connect the Zedboard with other devices including peripherals, which defines it as a suitable solution for a vast majority of tasks within the sphere of embedded systems. It also allows designing various applications due to the extensive number of features available which are applicable in signal processing, embedded systems, and real-time control.

On the following part of this report, we will focus on the general information regarding the Zedboard, as well as the outline of advantages, processing capabilities, and possible applications of the board and in-this-project used components. Thus, knowing the working profile of the Zedboard allows for the efficient and creative development of general and embedded systems. One accessible source of information on ZedBoard is the hardware manual, which contains all the essentials regarding the specifications and key features of the device[8].

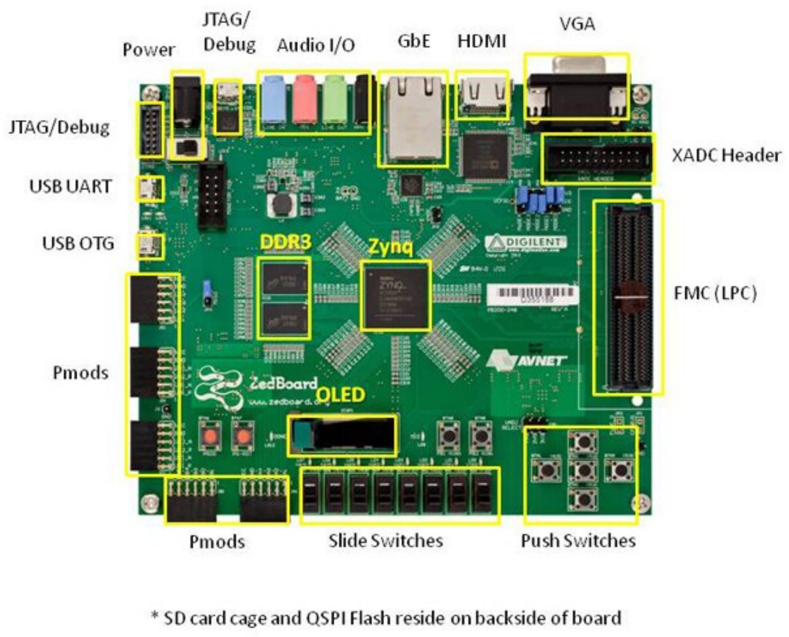


Figure 2.1: Zedborad features [9]

2.2.2 Zynq-7000 SoC

Zynq-7000 is a family of system-on-chips (with minor differences between them)developed by Xilinx. it includes both PL (programming logic) which is an FPGA and PS (processing system) represented by ARM Cortex-A9 Processor. The integration of both PS and PL enables developers to implement custom hardware accelerators, high-speed interfaces, and real-time processing capabilities in a single chip. In addition to that,the zynq-7000 Soc contains many other features among them the XADC which is an 12-bit analog to digital converter, AXI which is high-frequency bus for internal connections, DMA which is a direct access memory and many more features can be found in the zynq-7000 datasheet[10].

this section will focus on the PS, XADC, and the AXI-bus protocol due it's pivotal role in the project

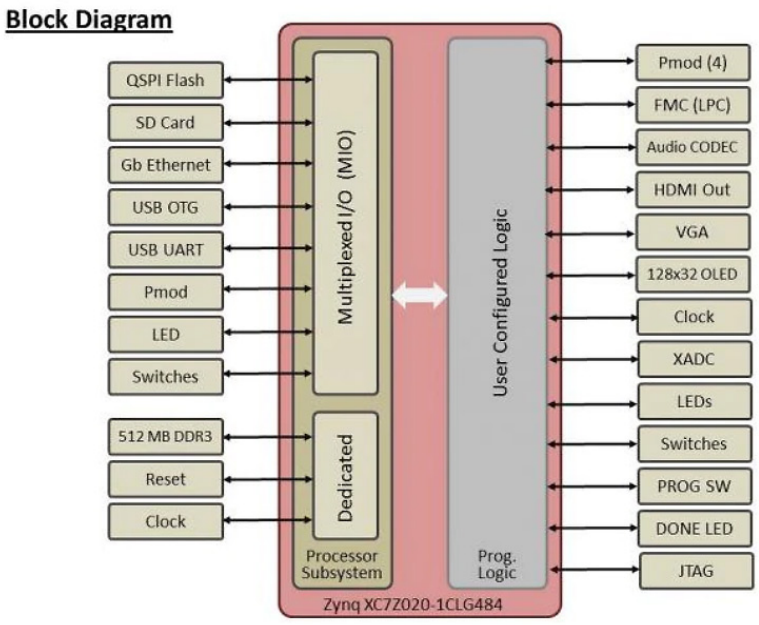


Figure 2.2: Zynq-7020 block diagram [8]

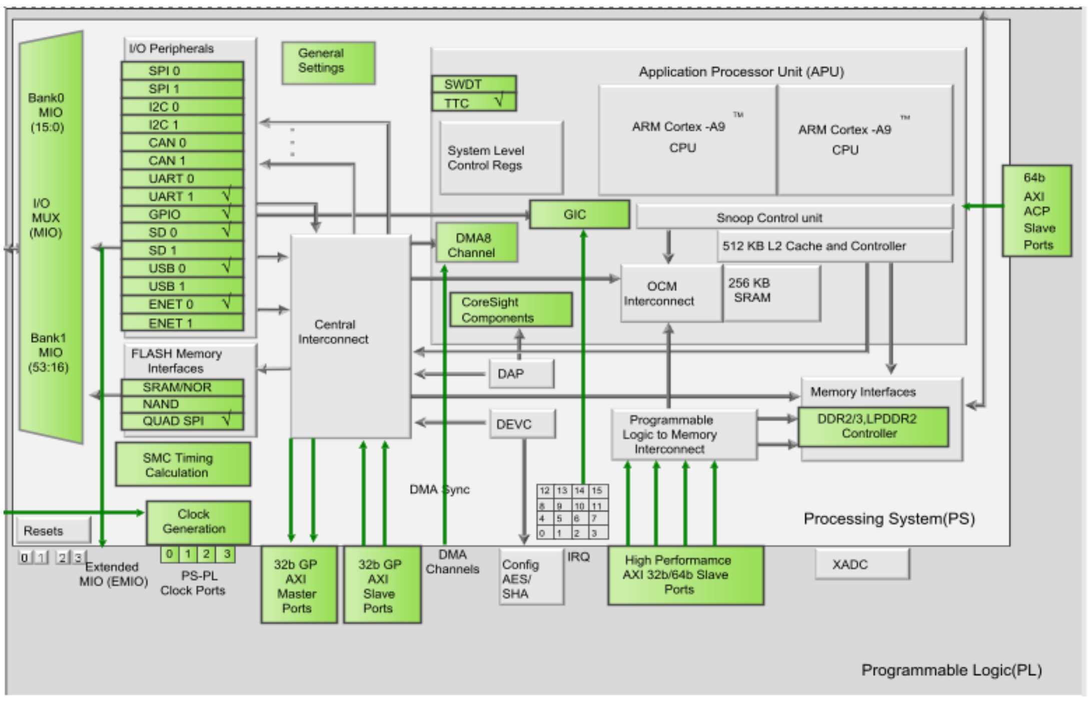


Figure 2.3: zynq-7020 Architectural Overview

2.2.2.1 Programming Logic

The Programmable Logic (PL) part in a Zynq 7000 is the Field-Programmable Gate Array (FPGA) component of the System-on-Chip (SoC). It is a programmable logic device that can be used to implement custom digital circuits and interfaces. The PL is designed to work in conjunction with the Processing Subsystem (PS), which is based on an ARM Cortex-A9 processor.

It consists of thousands (depends on zynq-7000 family member) of programmable logic elements called configurable logic blocks (CLBs). Each CLB has combinational (multiplexer) and sequential (flip-flops) elements which combined together to implement any basic to complex digital circuit. More details about the PL can be found in Zynq-7000 reference manual[10].

2.2.2.2 Processing System

The PS part is implemented using ARM Cortex-A9 dual core 32-bit μProcessor in addition to multiplexed input/output unit (MIO) used to interact with the outside world. The ARM Cortex-A9 is an implementation of ARM architecture referred to as the ARMv7-A architecture, which is a 32-bit architecture. This implementation facilitates the execution of a large number of programs and a range of applications, from simple tasks to complex computations. A key point about Cortex-A9 is that it have two cores in it, making it possible to have a parallel execution using multi-threading applications. The usage of the dual-core feature to process two streams of data simultaneously aids in increasing speed and efficiency. Specifically developed to incorporate power efficiency the Cortex-A9 contains several elements which can effectively make the processor power consumption friendly, ideal for use in embedded systems. The ARM Cortex-A9 features SIMD technology, which are Single Instruction, Multiple Data instructions that enhance Multimedia and Signal Processing works. This is particularly useful in tasks such as audio, video, or graphic processing where the processor can handle more complicated data algorithms[11].

In addition to the CPU, PS contains memory presented in flash memory to store instructions(firmware), SRAM to store data in runtime, cache memory to optimise the execution time. It contains as well, timers and circuit to interconnect with the I/O peripherals. And other circuits can be found in [10] zynq-7000 reference manual.

2.2.2.3 In-Soc interconnection: AXI-bus protocol

Since zynq-7000 is a heterogeneous embedded processing platform that includes the software programming ability of a powerful ARM processor (PS) together with the hardware programming ability of an FPGA (PL) and other features, a very fast connection is established between the parts.

This connection is implemented using the AXI-bus protocol. The AXI or Advanced eXtensible Interface is a high-performance, high-frequency bus protocol developed by ARM as a part of Advanced Microcontroller Bus Architecture (AMBA)[12]. AXI provides a robust framework for high-bandwidth and low-latency data transfer,for more details and features about the AXI check [12].

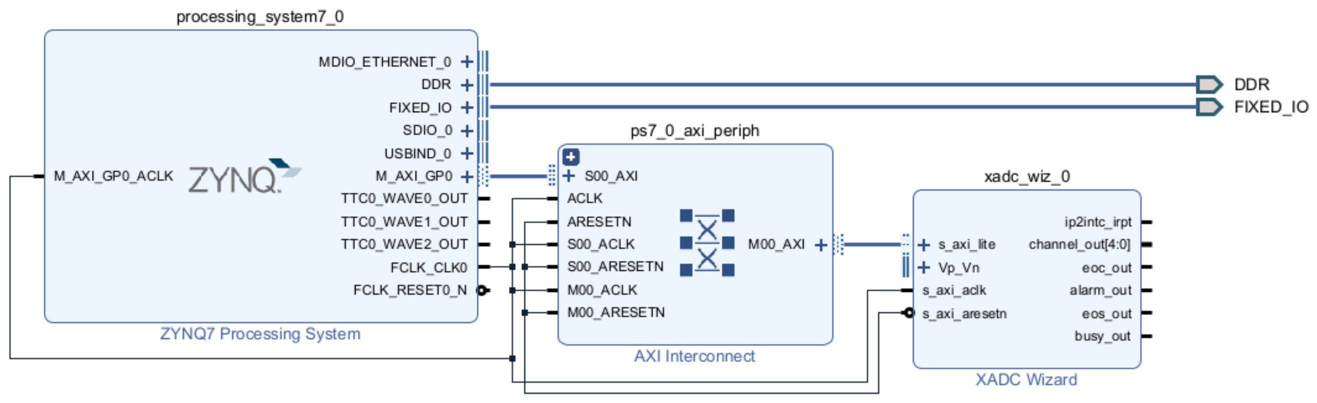


Figure 2.4: Interconnection between PS and XDC using AXI

The figure 2.4 shows the the interconnection using the AXI-bus protocols.

2.2.2.4 Xilinix Analog to Digital Converter

The XADC is an analog-digital converter part of the Zynq-7000 chip present on the Zedboard. The features and capabilities of the XADC will be outlined in this part as follows. The XADC is a dual 12-bit ADC that have 1 volt as maximum voltage reference[10]. On the XADC, there are multiple different channels that can be used to start conversions. They can be divided into two groups: analog input and internal sensors, the internal sensors that can be used are temperature sensor and six power supply sensors(each one measure a different input voltage), and 16 other general analog inputs. The XADC has more than one sampling configuration, default, continuous, single pass.. [13] The scope of this study deals with continuous mode XADC using temperature channel to monitor the temperature.

2.2.3 On Board Peripherals

The ZedBoard integrates various on-board peripherals for versatile embedded systems development:

1. **Switches**: ZedBoard includes 8 slide switches for user input or configuration purposes.
2. **user LEDs**: There are 8 LEDs on the ZedBoard for visual feedback and status indication.
3. **PMOD Connectors**: ZedBoard includes PMOD connectors, which are standardized connectors for interfacing with a variety of peripheral modules such as sensors, communication modules, etc.
4. **User Push Buttons**: ZedBoard comes with 5 push buttons for user input or control.
5. **JTAG Connector**: ZedBoard includes a JTAG connector for programming and debugging purposes.

6. **USB-UART Interface**: ZedBoard features a USB-UART interface for serial communication and debugging. The USB-UART is wired to the UART1 of the PS of the zynq-7000.

7. **Expansion Headers**: ZedBoard includes various expansion headers that expose additional interfaces like GPIO, SPI, I2C, XADC analog inputs etc., allowing you to connect custom peripherals or expansion boards.

These on-board peripherals provide easy access to common interfacing components, making ZedBoard a convenient platform for embedded systems development.

2.3 AMD Design Suite: Vivado & Vitis

Development environment kits can be viewed as essential tools for both software and hardware development initiatives. These kits, as seen with devices such as AMD's Vivado and Vitis platforms for boards such as the ZedBoard, are essential elements in the development cycle. To software engineers, these kits provide IDEs with debugging tools, profiling utilities and optimize workflows hence increasing efficiency and productivity. On the hardware front, development environment kits furnish designers with synthesis, and implementation tools essential for realizing custom hardware designs. They facilitate verification and validation processes, ensuring compliance with functional and timing requirements. Furthermore, these kits introduce IP integration, allowing designers to seamlessly incorporate pre-designed IP cores into their hardware designs. By combining software and hardware development kits, altogether encourage collaboration and synergy between software and hardware teams and enhance innovation in embedded system design.

2.3.1 Vivado

The Vivado Design Suite by AMD is a hardware development environment for the developing FPGA systems containing a wide variety of tools and features that make it suitable for creating, verifying, and implementing complex circuits in FPGA and SoC technologies by Xilinx. Vivado includes a variety of tools and elements to help engineers at every stage of the design process, from architecture and logic synthesis to implementation, testing, and analysis and even exporting hardware. As the result, Vivado offers a simple graphical user interface and wide range tools, which allows engineers to deliver the most of FPGAs and SoCs, and to create fast and optimized embedded systems for various application in automotive, aerospace, telecommunication industries and others[14].

Vivado has a rich list of features and capabilities aimed at improving development of embedded systems with FPGA and SoC technologies. Vivado's IP integrator facilitates integration of Xilinx IP cores and custom IP interfaces on a system level design for better integration and ease of configuration. As for the debugging and verification instruments, it includes the wide variety of the features of the Vivado Logic Analyzer that allows for the real-time examination of the sorting signals for the debugging of the hardware. Vivado has the ability to export the generated hardware as form of a file so software development kits (sdk) use it as a base hardware for applications. Since Vivado is a comprehensive tool, it also offers capabilities to address time analysis and power issues that projects may have. Built around an efficient graphical user interface, the tool is a versatile environment created with an eye for the newcomer in design as well as a powerful environment for complex designs that professional designers will find vital to the creation of today's hardware platforms[14].

2.3.2 Vitis

The Vitis Software Development Kit (SDK) is an advanced development environment provided by AMD, designed to facilitate the creation, analysis, and optimization of software applications for SoC systems. Vitis SDK integrates seamlessly with Xilinx hardware, allowing developers to program, debug, and profile software on a variety of platforms including Xilinx Zynq® SoCs, MPSoCs, and Versal ACAPs. Key features of Vitis SDK include support for heterogeneous computing, enabling the use of both CPUs and programmable logic (FPGA circuits) for efficient task execution. It offers a comprehensive set of libraries and APIs tailored for FPGA acceleration, and an intuitive GUI as well as command-line interfaces for versatile development workflows. Additionally, Vitis SDK provides advanced debugging and profiling tools, which help in identifying performance bottlenecks and optimizing both software and hardware components. With its robust support for high-level synthesis (HLS) and direct integration with Vivado Design Suite, Vitis SDK streamlines the development process, ensuring high performance and power efficiency in embedded applications[15].

2.4 Renode

Renode is an open-source framework developed by Antmicro, with the main purpose of emulating embedded systems, prototyping, validation, and debugging. With the help of this software, developers can model and emulate entire hardware systems, from individual microcontrollers to interconnected IoT devices, peripherals, sensors, environments, and wired or wireless communication between nodes, on a host computer. This makes it possible to test and verify the functionalities of software in a emulated hardware and can be repeated several times without involving physical platforms. Renode simulates various architectures and peripherals, which helps in getting deep visibility of systems and makes any problem easily identifiable at an early stage. This tool improves the productivity and effectiveness of embedded systems in line with standard development tools and CI/CD [16].

It is noteworthy that Renode can be used in various domains and applications, and thus it is becoming important to the embedded system development and testing field. One of the most important uses is in developing IoT devices, where it allows the emulation of all interconnected devices, so that complete testing of communication protocols and general network behavior can be done without requiring several physical devices. Renode can be used in industrial automation to design and test the performance and reliability of control systems within the industrial field. Another important area in which it is applied is the academic and research field, where design can take place within a flexible environment that enables new solutions to be tested on their architectures. Also, the possibility to run unmodified binaries is useful in continuous integration and continuous delivery (CI/CD) where updated pieces of software have to be tested automatically on different hardware platforms. Altogether, due to its flexibility and the availability of almost all essential features, Renode can be used at any stage of development, from initial prototyping to extensive product deployment and servicing [17].

When it comes to how it works, Renode treats any piece of hardware as a C# object, defining methods for that object to behave the same as real hardware. It classifies the peripherals into three categories: 8, 16, and 32-bit peripherals to make the connection to the system's bus easier. Sensors, DACs, and ADCs are emulated using functions that simulate the input and output of these devices [7].

2.5 FreeRTOS

FreeRTOS is an open-source real-time operating system (RTOS) kernel specifically developed for microcontrollers used in embedded systems for concurrent monitoring and execution of multiple tasks. Its functionality includes preemptive and cooperative multitasking, task prioritization, and dynamic memory allocation. FreeRTOS includes support for task management, scheduling, as well as methods of passing information and coordinating between tasks: semaphores, mutexes, queues etc. These features assist in coordinating tasks and keeping data synchronous throughout different functions. The other benefit of FreeRTOS is that it also supports an idle mode which is appropriate for low power applications like battery operated devices[18]. One of the fundamental features of FreeRTOS is modularity and the possibility of integration. It can be ported to a variety of microcontroller architectures such as MSP432, ZYNQ-7000 PS, arduino and many more. Such flexibility means that developers can employ FreeRTOS in a wide variety of applications without having to limit themselves by a particular type of hardware platform[19]. Additionally, FreeRTOS is accompanied by extensive documentation and a strong community, which means a wide range of resources for developers[20]. Due to the strong foundation and the continuous updates and support for FreeRTOS, FreeRTOS will always continue to be a stable and optimal choice for real-time applications in embedded systems. Due to the incorporation of cloud connectivity features in FreeRTOS, it also helps to create IoT applications and guarantees secure and effective connection for numerous IoT devices[21].

2.6 Summary:

The chapter is the overview of the Hardware and Software, initial concepts that will help the reader approach the following topics. First we have seen the Zedboard Development kit's major points and the programmable component, Zynq-7000 SoC, along with the Processing system, and interconnect within the SoC using AXI-bus protocol. Also, it discusses the Xilinx Analog to Digital Converter and on board peripheral devices. The subsequent section of the chapter describes the tools that are included in the AMD Design Suite: Vivado and Vitis as well as their purposes. Additionally, it brings into discussion Renode, an open-source emulation platform designed for embedded systems. In the next chapter we will design and implement a single node embedded system using Zedboard.

Chapter 3

Design and Implementation of a Single-Node System

3.1 Introduction:

In this chapter, we step into the action to set up and build a single-node system with the help of an exciting-to be in environment, known as the Zedboard. Moving on from the theoretical concepts outlined earlier in the report, where we introduced the reader to the capabilities of the Zedboard hardware, as well as the tools represented in the Vivado and Vitis design suites and the context of the FreeRTOS open source real-time operating system, this chapter will detail the practical approach to build a functioning single-node system.

We will start with the simplest configuration, one physical node, and develop techniques for emulating multi-node system in subsequent chapter. As a starting point, the single-node system will set the context for the difficulties and issues faced in the design of embedded systems prior to expanding to a wider multiple-node environment.

As a practical application, our single-node system will monitor the temperature using the XADC (Analog-to-Digital Converter) and the on-chip temperature sensor of the ZYNQ-7000.

lastly, we will present findings of our work, by showing the results to illustrate performance of the proposed system and describe the problems that were identified during the process of implementation. After reading through this chapter, the reader will have grasped the ways and means of designing, programming, and validating a single-node system using to the Zedboard, Vivado, Vitis, FreeRTOS.

This knowledge will then lay the groundwork for the next chapter on multi-node systems where the system has more than just one node. All of these skills and understanding are going to be beneficial during our further work, aimed at the design and emulation of multi-node systems, which can improve the functionality and scalability of embedded systems.

3.2 Hardware Part

In this section we introduce the design constrains and the implementing steps for the hardware part to configure the Zedbaord to work as a single-node embedded system that monitor the internal temperature of the zynq-7000 and send it to PC via usb-uart port. At the end of this section we obtain a generated file to load it to the Zedboard in order to configure it.

3.2.1 Design Considerations

In order to achieve the needed hardware, we mainly need four parts: the processing systems of the zynq-7000, the XADC IP, the AXI interconnect IP and processor system reset IP. The objective of the design is to configure and connect the mentioned IPs in such way they work together.

The processing system will be the central component responsible for controlling and processing information in the entire system. The XADC IP performs the sampling and converting of the analog signal, so that the system is able to interface with the on-chip temperature sensor which is connected directly to it. This is important in the monitoring of temperature of the Soc. The AXI interconnect IP provides a medium of communication between the processing system and the various IP cores that are present in the design. It works like an efficient and fast highway that allows the data exchange between the components. The processor system reset IP will control the reset signals that are present in the design and will co-ordinate the resets of the system when needed.

When these four essential blocks are appropriately configured and interconnected, the end hardware architecture will provide the performance and functionality to the intended application demands. The processing system will manage the flow and control of data, the XADC will perform the analog interfacing, the AXI interconnect will facilitate connection between them and the reset IP will manage resets for the system.

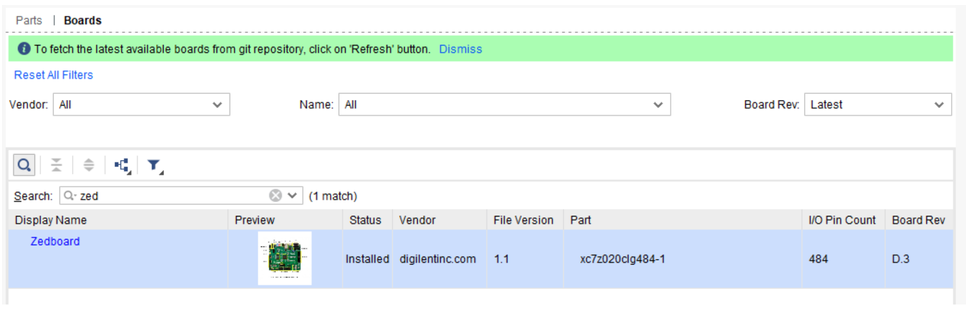
3.2.2 Implementation

In this section, we will discuss the steps and methods used to implement the hardware needed. This includes the setup of the environment, the configuration of the necessary blocks and IP cores, and at the end obtain the intended hardware platform.

Firstly, we setup the development environment. This involves installing Vivado and Vitis development suite, we used the 2023.1 version. The installation and the setup of the environment is straight forward. A boards file for the Zedboard needs to be downloaded from [22], and paste it in the path bellow in order to get known by Vivado.

```
vivado_install_path\Xilinx\Vivado\20xx.x\data\boards\board_files
```

Subsequently, we launch the the Vivado design software and start creating new project. In this step we are going to choose ”Zedboard” as platform this should help us in further steps by letting vivado knows the target platform.



Parts | Boards

To fetch the latest available boards from git repository, click on 'Refresh' button. Dismiss

Reset All Filters

Vendor: All   Name: All   Board Rev: Latest

Search: zed (1 match)

Display Name	Preview	Status	Vendor	File Version	Part	I/O Pin Count	Board Rev
Zedboard		Installed	digilentinc.com	1.1	xc7z020clg484-1	484	D.3

Figure 3.1: including zedboard in the project

After the creation of the project, we tested the functionality of the available board by writing a VHDL code to test switchs and leds. The process is as follows, first, in the project manager we add sources then choose VHDL file. The second step is we wrote a simple VHDL code to implement some logic functions. Next is an example of a VHDL code.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity Test_hardware is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           c : in STD_LOGIC;
           d : out STD_LOGIC;
           e : out STD_LOGIC);
end Test_hardware;

architecture Behavioral of Test_hardware is

begin
d <= a and b;
e <= (a or not b) xor c;

end Behavioral;
```

After writing the VHDL code, the next step is to assign the SoC pins to the switches and LEDs available on the ZedBoard. This is done by creating a constraints file (XDC file) that includes the necessary pin assignments. Xilinx provides a constraints file with all possible assignments; we just need to use three switches and two LEDs. Finally, we generate the bitstream to load it onto the board. This process ensures that your VHDL design interacts correctly with the physical switches and LEDs on the ZedBoard.

Following successful testing, we transitioned to the hardware implementation phase. Beginning in the project manager, we selected ”Create block design” to lay the foundation for our project. Within this design, we integrated essential components the ”ZYNQ processing system” which is the PS part. Additionally, we incorporated the ”XADC wizard”. The next step is the configuration of the XADC IP by double clicking on the block. Then we select continuous operating mode to monitor the temperature, and choosing the temperature analog sensor, select the channel sequencer and disable the alaram outputs. This step can be done by software when accessing the xadc control register, but it easier to do it in vivado graphically. After that we run the ”Block automation” feature which is provided with vivado to configure the processing system automatically and connects it with external I/O and memory, its noteworthy to disable unneeded features like TTC timers and communication ports and so on. Vivado provides a feature called ”RUN CONNECTION AUTOMATION”, which interconnects the IP cores automatically by adding AXI-bus and processor system reset IP cores.

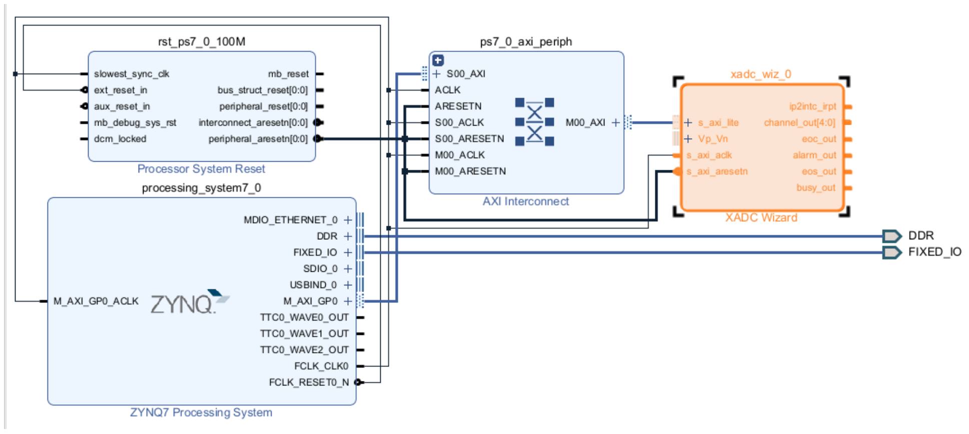


Figure 3.2: project block diagram

The figure 3.2 shows the final block diagram for the hardware part.

3.3 Software part

After we have successfully complete the hardware design and implementation phase for this project we now move to the next phase, which is the software development phase. This phase will use Vitis software development kits (SDK) and FreeRTOS. These tools will help us to develop a good and efficient software that takes the full advantage of hardware features that we have developed early on.

First of all, after opening Vitis SDK and the creation of the work space, the creation of a hardware platform is the needed step to be done. This step can be done by importing the hardware file(.xsa) generated earlier and build the platform. This operation automates the process of setting up the software development environment with the correct hardware settings and peripherals.

Following that, we create an example application from the templates. That will include the needed source and header files automatically without the need to import them manually. As we are working with freeRTOS, we create the ”FreeRTOS_hello_world” application and associated with the hardware platform created previously.

The objective of the software code is to implement two freeRTOS tasks, the first task reads the temperature from the sensor using the XADC and send it to a queue periodically. The second task should read form the queue and sends the data through UART port.

The following is breakdown of the code:

- Header files:

Listing 3.1: header files

```c

#include <stdio.h> // Standard I/O functions
#include "xparameters.h" // Xilinx hardware parameters generated by Vitis
   to initialize the platform
#include "xadcps.h" // XADC driver header
#include "xuartps.h"// UART driver header
#include "xil_printf.h" // Xilinx-specific printf
#include "FreeRTOS.h" // FreeRTOS header
#include "task.h" // FreeRTOS task management
#include "queue.h" // FreeRTOS queue management
```

- Definitions and prototypes:

Listing 3.2: Definitions and prototypes:

```c

// Definitions for XADC to match the xparamaters header
#define XADC_DEVICE_ID XPAR_PS7_XADC_0_DEVICE_ID

// Task parameters
#define TEMP_MONITOR_TASK_PRIORITY (tskIDLE_PRIORITY + 1)
#define TEMP_PRINT_TASK_PRIORITY (tskIDLE_PRIORITY + 1)
#define TASK_STACK_SIZE (configMINIMAL_STACK_SIZE )

// Queue length
#define QUEUE_LENGTH 10

// Function prototypes
void vTempMonitorTask(void *pvParameters);
void vTempSenderTask(void *pvParameters);

// Global variable for the queue handle
QueueHandle_t xTempQueue;
```

- XADC configurations:

Listing 3.3: XADC config

```c
// Configure the XADC sequencer

    // Set the sequencer mode to safe to prioritizes safety and reliability
         in the sampling process
    XAdcPs_SetSequencerMode(&XADC_Driver_Instance, XADCPS_SEQ_MODE_SAFE);

    // Enable the temperature channel
    XAdcPs_SetSeqChEnables(&XADC_Driver_Instance, XADCPS_SEQ_CH_TEMP);

    // Set the input mode for the temperature channel
    XAdcPs_SetSeqInputMode(&XADC_Driver_Instance, XADCPS_SEQ_CH_TEMP);

    // Set the sequencer mode to continuous passing
    XAdcPs_SetSequencerMode(&XADC_Driver_Instance,
       XADCPS_SEQ_MODE_CONTINPASS);
```

- Tasks implementations:

Listing 3.4: tempearture monitoring task

```
   void vTempMonitorTask(void *pvParameters) {
    u16 temp_raw;
    int temp_f;

    // Temperature monitoring loop
    while (1) {
        temp_raw = XAdcPs_GetAdcData(&XADC_Driver_Instance,
           XADCPS_CH_TEMP); //Read temp from XDAC FIFO

        temp_f = XAdcPs_RawToTemperature(temp_raw);// convert the
           readings to meaningful temperature unit by applying
           calibration factors provided by Xilinx.

        // Send temperature to the queue
        if (xQueueSend(xTempQueue, &temp_f, portMAX_DELAY) != pdPASS)
           {
            xil_printf("Failed. Queue is full !\n");
        }// check for queue before sending.

        // Delay for a period
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}
```

Listing 3.5: Temperature transmissiom task

```c
void vTempSenderTask(void *pvParameters) {
    int received_temp;

    // Temperature print loop
    while (1) {
        // Receive temperature from the queue
        if (xQueueReceive(xTempQueue, &received_temp, portMAX_DELAY)
            == pdPASS) {

         XUartPs_Send(&UART_Instance, (u8*)received_tempr, sizeof(
             tx_buffer));
             xil_printf("Temp is: %d C\n", received_temp);
        }
    }
}
```

- main function:

Listing 3.6: Main function

```c
    int main() {
    // Create the queue
    xTempQueue = xQueueCreate(QUEUE_LENGTH, sizeof(int));

    // Create tasks
    xTaskCreate(vTempMonitorTask,
                "Temp Monitor Task",
                TASK_STACK_SIZE,
                NULL,
                TEMP_MONITOR_TASK_PRIORITY,
                NULL);

    xTaskCreate(vTempSenderTask,
                "Temp Sender Task",
                 TASK_STACK_SIZE,
                 NULL,
                 TEMP_PRINT_TASK_PRIORITY,
                 NULL);

    // Start the scheduler
    vTaskStartScheduler();

    // the scheduler is running and will never return here.

    return 0;
}
```

At this level, both software and hardware parts for the single node implementation of the embedded system are completely set. Well, as we all know, when things have been set up, the next thing that is supposed to be done is to determine how the system functions. In the proceeding section, an explanation of the results achieved from the setup described above will be discussed. The aim of testing and however briefing is to analyze the behavior of the system, evaluate it to determine whether or not it is functioning as expected and, in general, to determine if it meets the requirements that have been set out.

3.4 Results & Discussion

Now let us turn to the Results and Discussion presented in the following section. Here, the results of the completed installation and configuration of software and hardware will be discussed. We shall be able to review how efficient our designed system is, discover the difficulties faced while implementing the system and consider the prospects for change.

3.4.1 Results

Before obtaining the result, a UART-USB communication needs to be configured. By using third party emulation programs that provide an interface for connecting to remote systems via various communication protocols such as serial. We setup ”Tera term” application to communicate with serial and we set the baudrate to 115200, same as baudrate configured in the Zedboard UART.

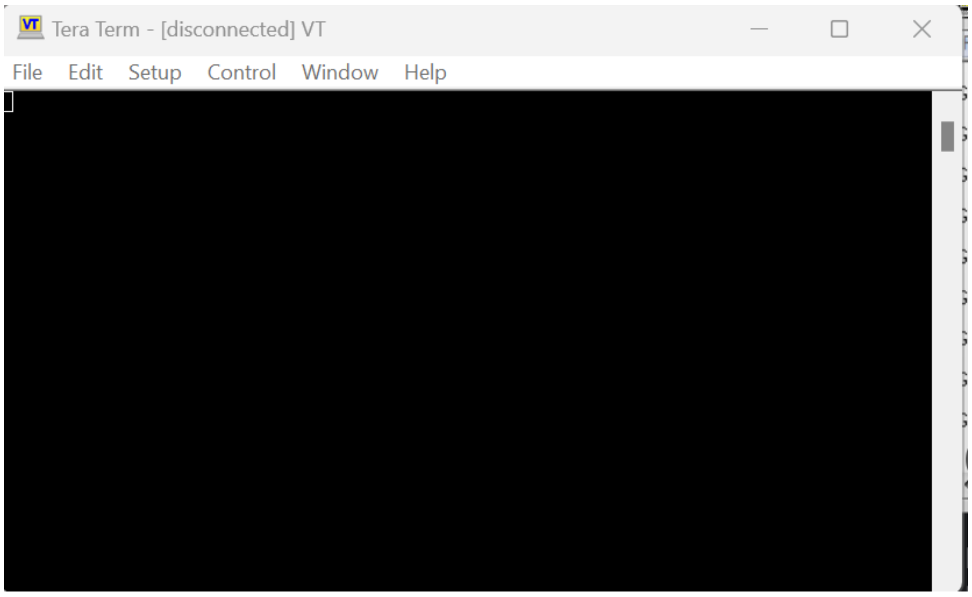
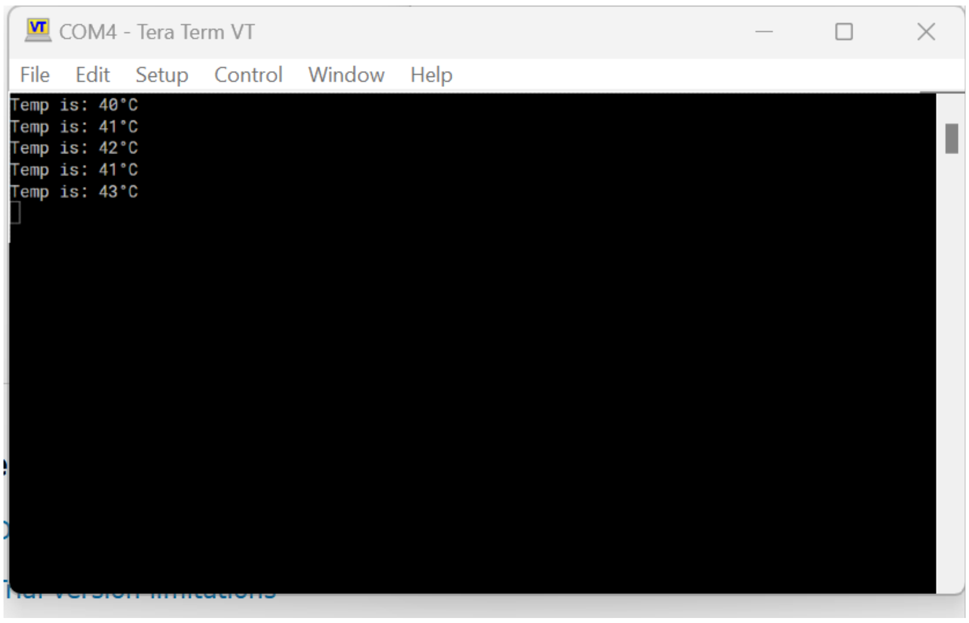


Figure 3.3: Tera term -Not connected-

We connect the Zedboard to the PC using ”JTAG Programing” port for the programming, and ”USB-UART” port for sending the data and run the application via Vitis. The communication starts and we get the following result:



```
Temp is: 40°C
Temp is: 41°C
Temp is: 42°C
Temp is: 41°C
Temp is: 43°C
```

Figure 3.4: UART output via Tera term

Task 2 is sending the temperature values via UART and displayed using Tera term.

3.4.2 Discussion

After letting the the temperature measurement run for a period we notice that temperature is slowly increasing then it settle. This is due to the heat sink which transfer the heat from the SoC to the environment making the it cooler [10].

3.5 Summary:

This chapter shifted from theoretical discussions to actual practice by implementing a single-node system with help of Zedboard hardware. Our objective was to offer a more practical perspective on the topic of embedded systems design. This initial implementation sets the stage for the exploration of multi-node systems covered in the following chapter. The practical application of the concepts that we take is illustrated through the experiment where we used XADC and on-chip sensor to measure temperature. In the process. By now, readers learned about designing, implementing, and validation of the single-node systems on a Zedboard using the Vivado, Vitis, and FreeRTOS tools. This knowledge helps them when it comes to designing and emulating multi-node systems hence improving on the functionality of embedded systems and scalability.

Chapter 4

Emulation and Testing The Multi-Node Systems

4.1 Introduction:

Expanding the work done in the previous chapter focused on the design and implementation of a single-node system and its results and discussions, this chapter goes deeper into the discussion about emulation and testing of multi-node systems. With the help of the Renode framework that has been mentioned earlier, it is possible to dive deeper into the description of various interconnected systems and show the work of their components as functions of the overall system. Multi node systems, which define an architecture with interrelated individual nodes, usually, present challenges and issues when it comes to design and implementation especially in communication and synchronization.

Both emulation and testing play significant roles to guarantee that there would be no problems with the network's communication, coordination, and functionality before the real implementation. In this chapter, we will discuss how to accomplish accurate emulation of a multi-node embedded system, leverage capabilities of Renode for developing successful test plans. By the end of this chapter, you will be able to use Renode to emulate different embedded systems, boards, peripherals and connection between them efficiently.

4.2 Renode basics

4.2.1 Renode Capabilities

Renode is a command line interface (CLI) application. It gives the user access to emulation objects (devices) and reveals a few basic commands. The Monitor allows the user to view and alter the states of those objects as well as carry out the actions that they provide. Renode also have a built-in logger system that log lots of information about the operation of the emulated environment in the logger window.This feature is very helpful when it comes to debugging.

Typically, the user would begin by setting up, configuring, and connecting the appropriate emulated (guest) platform or platforms (referred to as "machines") using a series of instructions. This can be done automatically using nested Renode scripts (.resc) which encapsulate some of the repeatable elements in this activity (normally, the user will want to create the same platform over and over again in between runs, or even script the execution entirely to test).[7]

In addition, Renode supports a broad range of hardware systems (Zedboard is supported), including different CPU varieties, architectures, and I/O capabilities, sensors and so on. Furthermore, for non-supported or user-created devices, users need to create their own peripheral models. This would involve writing the necessary code (in C# or python as mentioned earlier in Chapter 2) to define the behavior of the new device within the Renode framework. Once the peripheral model is created, it can be integrated into the Renode emulation environment.

4.2.2 Emulation Setup and Configuration

When beginning emulation in Renode there are several initialization steps that need to be performed in order to properly start the emulation and load all the necessary software. Manually it becomes very large to carry and time consuming, especially if many changes are made in the layout. To make these steps more efficient and repeatable, it can be saved as a Renode script (.resc). It is also worth mentioning that this script contains all the necessary commands to create and configure an emulation environment, allowing you to include this script in the Renode monitor. By using the (.resc) script, you can ensure that the emulation setup is consistent and reproducible, reducing the risk of errors that might occur with manual configuration.

Firstly, we will start with creating a single node emulation setup to gain a clearer understanding of the steps involved and to ensure that all is well as a first check. Finally, once the single-node setup is up and running, we will extend the script to add another node to emulate interconnected node system.

4.3 Single-node emulation

First of all we create a platform to begin with, this can be done using ”mach create” command followed by a label to give it a specific name. After that, we load the ”zedboard.repl” which is a file provided by Renode containing description of the emulated zedboard. The command essential to that is ”machine LoadPlatformDescription” followed by the path location of zedboard.repl. For the next step we load the (.elf) file which is the executable file generated after compiling and linking the software by Vitis. By using ”sysbus LoadELF @/path/of/genereted/-file/output.elf”. subsequently, we use command ”showAnalyzer uart” to interact with the zedboard’s uart port. Last but not least, we add the ”start” command to begin the emulation.

Listing 4.1: Renode Script for a single-node Zedboard

```
:name: Zedboard
:description: This script configures a single-node Zedboard.

$name?="Zedboard_UART_echo_test"
mach create $name

machine LoadPlatformDescription @platforms/boards/zedboard.repl

showAnalyzer uart1


$bin?=@path/to/your/binary/file.elf

```

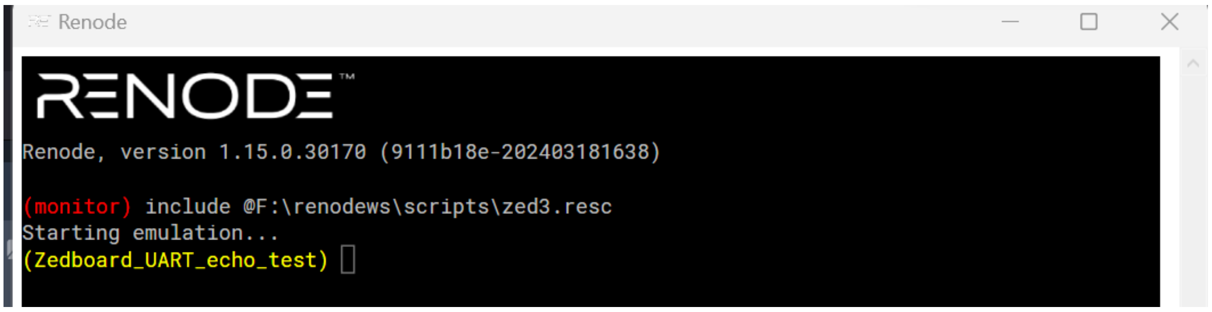
```
## load binaries ##
sysbus LoadELF $bin
start
```

The following algorithm represents the software (.elf) given above, it initialize UART then reads and echo-back characters to test the UART functionality on emulation.

Algorithm 1 UART echo loop

```
1: init UART
2: while true do
3:     if Receiver flag is set then
4:         $received\_char \leftarrow$ UART FIFO                ▷ receive the char
5:         UART FIFO $\leftarrow$ received_char                ▷ echo back the char
6:         if $received\_char =' Enter'$ then                  ▷ Enter key
7:             go to the next line
8:         else if $received\_char =' Backspace'$ then         ▷ Backspace key
9:             delete the character
10:        end if
11:    end if
12: end while
```

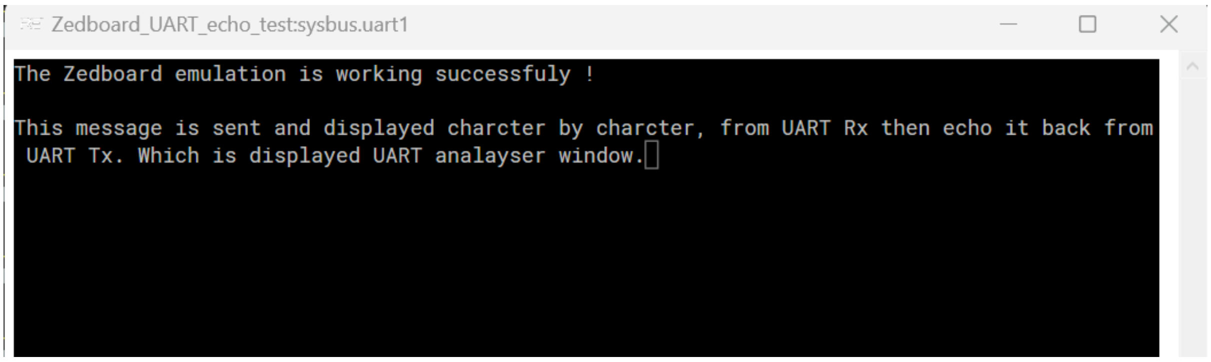
After those steps are done, in the Renode monitor we use ”include” command to include our script.



```
Renode, version 1.15.0.30170 (9111b18e-202403181638)

(monitor) include @F:\renodews\scripts\zed3.resc
Starting emulation...
(Zedboard_UART_echo_test)
```

Figure 4.1: The lunching of emulation

Now we can pause the emulation and debug or test many scenarios use debug commands provided by Renode[7]. The emulation output can be showed in the UART analyser as mentioned before.



Zedboard_UART_echo_test:sysbus.uart1

```
The Zedboard emulation is working successfuly !

This message is sent and displayed charcter by charcter, from UART Rx then echo it back from
 UART Tx. Which is displayed UART analayser window.
```

Figure 4.2: Emulation output

In order to confirm the emulation reliability, the same test has been done on a physical hardware(Zedboard). The behavior of the both systems (physical and emulated) was identical. So, after the single-node emulation analysis and receiving positive outcomes, we should proceed with the further research and move to the two-node emulation analysis. With this scaling, we will be able to study the behavior of nodes and their communication, giving an idea about Renode scaling capabilities.

4.4 Multi-node emulation

Based on the previous single-node emulation, this section will introduce multi-node emulation and their interconnection. To begin with, let's take the script 4.2 of creating a single node and build on it. It's as simple as duplicate each command with paying attention to some changes, hardware labels (names) and software paths. This can be done using the ”mach set” command to select the platform then load each one individually. In addition to that we need to emulate a communication between the two nodes, we use the ”UARThub” to do that. this is the script:

Listing 4.2: Renode Script for a double-node Zedboard

```
:name: Zedboard
:description: This script configures two nodes of Zedboard.

$name1?="Zed1"
$name2?="Zed2"
$bin1?=@path/to/your/binary/file1.elf
$bin2?=@path/to/your/binary/file2.elf

mach create $name1
mach create $name2

emulation CreateUARTHub "uartHub" ## create the uarthub
```

```
mach set Zed1
machine LoadPlatformDescription @platforms/boards/zedboard.repl
showAnalyzer uart1
sysbus LoadELF $bin1
connector Connect sysbus.uart1 uartHub ##connect the uart to the
   uart hub

mach set Zed2
machine LoadPlatformDescription @platforms/boards/zedboard.repl
showAnalyzer uart1
sysbus LoadELF $bin2
connector Connect sysbus.uart1 uartHub ##connect the uart to the
   uart hub

uartHub Start
start
```

The topology given by the script in listing 4.2 is described as follows:

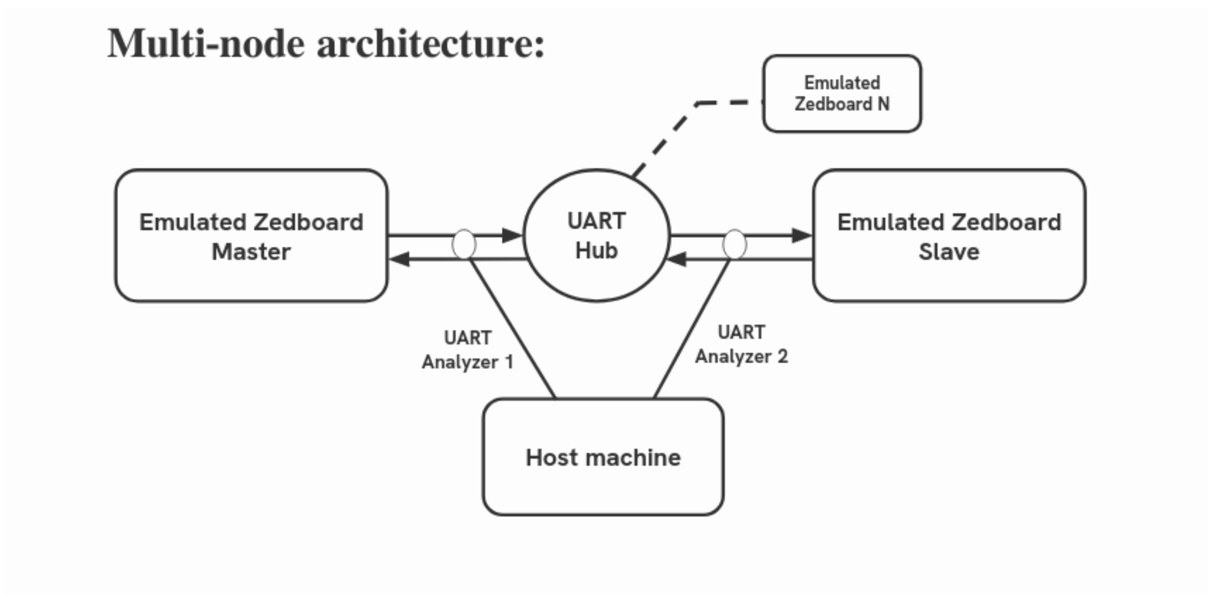


Figure 4.3: Multi-node architecture approach

The figure 4.3 shows two nodes a master and a slave, as well as a communication between them using the UARThub mentioned in the listing 4.2.

The goal of this emulation is to monitor the temperature in the slave node and send it to be stored and displayed by the master node. Hence, two firmware's has been needed.

For the slave node, we developed two FreeRTOS tasks: vTasktemp and vTasktransmit. The vTasktemp task is responsible for monitoring temperature values. It emulates reading these values by utilizing a function that simulates a temperature sensor, as described earlier. The vTasktransmit task handles the transmission of these temperature readings to the master node via UART port.

Listing 4.3: slave node software pseudo-code

```
// Define functions
- uart_init(baud_rate, clock_frequency) // Initialize UART with baud
rate and clock frequency
- uart_send(message) // Send a message via UART
- readTemperatureSensor() // Simulate reading temperature with
realistic variation
-FreeRTOS task functions

//In the vTasktemp task:
- Read temperature using readTemperatureSensor function
- Inqueue the the readings in FreeRTOS queue

//IN vTasktransmit task:
-Dequeue the temperature from the FreeRTOS queue
-Send the message with temperature via UART using uart_send function
- Delay for a period

// In the main function:
- Initialize clock frequency and baud rate
- Initialize UART
- Initialize a the FreeRTOS queue and create the tasks
- start the scheduler
```

For the master node, we developed two FreeRTOS tasks: vTaskReceive and vTaskDisplay. The vTaskReceive task is responsible for reading data from the UART port and storing it in a buffer. Once the data is read, it queues the temperature readings into a FreeRTOS queue for further processing. The vTaskDisplay task retrieves the temperature values from the queue and displays them on the screen, ensuring that the master node continuously monitors and outputs the received temperature data in a timely manner. This setup allows for efficient data handling and real-time monitoring of temperature values sent from the slave node.

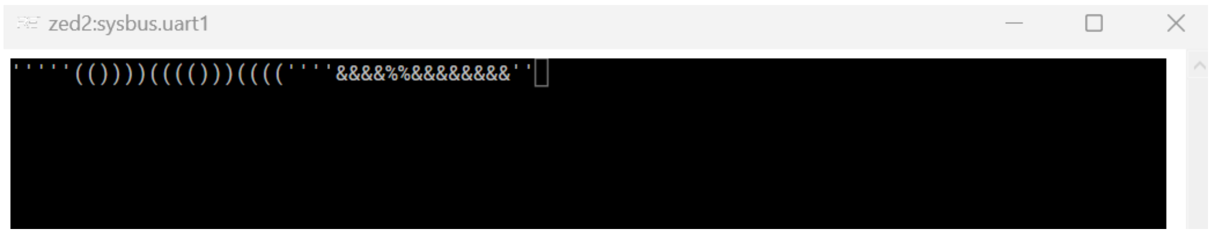
Listing 4.4: master node software pseudo-code

```
// Define functions


- intToString(value, str, base)
```

```
  // Convert an integer to a string
- uart_send(message)
  // Send a message via UART
- uart_receive()
// reads from the UART's FIFO
- FreeRTOS tasks functions

//In the vTaskReceive task:
- Check if a character is received
  - If received:
    - Read the received character
    - Store the received character
    - Enqueue the received character

//In the vTaskDisplay task:
    - Convert received character to string //using intToString fun
    - Send temperature received from the slave is: //via UART
    - Send the converted string via UART
  - Insert a delay to avoid busy waiting

// In the main function:
- Initialize clock frequency and baud rate
- Initialize UART
- Initialize a the FreeRTOS queue and create the tasks
- start the scheduler
```

After including the script and starting the emulation, this is the output we got from each UART.

- Slave node output:

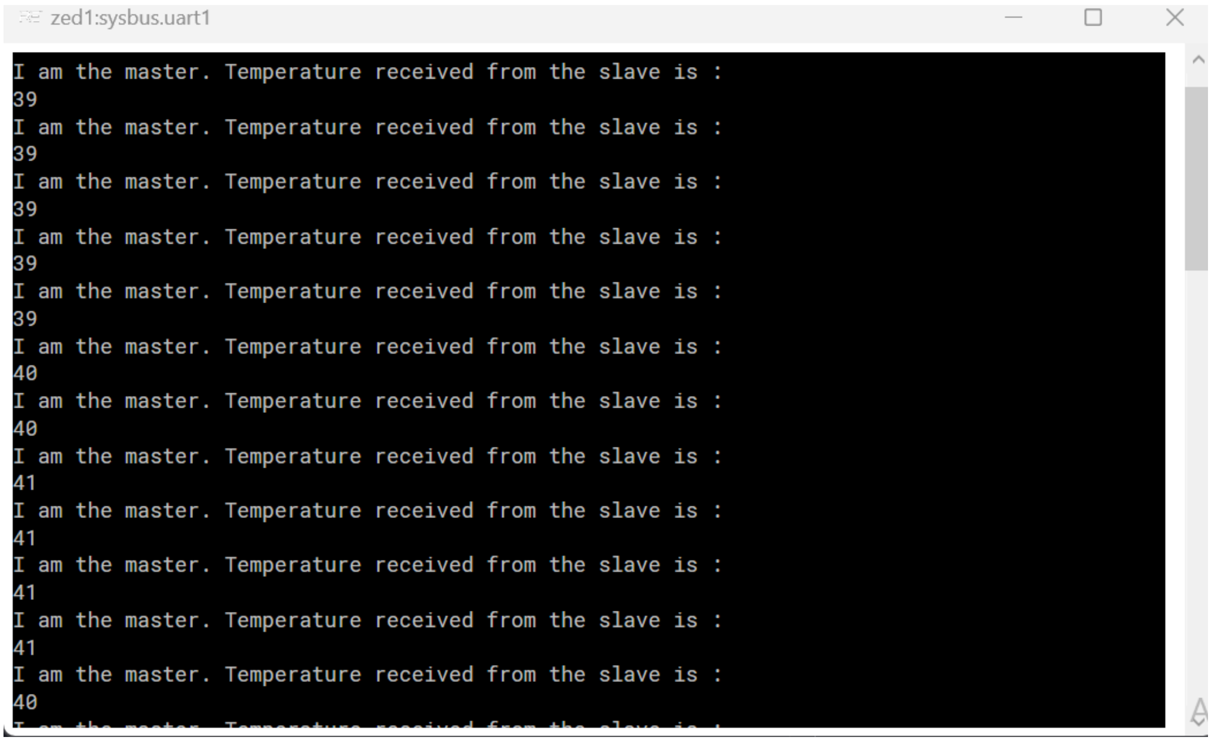


zed2:sysbus.uart1

```
'''''(())))(((()))(((('''' &&&&%%&&&&&&&&''
```

Figure 4.4: The slave node UART output

• Master node output:



```
I am the master. Temperature received from the slave is :
39
I am the master. Temperature received from the slave is :
39
I am the master. Temperature received from the slave is :
39
I am the master. Temperature received from the slave is :
39
I am the master. Temperature received from the slave is :
39
I am the master. Temperature received from the slave is :
40
I am the master. Temperature received from the slave is :
40
I am the master. Temperature received from the slave is :
41
I am the master. Temperature received from the slave is :
41
I am the master. Temperature received from the slave is :
41
I am the master. Temperature received from the slave is :
41
I am the master. Temperature received from the slave is :
40
```

Figure 4.5: The master node UART output

4.5 Discussion

After successfully emulating a single-node system and confirming the same output between the physical and emulated hardware, we moved on to multi-node emulation. At first look, emulation seemed unusual due to the output shown in Figure 4.4, which shows the slave's output. The characters appearing on the screen are actually integers, but the "UART analyzer" interprets them as characters and prints them accordingly. We suggest a solution for improvement of this emulation problem by creating an other analyzer that shows the frame transmitted as it is. Hence, making analysing and debugging UART transmissions easier. On the other hand, we see in in Figure 4.5 the temperature values displayed by the master after receiving them from the slave.

For the communication protocol, and since the Zedboard has the ability to communicate using Ethernet port, we tried to emulate the connection between the nodes using the TCP/IP protocol. Therefor, we faced many problems due to the complexity of the protocol implementing and lack of time; despite that, our application functions as intended, because it doesn't need a high speed communication.

4.6 Challenges and difficulties

We experienced some difficulties while emulating using Renode, because the process of its development since it's an open source software. The first challenge was the unavailability of strong documentation, which is an issue because Renode is often updated, and the materials which can be found on the Internet sometimes describe the tool features that are no longer relevant. This made it challenging to seek for more particular information on setting up the emulated environment, incorporating the required peripheral devices, and managing problems as they occurred. Further, we came across such cases where the base addresses of some peripherals did not match the expected addresses, making a difference with regards to the emulator and the actual hardware. This made it necessary to look at the peripheral details to determine any contradictions in the respective device documentation and configure or bypass different features to allow proper functionality of the emulated system.

4.7 Summary:

Thus, in the given chapter we learned that by using Renode, we can emulate embedded that guarantee effective communication, synchronization, and performance. Through practical examples and discussions, we have shown how to efficiently emulate different embedded systems, boards, peripherals, and their connections using Renode. We also suggested a solution to an issue occurred during the emulation. With these skills, you now have what it takes to address potential challenges of emulating multi-node systems using Renode while guaranteeing efficiency.

Conclusion

This work has established the need for emulation in embedded systems and IoT and the usefulness of Renode as a flexible emulation platform. The feature of emulating the full system along with its capability to support multiple nodes makes Renode a useful tool for prototyping and testing. The interactivity and the support for various components of the hardware make it provide flexibility in its use and integration in both learning and working environments.

In conclusion, it can be seen that the importance of emulation will grow in tandem with the advancements in embedded systems and the increasing deployment of these systems in various domains. Tools like Renode will play a vital role in facilitating efficient and effective development and testing processes, improving outcomes in this dynamic field.

In the future, we want to master emulation using Renode by implementing more complex communication protocols such as TCP/IP and wireless communications, design and use our user-peripheral and hardware and integrate it in the emulation, as well as test the capability of high nodes scaling. We wish that our work lays a ground for students and researchers to use emulation in academic purposes to advance more in the embedded systems field.

Bibliography

[1] Peter Marwedel. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*. 2010. ISBN: 9789400702561. DOI: 10.1007/978-94-007-0257-8.

[2] BAP Software. *What is Embedded Systems?* Accessed: 2024-06-10. 2023. URL: https://bap-software.net/en/knowledge/what-is-embedded-systems/.

[3] Asma Mushtaq. "What are the Challenges in Embedded Systems Design?" In: (2023). URL: https://eevibes.com/computing/introduction-to-computing/what-are-the-challenges-in-embedded-systems-design/.

[4] Varun G Menon et al. "An IoT-enabled intelligent automobile system for smart cities". In: *Internet of Things* 18 (2022), p. 100213. ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2020.100213. URL: https://www.sciencedirect.com/science/article/pii/S2542660520300494.

[5] Abraham A Clements et al. "HALucinator: Firmware Re-hosting Through Abstraction Layer Emulation". In: *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1201–1218. ISBN: 978-1-939133-17-5. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/clements.

[6] Mark Jonas and Drew Chambers. "The Use and Abuses of Emulation as a Pedagogical Practice". In: *Educational Theory* 67 (June 2017), pp. 241–263. DOI: 10.1111/edth.12246.

[7] *Renode Wiki*. https://renode.readthedocs.io/en/latest/. Accessed: 2024-06-09.

[8] AVENT. *Hardware Manual for [ZedBoard Getting Started Guide]*. AVENT. 2012. URL: https://www.avnet.com/wps/wcm/connect/onesite/7ae0f288-1cc5-4283-9e85-300c5401b680/GS-AES-Z7EV-7Z020-G-V7-1.pdf?MOD=AJPERES&CACHEID=ROOTWORKSPACE.Z18_NA5A1I41L0ICD0ABNDMDDG0000-7ae0f288-1cc5-4283-9e85-300c5401b680-nxyWIEs.

[9] .

[10] Xilinx Inc. *Zynq-7000 SoC Data Sheet: Overview*. 2018. URL: https://docs.amd.com/v/u/en-US/ds190-Zynq-7000-Overview.

[11] ARM webpage for cortex a9. URL: https://developer.arm.com/Processors/Cortex-A9.

[12] ARM Limited. *AMBA AXI and ACE Protocol Specification*. Accessed: 2024-06-03. 2024. URL: https://documentation-service.arm.com/static/651c285c15583d1bff97
token=.

[13] Pascal Engeler. *Using the Zynq-7000 XADC and signal pre-conditioning*. 2017. URL: https://ethz.ch/content/dam/ethz/special-interest/phys/quantum-electronics/tiqi-dam/documents/semester_theses/vacationthesis-Pascal_Engeler.pdf.

[14] *Vivado Overview*. https://www.xilinx.com/products/design-tools/vivado.html. Accessed: 2024-06-10. AMD.

[15] *Vitis Unified Software Platform*. https://www.xilinx.com/products/design-tools/vitis.html. Accessed: 2024-06-10. AMD.

[16] Antmicro. *Renode - Antmicro's Open Source Simulator for Complex Embedded Systems*. https://renode.io/cases/. Accessed: 2023-06-10. 2023.

[17] Antmicro. *Use Cases*. https://antmicro.com/platforms/renode/. Accessed: 2023-06-10. 2023.

[18] FreeRTOS. *FreeRTOS Functionality*. Accessed: 2024-06-10. 2024. URL: https://www.freertos.org/features.html.

[19] FreeRTOS. *FreeRTOS Portability*. Accessed: 2024-06-10. 2024. URL: https://www.freertos.org/RTOS_ports.html.

[20] FreeRTOS. *FreeRTOS Support and Documentation*. Accessed: 2024-06-10. 2024. URL: https://www.freertos.org/Documentation/RTOS_book.html.

[21] FreeRTOS. *FreeRTOS and AWS IoT*. Accessed: 2024-06-10. 2024. URL: https://www.freertos.org/iot-libraries.html.

[22] Avent. *Use Cases*. https://digilent.com/reference/software/vivado/board-files. Accessed: 2023-06-10. 2023.

People’s Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M’Hamed BOUGARA - Boumerdes

Institute of Electrical and Electronic Engineering

Department of Electronics

Authorization for Final Year Project Defense

Academic year: 2023/2024

The undersigned supervisor: **Dr. MAACHE Ahmed** authorizes the student(s):

HACHMANE Abderrahemane	Option:	Computer.
KHOUAS Aness Mohamed	Option:	Computer.

to defend his / her / **their** final year Master program project entitled:

Emulating Multi-node Emended Systems using Renode

during the [✓ June] ☐ September session.

Date: 11 / 06 / 2024

The Supervisor

[illegible]

The Department Head