Murat Can Özbasaran

ARM RDN2 and TSS Integration as a System-on-Chip Simulation

School of Technology
Master's Thesis in Technology
Master's Programme in Computing Sciences Sustainable and Autonomous Systems

Vaasa 2025

Contents

Figures

Abbreviations

ASTF	Arm Software Trace Format
ATF	Arm Trusted Firmware
CADI	Component Architecture Debug Interface
CPU	Central Processing Unit
CSR	Control and Status Register
DPU	Data Processing Unit
FM	Fast Models
FPGA	Field Programmable Gate Array
GIC	interrupt controller
GPUs	Graphical Processing Unit
HDLs	Hardware Description Languages
IC	Integrated Circuit
IP	Intellectual Property
KVM	Kernel-based Virtual Machine
MPSoCs	Multi Processor SoCs
MTI	Model Trace Interface
PCIe	Peripheral Component Interconnect Express
PMU	Performance Monitoring Unit
PV	Programmer's View
QEMU	Quick Emulator
RDN2	Reference Design Neoverse 2
RTL	Register Transfer Level
SAT	Boolean Satisfiability
SimGen	System Generator
SoC	System-on-Chip
TLM	Transaction Level Modelling
TSS	Target SoC Simulator
VHDL	Very High Speed Integrated Circuit

UNIVERSITY OF VAASA
School of Technology
Author: Murat Can Özbasaran
Title of the thesis: ARM RDN2 and TSS Integration as a System-on-Chip Simulation
Degree: Master of Sciences
Discipline: Sustainable and Autonomous Systems
University Supervisor: Mohammed Elmusrati
University Evaluator: Petri Välisuo
Company Supervisor: Elias Alakokkare
Year: 2025 **Pages:** 44

ABSTRACT:

System-on-Chip (SoC) technology plays a key role in modern electronic devices by making them smaller, faster, and more energy-efficient. It facilitates this by combining all the main components such as processors and memory into a single chip. Today, SoCs are used in many areas like smartphones, cars, airplanes, and other embedded systems. However, traditional verification methods struggle to detect hardware-software integration issues effectively in early development stages as systems become more complex. One challenge is to detect integration problems between hardware and software components at early stages.

This thesis focuses on challenges of the traditional methods by creating a new co-simulation approach. The aim is to use data processing unit that supports up to date ARM processors architecture while keeping the flexibility advantages of Target SoC Simulator (TSS) by replacing QEMU in current framework. Since ARM Fast Models support the latest ARM processor architectures, a platform based on Fast Models was integrated into the framework. As a proof of concept, the ARM Reference Design N2 (RDN2) was selected. The approach is to integrate the ARM RDN2 platform with the TSS to use advantages from both platforms.

This thesis mainly explores two primary research questions: Can the ARM RDN2 platform be integrated into the TSS environment without changing the original structure of TSS, and will the combined system maintain synchronization and correct functionality? To answer these questions, a systematic methodology was used. The methodology involved tool selection, integration of platforms through a special interface wrapper for data transfer and testing to ensure synchronization.

Results of the early tests showed that the system worked properly. The data was correctly transferred between TSS and ARM RDN2 while both platforms maintaining synchronization by 1 microsecond. This means that the framework is useful for testing hardware and software together before the real silicon is ready. These results suggest that the system can help make SoC design more accurate and reduce development time and cost.

Despite these advantages, there were some limitations identified. Integration was complex due to differences in transaction formats and timing models used by TSS and RDN2. Additionally, the system currently lacks interrupt support. Future research should focus on testing this integration with more complex systems, optimizing data transfer speeds, and adding interrupt handling capabilities. This integrated system is a promising way to test precise and reliable early validation

in the development of advanced SoC systems. This integration specifically beneficial in fast-paced industries like telecommunications.

KEYWORDS: System-on-Chip (SoC), Target SoC Simulator (TSS), ARM Fast Models, ARM Reference Design

1 Introduction

System-on-Chip (SoC) technology plays a key role in modern electronic devices. A variety of components such as processors, memory, communication units, and special hardware can operate together on a single chip with the help of SoC. This enables devices to be smaller, faster, and more energy efficient. Because of these benefits, SoCs are used in various devices such as smartphones, cars, and systems used in the aerospace and space industries (Marcian Cirstea ,Khaled Benkrid, Andrei Dinu,Romeo Ghiriti, Dorin Petreus, 2024). When the complexity of these systems increases, it becomes crucial to have robust design and verification methodologies for modern embedded applications.

In a typical digital IC (integrated Circuit) design process, the development begins with system-level specifications and continues with modelling, designing at the RTL (Register Transfer Level), running simulations, and preparing the layout (Krula, 2024). Before the actual chip is manufactured, verification and validation steps are used to determine that the design works properly and meets the design requirements (Karmitsa, 2023). However, this high level of integration creates new challenges, specifically in the design validation phase. Usually, traditional SoC verification techniques divide the testing of hardware and software. This leads to delayed error detection, synchronization issues, and inefficient development cycles. At the same time, in addition to these setbacks, improvements in SoC design methodologies and tools aim to meet these strict standards (Drechsler, 2004).

Orlov and Syschikov explained that (2012), systems that support multicore architecture have become more common nowadays. Because of this, system reliability can be affected by problems such as delays in the development cycle. Also, simulation tools usually face the setback of staying up to date when new designs reach the market and become increasingly complex. This slows down the early stages of system development. To overcome these challenges, hardware/software co-simulation is used as a key technique in the modern SoC verification process. Before actual prototypes of hardware are ready, co-simulation enables evaluation of the interaction between software and hardware

components in a unified simulation environment. This helps create a smoother connection and validation process between hardware and software. This approach, often called "left-shift validation," begins testing at earlier stages of the design cycle (Karmitsa, 2023). As a result, developers can start iterations earlier and iterate faster. It also lowers costs and improves system stability.

This thesis aims to show the benefits of a co-simulation framework based on the integration of the TSS (Target SoC Simulator) with the ARM RDN2 (Reference Design Neoverse 2) platform. Open-source emulators like QEMU (Quick Emulator) do not always support the newest processor architecture immediately. On the other hand, the ARM RDN2 platform provides faster support for new architectures and shows more accurate results in the validation process. The integrated environment allows early validation of complex SoC systems because of TSS's flexible support for SystemC, MATLAB, and RTL-based models. This validation is performed in conditions that are similar to actual hardware environments, which improves accuracy and reliability in the design process. This solution is needed because of the requirements of multi-die SoC architectures, where IP (Intellectual Property) modules may be developed in different languages and require integration (Aho, 2023).

The integrated system is developed to solve the challenges in the SoC development process of industries that need real-time performance and high speed, such as telecommunication field (Koivula, 2023). It helps developers to find integration errors in the early phase of the SoC development process while working synchronously. This makes the design process more efficient, shortens development time, and allows faster and more flexible testing. The main objective of this thesis is to demonstrate a proof-of-concept integration of TSS and ARM RDN2 platforms to create significant advantages for both chip manufacturers and customers by improving the modern SoC development process.

The rest of this thesis is structured as follows: Chapter 2 gives a detailed review of past studies and explains the problem based on current literature gaps. Chapter 3 describes

the method used to integrate TSS and ARM RDN2. It also includes technical steps and how the platforms perform synchronously. Chapter 4 shows the results of integration and testing. Chapter 5 explains the results and discusses the limitations of the work. Chapter 6 concludes the thesis and provides recommendations for future research.

2 Literature Review and Problem Definition

This section includes detailed literature review to understand the current state of SoC design, simulation methodologies and co-simulation techniques. The selected works were chosen specifically for their efforts overcome challenges in early stage SoC design, such as limitations in hardware software co-design and validation. Relative works such as (Karmitsa, 2023), (Marcian Cirstea ,Khaled Benkrid, Andrei Dinu,Romeo Ghiriti, Dorin Petreus, 2024) and (Orlov & Syschikov, 2012) were analysed to identify current limitations. For example, challenges in early-stage hardware software verification and integration. This step focus on to deliver a base for defining the problem and developing the formulation of the research questions that guided to this study.

2.1.1 Traditional SoC Verification Methods

Due to the increasing complexity of System-on-Chip design, robust and accurate verification methods are required to guarantee functional efficiency and precision before the manufacturing process. Although there have been new improvements, traditional verification techniques are still commonly used. Some of the most well-known SoC verification techniques are simulation, formal verification, hardware emulation, and FPGA (Field Programmable Gate Array) prototyping. Each of these techniques has trade-offs in different aspects (Andrews, 2007).

The most dominant technique is Simulation-based verification because of its flexibility, simplicity, and widespread industry adoption. According to Foster and Krolnik (2008, p. 13), for the purpose of to detect design errors, simulation-based verification implements a variety of test scenarios using testbenches that simulate actual operational situations.

Moreover, Foster and Krolnik (2008), modern simulation methods commonly use Hardware Description Languages (HDLs) such as SystemVerilog, Verilog, and VHDL (Very High Speed Integrated Circuit). These languages are preferred because they have strong compatibility with many existing verification tools and methods to make simulations process

more efficient. Even with these advanced languages, Foster and Krolnik (2008) explain that, achieving full test coverage remains an important challenge. They mentioned that running simulation for large and complex SoC designs requires a great amount of computer resources such as memory and processing power. Also, this requires more time and computational power if complexity level increases in the design. However, designers should consider those facts when planning verification strategies to guarantee the reliability of the system before manufacturing.

In addition, study by Bailey, Grant and Andrew (2007) shows that creating tests manually for simulation-based verification is complex and slow process. It requires more time than automatically generated tests. Also, it may involve human error, it means some important parts of the system may not be properly tested. To figure out this problem, verification engineers developed automated test generation methods. These methods can create tests faster and with fewer errors. However, simulation-based verification causes problems when the system has very complex interactions, limiting its scalability.

Formal verification techniques have been significantly improved to support traditional simulation techniques. Formal verification techniques do not depend on running multiple test cases to validate system designs. Instead, these methods provide mathematical validation of system designs. As Kern and Greenstreet (1999) describe, formal verification examines logical properties of the system to confirm system's correctness. In the automotive and aerospace industries, where even small mistakes can have serious consequences, techniques such as model checking and equivalence checking are especially important for critical components.

In contrast to its advantages, formal verification techniques have their own significant disadvantage. The difficulty in this verification method is handling very large and complex SoC designs. Because these methods require computational resources. It means that applying them to full systems is generally challenging and sometimes impractical as detailed by Kern and Greenstreet. To support explained perspectives, Drechsler (2004)

explained SAT (Boolean Satisfiability) based techniques and symbolic model checking formal verification methods, like SAT-based techniques and symbolic model checking. Compared to earlier formal techniques that struggle to high complexity situations, these new formal techniques make verification easier and faster.

Hardware emulation has widespread usage to overcome some of the limitations and problems with simulation and formal verification techniques. According to research by Andrews (Andrews, 2007), large test cases can be run many times faster than the software simulations. Because of this high speed, it allows users to test hardware and software behaviour comprehensively. This helps users to detect integration problems earlier. However, hardware emulation systems are more expensive and more complicated to set up and use. This high cost and complexity make hardware emulation feasible only for large organizations with extensive verification resources

FPGA-based prototyping plays a key role recently, specifically for software developers who need a realistic version of the hardware in the early design process. Andrews (2007) explain that platforms that can run at real-time speeds can be provided by FPGA prototyping. This enables them to begin system testing and software development far early. Hung and Wilton also point out that development cycles could be faster and safer by using FPGA prototypes because timing and integration problems can be found early. However, FPGA prototypes have significant complexity problems. However, large and complicated designs can be extremely challenging to fit into an FPGA, and debugging issues can be time-consuming. These challenges make FPGA prototyping challenging in some cases.

Moreover, research by Kumar, Litterick and Candido, (2024) highlights the increasing challenge of verifying power management and dynamic behaviours within SoC architectures. They argue that traditional simulation techniques often fail to accurately analyse these dynamic behaviours. As a result of this, conventional simulation techniques frequently fail to capture dynamic behaviours, which restricts their ability to completely

validate modern low-power solutions. Kumar, Litterick, and Candido (2024) further suggest that, while traditional verification methods remain their fundamental role, they are not sufficient by their own to solve the challenges of growing complexity of power and dynamic behaviour verification. Therefore, they suggest to using combined methods that bring formal analysis and power simulation all together.

In summary, traditional System-on-Chip verification methods play a key role in SoC development process. Simulation-based verification, formal methods, hardware emulation, and FPGA-based prototyping are the methods that all have their own advantages and disadvantages. Even though simulation is flexible, it has difficulty to cover every test case and scale to large designs (Krolnik & Foster, 2008) (Kern & Greenstreet, 1999). Formal verification provides gives precise results however, it cannot handle large and complex systems easily (Kern & Greenstreet, 1999) (Drechsler, 2004). Hardware emulation makes verification faster but at the same time, it is expensive and complex to set up and use (Andrews, 2007). FPGA-based prototyping supports real-time system validation but faces challenges related to mapping complex designs onto FPGAs and long debugging times (Andrews, 2007). Finally, because verifying dynamic power management and behaviour is getting harder, new and combined verification methods are needed to fully manage the complexity of modern SoCs (Kumar, Litterick , & Candido, 2024). These integrated techniques can help cover the weaknesses of individual methods by combining their strengths. As SoC designs continue to grow in complexity, the development and adoption of hybrid verification strategies will become significant to ensure full system correctness and performance.

2.1.2 Target SoC Simulator (TSS)

Sophisticated platform called Target SoC Simulator is developed specifically for validation and verification of complex System-on-Chip architectures. Various type of hardware components, such as processors, memories, interconnections, and dedicated IP block are integrated with software components, such as device drivers and operating systems, into a single integrated circuit in SoCs. Conventional verification methods face challenges

because of the increasing complexity of SoCs in modern times. Therefore, traditional verification methods face challenges when integrating software with hardware, especially later in the design process. TSS follows a design approach to software and hardware integration follows the *'left shift'* approach. This approach provides an important solution by enabling hardware and software integration early in the design process. In particular, this method helps developers by significantly improving the verification and validation processes (Karmitsa, 2023).

Enormous amount of hardware and software interactions are the primary source of complexity in modern SoCs. Traditionally, hardware teams are responsible for validation of models by employing techniques called RTL simulation. In those simulations, software components that are created independently by software teams separately. This division of software components creation leads to concerns of detecting any design problems later of the design phase. This late detection could cause increasing time of development and expenses. TSS can be used to solves this problem by enabling the execution of software components such as device drivers and embedded applications with hardware models. Those hardware models at different abstraction levels, from TLM to RTL, can be integrated by TSS with executing software instructions exactly as they would on real hardware. This integration makes it possible for realistic validation scenarios which mimics to real hardware (Karmitsa, 2023).

Koivula (2023, p. 25) claimed that TSS supports wide range of modelling tools is one of the key features. SystemC, MATLAB and RTL models can be integrated and defined in TSS. TSS specifically includes SystemC models which are offering robust platform for high-level hardware description that appropriate for quick design iterations and early software integration. Additionally, MATLAB models could be included into the TSS environment to allow for algorithm-level validations and simulations. Those simulations are particularly useful for applications that are telecommunications.

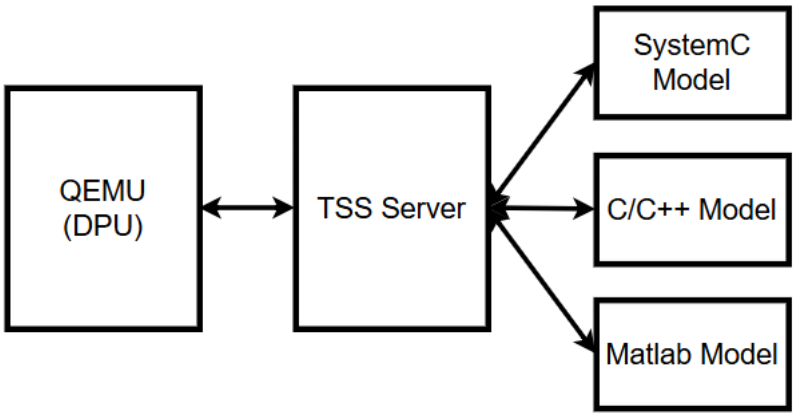

Figure 1. TSS-QEMU Configuration (Koivula, 2023)

For common usage, TSS uses QEMU as DPU (data processing unit). It is a popular open-source virtualization tool for emulating different types of processor architectures. This enables software engineers to run and test applications in an accurate way without placing real hardware to work with. The simulation environment using QEMU effectively tests and evaluates software behaviour while guaranteeing integration with hardware components (Aho, 2023).

Capability of identifying hardware and software integration issues in early stage is one of the key benefits of TSS. While executing precise hardware models and executing real software together, developers could detect the integration issues in early phase of the design process. One of the reasons of decreasing the cost dramatically is detecting issues at initial stages. Through, this leads to lower projects related risks and increase efficiency (Koivula, 2023).

Furthermore, single and multi processor architectures are supported by TSS with complicated simulation scenarios. Interactions between processors must be accurately modelled for multiprocessor SoCs (MPSoCs) which are frequently used in embedded systems and telecommunications. It used often because of its ability to precisely represent shared memory and inter-processor communications. TSS provides a robust framework

for simulating these connections. As a result, TSS provides useful advantages of evaluating synchronization, validation of data transfer mechanisms, and timing characteristics that are critical for MPSoC performance (Aho, 2023).

One major example of TSS's adaptability is its compatibility with various simulation and verification environments. Specifically made simulation models, vendor-specific tools and open-source libraries could all be integrated into the platform. This flexibility allows minimising delays from vendor while allowing design teams to select appropriate verification tools. This particular level of flexibility enhances results, simplifies and makes easier for teams to collaborate, and makes it easier to integrate with third-party tools and processes (Krula, 2024).

Furthermore, TSS offers notable benefits for timing validation. Most of the embedded and real-time systems, it is essential to be sure that software runs within the appropriate time intervals in relation to hardware processes. Early timing validations enabled by TSS's ability to precisely synchronize hardware modelling and software execution. As a result of these, engineers can reliably guarantee that the finished SoC satisfies its performance and timing requirements without largely depending on expensive physical prototypes (Koivula, 2023).

Capability of TSS to simulate Control and Status Register (CSR) interaction is another crucial feature. Control and Status Registers are crucial interfaces that software uses to configure and monitor hardware components in a SoC. TSS allows early access testing of CSRs to help developers identify potential register-level issues during early stages of development. By enabling early corrections and reducing the risks connected to late-stage design changes, early detection of CSR related issues greatly enhances the integrated system's reliability (Krula, 2024).

2.1.3 QEMU-Based SoC Verification

QEMU (Quick Emulator) is open-source software used for virtualization and emulation. It is generally used for embedded systems development and hardware software co-validation (Koivula, 2023, p. 6). QEMU allows those developers to run and test software for different processor architectures without needing real hardware. When hardware prototypes are not ready yet, QEMU is useful in these early stages of System-on-Chip development (Karmitsa, 2023). Flexible and suitable for a wide range of application architectures such as ARM, RISC-V, x86, and PowerPC are supported by QEMU (Koivula, 2023).

At the core of QEMU's functionality is its dynamic binary translation technique. This method transforms a guest architecture's instructions into instructions that the host computer can execute in real-time (Karmitsa, 2023). Traditional interpretation works by reading and running each instruction one by one, but it decrease the speed of the process. However, dynamic binary translation changes blocks of instructions into a form that the host computer can run and saves those instructions. This means that the same code can be run again without repeating the translation, which makes the process much faster and more efficient. As a result, the emulated system responds quickly and works efficiently, especially for tasks such as software development, debugging, and performance testing. It also allows developers to test and improve their code quicker in a virtual environment. In the early phases of SoC design when actual hardware is not yet ready, this significantly helps development process by reducing time and cost (Karmitsa, 2023).

QEMU mainly works in two different modes: system mode and user mode. In system mode, QEMU emulates a full hardware platform including processor, memory, input/output (I/O) devices, and network components (Aho, 2023). Because QEMU allows engineers to test how all components of the systems function all together, this full-system emulation is commonly used for hardware software co-validation. On the other hand, user mode has limited usage. Without simulating the entire system, It allows individual Linux or BSD user-space applications. Those applications are developed for several CPU (Central Processing Unit) architectures to execute directly on the host computer

(The QEMU Project Developers, 2025). This mode is useful for fast software compatibility checks on different architectures.

In Nokia's Target SoC Simulator (TSS) framework, QEMU is traditionally used as a (DPU)data processing unit to simulate the CPU component of an SoC according to Karmitsa (2023). This setup has crucial importance for pre-silicon validation and testing of interoperability of hardware and software before the actual chip is built. He supports that, developers can test software programs such as device drivers and firmware on QEMU to confirm that they function properly together and their compatibility with hardware interfaces.

According to Karmitsa (2023), QEMU works properly with other simulation and emulation tools in the TSS framework. It connects to TSS through a standardized transaction-level modelling (TLM) interface. To create efficient data transfer between simulated processor in QEMU with the hardware components modelled in TSS, TLM interface used in co-simulation environment. This design allows the system to closely replicate how real hardware and software interact. This improves the validation process by creating a realistic environment where integration problems and functional errors can be found early in development process of SoC (Karmitsa, 2023).

According to Karmitsa (2023) , another important reason why QEMU is widely used is its extensibility. Because QEMU is open-source, developers can adjust and improve its features to meet the specific needs of their projects. The flexibility allows developers to create customized components such as peripherals, debugging tools, and other features that are specific to the tested system. Such customization makes sure that the validation environment closely matches the real working conditions of the final SoC. This leads to more reliable and accurate validation results.

Furthermore, QEMU supports hardware acceleration technologies such as Kernel-based Virtual Machine (KVM) when running on Linux hosts. Making simulations run much

closer to the speed of real hardware, mentioned technologies can greatly improve the speed of simulations (Aho, 2023). This performance improvement is crucial for projects that require extensive testing iterations within strict deadlines. It helps developers respond to design changes quickly and reduces the total time needed for development (Aho, 2023).

Although QEMU has several advantages, it also has some important limitations, especially in supporting the latest ARM CPU architectures. Because of this, it is not the best option for testing on projects that use new ARM cores. Because of that, adding the ARM RDN2 platform to the TSS environment is a strategic decision. It helps to solve these problems and allows for more accurate, flexible, and reliable validation, especially in terms of working with up-to-date IP models.

2.1.4 ARM RDN2 and Its Role

According to Arm (2022), ARM Reference Design (RD) is a set of materials that offer an overview insight of common compute subsystems that can be developed and implemented into Arm IP generations. Software development and testing will be done on Arm RDN2 platform before actual hardware is read. This platform depends on the ARM Neoverse N2 processors that made for high-end systems such as data centers and servers. RDN2 includes components to perform properly. CPU, memory controller, interrupt controller (GIC), and PCIe (Peripheral Component Interconnect Express) support are examples of the crucial components of a real system that are included in the RDN2 platform. All those components allow engineers to test and run their programs in full virtual system.

Arm (2021, p. 2) explained that the Arm Neoverse N2 platform is efficient and useful for 5G networks. It provides high performance and uses less power in figure 2. Neoverse N2 is 40% faster than Neoverse N1 with the same power. It is also 2 to 3.5 times better than

old processors in performance per watt. This is important for 5G technology and applications. Neoverse N2 has different types. It can have 8 to 128 cores. This is good for different 5G systems from small servers to real 5G application equipment.

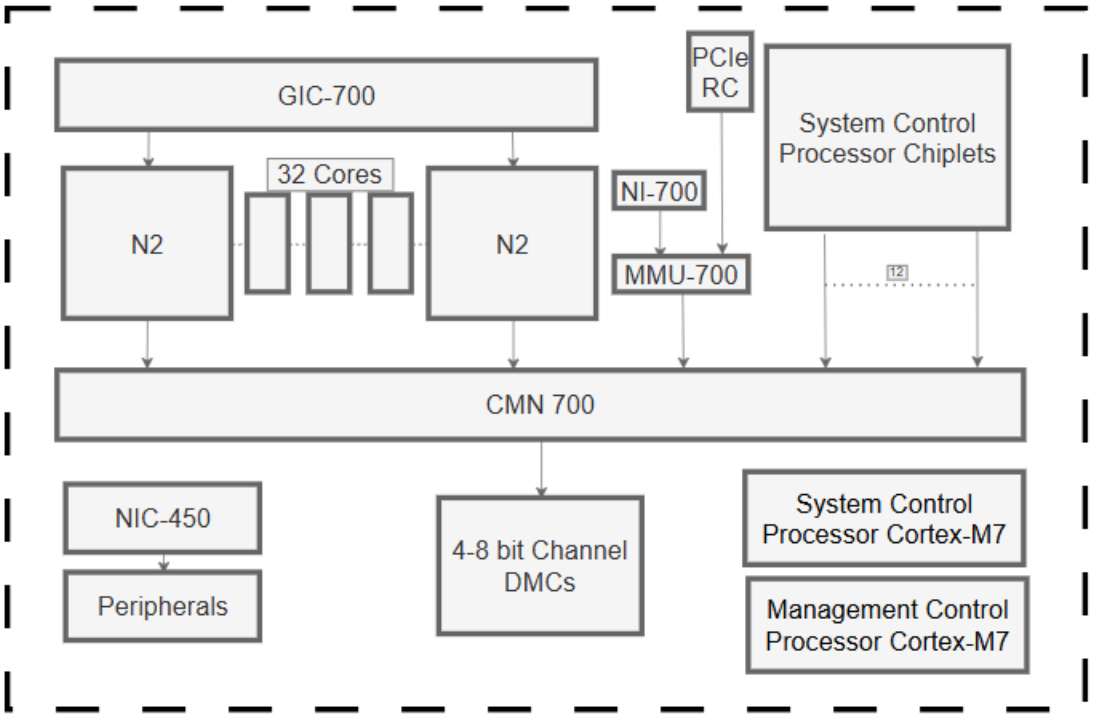

Figure 2. Neoverse N2 Block Diagram (ARM, 2021)

ASTF (Arm Software Trace Format) tracing is a useful feature of Fast Models inside of RDN2. ARM explained that (2025), This framework takes logs of the operation of the software while it is operating on the virtual platform. It can store data such as which functions are active, how long they utilise which memory is accessed, and which instructions are executed. Those data will help engineers to understand how the program operates and how to speed up the process. The ASTF plugin must be activated in the simulation environment to collect those data. After that, the trace data can be stored as a .astf file format and opened .astf format files with special tools like ARM Streamline. By using those tools, data can be seen in tables and chart for analysing performance. In addition to those tools, RDN2 platform supports PMU (Performance Monitoring Unit). It is a hard-

ware feature that counts events like cache hits or instruction cycles. However, Fast Models do not support cycle-accurate simulation. Because of that, PMU results may not reflect precise timing behaviour. Against that feature, PMU data can still be useful for general performance insights that is recommended for accurate performance on real hardware.

2.1.4.1 Fast models

According to ARM (2025), FM (Fast Models) are software tools to help simulate ARM processors and systems. Key benefit of ARM FM is they do not necessarily need real hardware to operate. Instead, they operate on standard PC or server. For this reason, developers utilize Fast Models while developing or testing programs, drivers, and operating systems.

Arm explained that (ARM, 2025), Fast Models use a system called Programmer's View (PV). Because of that, it does not simulate hardware cycle by cycle but it is fast and still accurate for software development. A result of their speed, real programs such as Linux or Android, could be run and tested without a physical board. Fast models have all those advantages because of it includes several tools such as System Canvas, System Generator (SimGen), Model Debugger, Model Shell.

Debugging and tracing interfaces such as Model Trace Interface (MTI) and the Component Architecture Debug Interface (CADI) are supported by Fast Models. These interfaces allow users to use tools to connect to the model and obtain data. Such data can be collected and show program activity by using some special trace tools like Tarmac Trace or Fastline. These traces are valuable for understanding and analysing the internal processes occurring inside of the system.

Fast Models also include models of many Arm processors such as Cortex-A, Cortex-R, and Cortex-M series. It also supports memory, buses, GPUs, and variety of other components. Those architecture are not available as open source. This enables developers to start

testing of software development on cutting edge architecture before the real silicon available. The reason how rdn2 platform supports the updated CPU architectures is its Fast model compatibility

In short, Fast models' advantages vary in different perspectives. They are useful because they allow developers start coding and testing before the real hardware is available. This early start can significantly decrease development time and launch products to market faster. Also, it is flexible so users can build custom systems and various setups. Debugging and tracing support helps finding problems and improve software quality because Fast Models are fast, and users can run large number of tests quickly.

2.1.4.2 SystemC

Modelling language called SystemC is often used for building hardware and software systems in a high-level and flexible way (Accellera, 2009). It enables developers to model and simulate the interactions of system components in virtual models. While integrating hardware and software in the same simulation environment, SystemC is especially crucial.

In this concept integration, SystemC is used for different purposes but one of the most important of them is for Arm Fast Models. ARM explained that (2025), SystemC Export is a feature of Fast Models that supports exporting virtual platforms to SystemC format. When a developer builds a platform using System Canvas or writes LISA+ code, the System Generator tool can create a SystemC model. These SystemC models follow the TLM 2.0 (Transaction Level Modelling) standard (Accellera, 2009). This is important because industry use it that SystemC models to create IP models. This allows Fast Model platforms to be integrated into bigger simulation environments, including RTL simulators or custom hardware models.

Moreover, ARM (2025) elaborate that, The SystemC Export feature in Fast Models allows Fast Models to connect Fast Models with other SystemC-based simulation tools. This

helps developers use Fast Models together with more complex simulation environments, such as custom hardware IP models. It also allows co-simulation with hardware models, which gives more accurate results. One key advantage is that developers can test the interaction between hardware and software before the actual hardware is ready in the design process. This early testing helps reduce errors and development time. Another advantage is the flexibility to connect different modules, which is useful in complex system projects. Because of these benefits, Fast Models can be used not only on their own but also inside advanced SystemC based simulation platforms.

To connect different parts of the system such as CPUs, memory, interconnects, and peripherals easily, Fast Models use standard SystemC ports. Thus, Fast Models not only can be used only their own but also can be used more complex Systemc based simulation environments (ARM, 2025). Furthermore, SystemC is used in TSS modules. These modules are modeled by using SystemC to make models more accurate and flexible. Considering this, TSS creates a full virtual prototype by using SystemC based modules and other model types.

2.1.5 Summary of Literature and Identified Gaps

Although conventional System-on-Chip verification approaches such as simulation-based testing, formal verification, hardware emulation, and FPGA prototyping, have advanced significantly, they have made significant progress, but recent studies show that there are still ongoing limitations. Complex SoC designs are still unable to be checked early, accurately, and efficiently due to these issues. To begin with, simulation-based verification is flexible and commonly used, but it has problems with scaling when SoC systems become more complex. It is still challenging to get full test coverage because the number of test cases increases rapidly as system complexity grows. Also, creating simulation scenarios manually takes a lot of time and can be missed in some cases. Because of this, various integration problems in complex SoC designs may not be found (Krolnik & Foster, 2008) (Foster & Krolnik, 2008; Bailey et al., 2009).

Formal verification methods perform appropriately for evaluating small and safety-critical parts by using math. But they are usually not practical approaches for large systems because they need significant computing power (Kern & Greenstreet, 2005). Also, these methods require a high level of expertise and have trouble handling changing behaviours. Hardware emulation and FPGA prototyping allow faster and more realistic testing. But they have some setbacks such as high setup costs, inflexible platforms, and trouble adding custom IP blocks. Also, these methods usually delay the hardware/software integration to the end of the design process. This increases the possibility of late detection of compatibility errors, which may result in costly modifications for SoC projects.

Furthermore, many validation platforms depend on open-source emulators like QEMU. QEMU is flexible and used in many projects, but it often does not support the newest processor architectures such as ARM N3. This causes challenges when new cores need to be used. These problems happen because developers test their systems using older processor architectures specifically in areas where fast adoption of new cores is critical.

In conclusion, current methods have some significant limitations. They often do not support early integration of hardware and software. They have challenges in validating new ARM processor architectures. Finally, these methods can be expensive, challenging to implement in different projects and not easy to use. This thesis demonstrates the integration of using the TSS environment together with the ARM RDN2 platform. This approach allows developers to validate modern SoC systems earlier and more accurately. Correspondingly, it helps to integrate upcoming IP modules easily and quickly, while keeping full system functionality.

2.1.6 Problem Statement

In recent years, the complexity of System-on-Chip designs has increased rapidly. This situation requires flexible tools for specific needs in early phase in testing of SoC development. One popular tool used in SoC co-simulation is QEMU. It is open-source and easy

to use, but it does not support up to date ARM architectures. This is a significant challenge for projects that are using the latest ARM processors, especially in fast-developing industries like telecommunications.

Because of these limitations, this thesis focuses on replacing QEMU and using the ARM RDN2 platform as a Data Processing Unit (DPU) for TSS framework. RDN2 supports newer ARM cores and provides significant features like ASTF tracing and better debugging tools. The main goal is to integrate RDN2 as a DPU without changing the internal structure of the TSS framework. However, achieving properly working integration has complexity because TSS and RDN2 operate using different transaction models and timing frequencies. Ensuring synchronized operation between these two platforms presents a technical challenge.

This thesis answers two main questions:

- Can ARM RDN2 be used as a DPU for the TSS without changing TSS's original structure?
- Will the newly integrated platforms adjust their own internal clocks, operate in synchronously, within the same time frame and maintain functionally correctness?

3 Methodology

This section described the methodology of the integration between the ARM RDN2 platform and the TSS environment. It offers precise justification for the use of simulation tools, outlines the integration process's mechanisms and procedures. The methodology includes research framework, selection and setup of simulation tools, integration of ARM RDN2 platform and TSS, RDN2-TSS configuration, conclusion and Implications for SoC design. The main focus is on IP model flexibility, early validation, and support for the latest ARM architectures.

3.1 Research Framework

The study presented in the thesis uses a systematic approach to reach robust hardware software co-validation known as left shift validation. This approach's basic principle is to evaluate hardware software interactions at the earliest possible stage, which means greatly increasing reliability of complicated SoC designs and decreasing of total costs and development time.

The research framework of this thesis is structured into four main parts. First, the tool selection and justification phase involve identifying and selecting suitable simulation tools to prove the concept and satisfy the specific requirements of the project. Considering their beneficial features, both ARM RDN2 and are relevant in this scenario. The next phase is integration. This section answers that “how the chosen tools are connected?” in this section. The objective is to ensure that the tools operate inside the same environment and precise data transfer. This includes creating necessary blocks of codes and building the system that required for their integration. The third phase is validation. Scenarios that are close to real life applications has been used to validate integrated system. This validates whether the integration was successful and whether the system operated properly. This helps to make the implementation of a co-validation framework suitable for early-stage SoC development

3.2 Selection and Setup of Simulation Tools

To implement theoretical study to the practical simulation, TSS simulator and ARM RDN2 Platform are chosen as the main tools. This selection was made because of their unique capabilities and ability to support one another to deliver efficient and accurate co-validation environment.

The ARM RDN2 platform was chosen because it is a reference hardware design tool made for testing complex System-on-Chip designs. Despite software simulators, ARM RDN2 provides more realistic results by demonstrating how the hardware responds closer to real time scenario. It also runs faster than other tools which are useful for testing complex systems or run longer validation tests. Another benefit of ARM RDN2 is versatility. It is particularly useful for newer SoC designs because it can manage a wide range of hardware configurations including systems with multiple processors.

The Target SoC Simulator (TSS) was also selected because it supports different modelling languages like SystemC, MATLAB, and RTL. TSS allows developers to start testing the system in early stage of SoC design process, even before the real hardware is ready. It uses transaction-level modelling to simulate system faster. In this simulation process, developers can observe how components interact with each other. TSS also simplifies the integration of different simulation components into one environment. This allows more flexibility for verifying bot software and hardware at the same time.

To use beneficial features of the tools, this study setup includes several important steps to create a working simulation and validation environment for System-on-Chip designs. First, the Target SoC Simulator is configured to support booting with RDN2 architecture. This means that, necessary configuration files and codes adjusted to work with RDN2. Then, the TSS is connected to the ARM RDN2 platform. This connection allows ARM RDN2 platform to work together with IP (intellectual property) models that run on TSS. To make this connection useful, transaction conversion mechanism is created. In this mechanism, TSS transactions are converted into ARM RDN2 transactions or vice versa. This guarantees that data can transfer correctly between TSS and ARM RDN2, allowing both platforms to effectively data transfer during simulation and validation.

In summary, the selection and setup of the TSS and ARM RDN2 platforms provide a strong and flexible environment for co-validation in the SoC design process. ARM RDN2 was chosen for its capability of supporting latest ARM CPU architectures. Because it is compatible with ARM FM with behaviour close to real silicon. Because of this, it is ideal for testing modern, multi-core SoC architectures. On the other hand, TSS was selected because it supports different modelling lan-

guages and allows developers to test on their system in early stage, even before the real hardware is ready. These two tools work together with the help of transaction transformation mechanism. This mechanism changes the transaction format so both platforms can send and receive data correctly. As a result, the simulation and validation process become smoother and more accurate. This setup increases the speed of development and allow easier testing for modern SoC designs.

3.3 Integration of ARM RDN2 Platform and TSS

This integration of the ARM RDN2 platform with TSS is the core technical focus of this thesis. By integrating TSS and ARM RDN2 platform, the integration aims to proof of concept to assist high-accuracy co-simulation. This chapter explains the integration process in detail; how the platforms transfer data, how they stay in sync, and how register-level interfaces are used.

3.3.1 Transaction Interface of ARM RDN2

The ARM RDN2 platform uses LISA-based models. However, it connects to SystemC TLM-2.0 environments through AMBA-to-TLM bridges, which allow data transfer between system components and external simulators. This data transfer can be simulated not also with exact timing but also with approximate timing. The key part of this method is a class called tlm_generic_payload. This class is used to carry basic memory operations such as read, write, and debug in a standard way.

To define the transactions, tlm_generic_payload class provides flexible and protocol-independent approach. It has various properties to describe the transaction's structure. Therefore, RDN2 and other components of the system can transfer data efficiently and continuously. Each and every tlm_generic_payload object represents one transaction and includes different attributes that explains what the transaction's duty and how will it work. The structural composition of tlm_generic_payload includes different parts of the transaction. The used components in this project are described below:

- m_address: It represents transaction’s destination memory address. It set to zero by default. It means, it has not yet been assigned a valid address.
- m_command: It specifies the behaviour of the transaction. The default command is TLM_IGNORE_COMMAND. It implies that operation has not been defined nor read or write at instantiation.
- m_data: it implies to the data will be used in the transaction. If it set to 0, it means it is null pointer to the data buffer.
- m_length: The size of the data buffer is set to 0, meaning that no data will be sent unless explicitly defined.
- m_response_status: This demonstrates that the transaction has not yet been completed if it is set to TLM_INCOMPLETE_RESPONSE. It means that transaction has not yet been executed or processed by any target component.

These parameters form the payload's initial state and must be properly defined before the transaction can be sent to its destination.

3.3.2 Transaction Interface of TSS

The Target SoC Simulator serves as a modular simulation framework designed to facilitate early-stage co-validation of hardware and software components in System-on-Chip design. In this thesis, TSS was configured to enable behavioural modelling data transaction with the ARM RDN2 platform. This simulator plays a key role in abstracting hardware behaviour and preserving timing functionality because, TSS models. The communication between system components using a well-defined transactional abstraction. The simulator operates with discrete transaction objects, which encapsulate all relevant information about a memory or register accesses. Structured set of attributes used in TSS transactions, and all transactions includes those features. TSS transaction includes:

- AddressPart: Represents the starting address of the targeted endpoint in the global address map. Basically, it represents the first register address of models which the transaction is routed.

- OffsetPart: Indicates the byte-level offset relative to the addressBase. It is specifying the exact location of the register will being accessed. This is essential for accessing specific register's address.
- LengthPart: Specifies the total length of the transaction payload that measured in bytes. This parameter determines the size of the data will be transferred.
- Value: Contains the actual data that will be transferred in the transaction. For write operations, this field holds the value to be written; for read operations, it is filled with the data returned from the module.
- Command: It indicated the behaviour of the transaction. It is typically either a read or write operation. This attribute defines the transaction's effect on the memory or register space.

This transaction mechanism allows for an efficient representation of hardware behaviour while providing synchronised integration. It is appropriate for early-stage Soc development validation because it supports a wide range of system architectures. In this thesis TSS was configured with the following capabilities:

- Work with SystemC for modelling hardware components at a high level.
- Use timing synchronization to stay sync with the ARM RDN2 hardware platform.
- Flexibly support custom peripheral models, which can be quickly modified as needed during development.

In summary, the transaction interface in TSS is crucial to fast and high-level data transfer. Instead of using detailed connections, TSS uses structured transaction objects to makes the simulation simpler to manage and more flexible. In this thesis, the TSS transaction mechanism was a key part to support a reliable data transfer with the ARM RDN2 emulator. It supported synchronized operation and allowed it to check values at the register level. This approach not only speeds up early SoC development but also improves the quality of software-hardware testing. As a result, it played a significant role in reaching the main goals of this thesis.

3.3.3 RDN2-TSS Configuration

CSR read and write functions are supported by the integration between ARM RDN2 and TSS. The implementation of these functions could be described as a transaction conversion process because the main issue about the integration is TSS and ARM RDN2 platforms operate by using different types of transactions. The RDN2-TSS-wrapper enables the conversion between these transactions. Effective data transfer between the two platforms can be achieved by this wrapper which also enables the precise conversion of core message within transactions. To comprehend this situation better, one could take at the TSS transaction and TLM generic payload transaction.

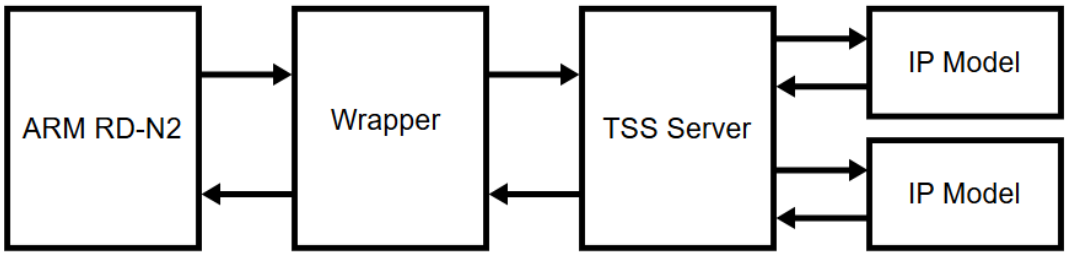

Figure 3 Visualization of RDN2 and TSS Configuration

Transactions of RDN2 and TSS are not exactly same that is why using the wrapper is essential. One of the most important differences between transactions are address fields. In TSS, instead of directly writing the address into the transaction, it works on first identifying the IP module related with the transaction. the module to be identified is determined by an address mechanism built in the wrapper. Once the related IP module is identified, the target register is determined by adding an offset value to the base address of the identified module. After determined the register, execution of the command that is specified within the transaction occurs. Depending on the type of operation (read or write), the data section of the transaction is either filled or cleared.

To clearly explain how this data field changes based on the type of operation (read or write), a simple example can be considered in which read and write operations are performed one after another. On RDN2 platform, a basic write command includes the register address and the data to be written. When write command executed, command arrives at the wrapper as a TLM generic payload transaction, it is converted into a TSS transaction into the wrapper. Following transformation, the TSS transaction proceeds to the related module, identifies the correct register, and completes the write operation. In this process, TSS waits until the current operation finishes before starting another. Although this method uses blocking transaction approach to prevent many possible CSR operation failures. However, it slightly increases the time needed for completion of the transaction execution.

After the write operation is completed, CSR read command can be used to verify and read the data we wrote in write operation. A read command is sent to the same register address where data was previously written. When we wrote read command in RDN2, read command is a TLM generic payload transaction. Inside the wrapper, this TLM generic payload transaction is converted into a TSS transaction and directed to the appropriate IP module’s register within the TSS platform. The data at the specified register address is read, and this data remains within the TSS transaction during this step. The integrated system transforms the TSS transaction back into a TLM generic payload transaction because the goal is to see this data on the RDN2 platform. Later that point, the RDN2 system receives this TLM transaction directly and displays the data that was saved in the TSS transaction's data field. As explained previously, this data is exactly the value stored in the register defined by the transaction address. This example clearly explained how the integration process operates.

For this integration of the RDN2 platform within the co-simulation environment, the firmware layer has important role in the operation. In this setup, the RDN2 system is configured to boot on Linux operating system by using Arm Trusted Firmware (ATF) as a bootloader. ATF is responsible for the basic setup of the RDN2 platform. This includes

setting up the processor exception levels, enabling secure world configurations, and starting important system parts like memory controller. This booting process allows the RDN2 platform performs like real hardware. This is important for realistic software validation. By using ATF to the simulation, the integrated system can run the full boot process from firmware to the kernel. This provides early test of secure firmware and kernel startup in the RDN2-TSS system.

Maintaining time-based synchronisation between the ARM RDN2 platform and the TSS simulation environment is critical, because both systems use different simulation engines. To use them synchronised, they must align on time to function properly. particularly for time-sensitive tasks such as reading from or writing to the CSR, synchronisation is critical to ensuring that operations are completed in the proper sequence, data is transferred appropriately, and test results are valid.

In this setup, the ARM RDN2 platform and the TSS simulator were designed to stay in sync by 1 microsecond time interval. However, both platforms can work faster than 1 microsecond if needed. To synchronise both platforms, both platforms are set to 1 microsecond run time. When the RDN2 finishes its 1 microsecond step, it sends an update to the TSS. Then, the TSS run same amount of time to stay synchronised. This way, the two platforms stay synchronised with each other without adding extra delay time.

This synchronization is implemented through a timing update function within the RDN2-TSS-wrapper. Each time the RDN2 advances its simulation by one microsecond, the wrapper creates and send a timing update in TSS, incrementing its simulation time 1 microsecond. This approach effectively sets TSS's simulation time to RDN2's timing, while still allowing TSS to execute internal operations at higher resolution if needed.

The Global Quantum setting on the ARM RDN2 platform is an essential part of this configuration. The global quantum determines the least amount of simulation time that

spans before the RDN2 platform syncs with other components or simulators. In this situation with adjusted by using quantum, it is defined as one microsecond. This let the system to transmit timing changes and remain in regular intervals sync with the TSS simulator. This is crucial for precise and clear data transfer operations between both systems.

To sum up, this created synchronization method allows the DPU and TSS to work together smoothly. It helps both systems sync with the same time interval, operate register-level actions correctly, and verify the system in a reliable way all parts of the integrated platforms. This procedure guarantees precise timing alignment and synchronized operation between the DPU and TSS.

3.4 Conclusion and Implications for SoC Design

The final stage of this study's methodology thoroughly examines whether the combination of the findings from the simulation would be possible and beneficial for the SoC design cycle by doing TSS emulation with ARM RDN2 platforms. Reducing development time, improving design efficiency, and cost savings are the main areas where this study should have a positive effect.

By combining practical and theoretical approaches, this research aims to demonstrate the potential benefits of advanced co-simulation methods to change modern SoC design processes. It aims to solve the challenges faced by engineers designing SoCs and to provide a solution that can be used in different industries.

4 Results

This section shows the results obtained from the proposed methodology, focusing on the outcomes of tool integration, co-simulation framework validation, and performance evaluation. The results are organized into subsections, highlighting the initial configuration process, validation metrics, and the broader implications for System-on-Chip design.

4.1 Initial Configuration and Tool Integration

The first thing of the thesis includes the configuration of the TSS emulator for emulating both multi-core and single core architectures. After, the ARM RDN2 was integrated to develop the connection and communication protocol. This step was required because the creation of a robust transaction mechanism provides robust data transfer between the hardware and software platforms. The transaction mechanism handled data transfer and ensured efficiency in interactions during simulations.

The integration showed that TSS with ARM RDN2 is integrated properly with wrapper framework. This wrapper transforms platform’s transactions into one form of the platform to other one. This step started the development of a single and unified simulation environment. Initial tests showed the framework effectively handled complex SoC designs and provided accurate outputs for chip architecture. These multi-die architectures provided accurate simulation outputs. This integration validates the effectiveness of co-simulation methods in SoC design, leading to further exploration of its benefits. Unlike prior studies, which focus on separate validation steps, this approach unifies hardware software testing, enhancing overall efficiency.

4.2 Validation of Co-Simulation Framework

The co-simulation framework was validated through a series of unit tests. These tests evaluated CSR read and write functions of integrated system. They specifically assessed RDN2 and TSS's ability to communicate effectively with IP models. Robust data transfer

is important to guarantee the system's overall functionality. Simulation data was logged during these tests. The results were then compared with benchmark datasets. In the end, the results showed a high degree of coherence with the expected outcomes.

This validation not only ensured functional correctness but also include timing verification between the platforms. Specifically, integrated simulation environment was tested for its ability to maintain synchronisation between the ARM RDN2 and TSS platforms. This was achieved by monitoring and comparing the simulation time progress of both platforms by using metric called tick span. Tick span refers to the number of simulation time units (steps) that a platform has advanced during a given period. TSS and DPU tick spans were collected from simulation logs, and these values were compared to determine synchronization accuracy. If the difference between tick spans remained within a predefined acceptable error rate, the systems were operating as time synchronized. The result of the comparison must show that all synchronization errors stayed below the threshold. It will confirm correct timing alignment.

In addition to these results, the co-simulation platform ensured robustness in handling various use cases. During the initial validation phase, the ability to identify integration errors reduced development costs and decreased development time for the design process. The synchronization test provides a strong evaluation for SoC co-simulation environments with CSR data transfer validation. The logged data from these tests will be used as a benchmark data for future research which aims to optimize SoC co-simulation frameworks. Specifically, tick span comparison method provides a practical approach for timing verification in future implementations.

4.3 Performance Metrics Evaluation

Correct test execution is the last step needed to demonstrate that this proof-of-concept simulation integration has been finished successfully. After that, it is necessary for analysing the test findings. The indicated tests were performed on the system where a concept was integrated and verified. One of the tests called IP model function test. IP model

is responsible for approximate timing processes therefore, the purpose is to determine if the ticks in the IP model increase by the same amount over the same amount compared to DPU. After this test, the results are displayed statistically in the same display that the test run. Timing errors that exceed a certain limit can cause synchronization problems. The system can handle timing errors maximum of 1 us without losing synchronization. If the error is smaller than this limit, it does not affect the system's synchronized behaviour.

The test will read and store 100 ticks from DPU and IP model with a sampling interval of 1 microsecond. To determine the tick span, the difference between current tick and previous tick is calculated. For the test to pass, difference between tick intervals from the DPU and IP must not exceed 1 microsecond. Also, those tests confirms that the transaction transfer system is working correctly. Timing information is sent through the transaction mechanism, which shows that it functions properly. It also proves that the system can access CSR addresses, and that both read and write operations work as expected. The mentioned test is executed 20 times to guarantee the consistent results. Even with minor deviation can be observed, results are consistent as shown in figure 4. This means that the system components, ARM RDN2 and TSS, synchronise their internal clock frequencies accurately. In the test, performing a read operation in 1 microsecond interval also creates some delay in the results. Each read command causes a certain amount of latency, which contributes to the overall deviation. Therefore, some of the observed 1 microsecond level errors comes from the read command itself.

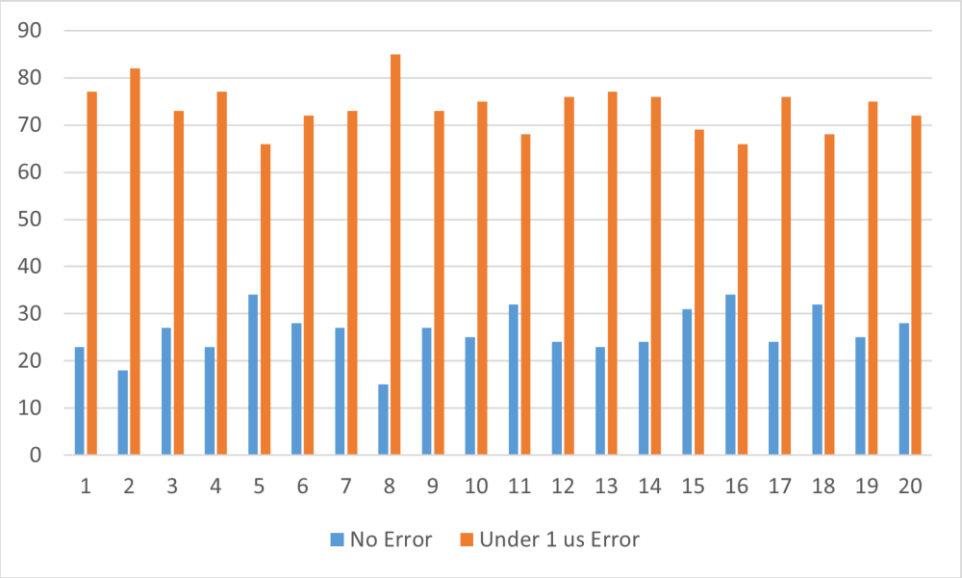

Figure 4. IP Model Tick Span Test Results

In addition, the booting time increased by 120% compared to booting time of the RDN2 alone. This means that overall system booting time is significantly affected. The main reason for increasing the booting time is to the way the simulation progresses in fixed time increments. This adds delays while keeping both systems synchronized at 1 micro-second intervals. However, once the system is booted, there is no additional waiting time required to start the tests. Because the test is simplistic, the testing duration was not affected. However, it could increase if more complex tests were executed.

4.4 Implications for SoC Design

The integration of TSS with ARM RDN2 platforms helps to solve key challenges in early-stage hardware software integration. It allows earlier detection of errors, increase the speed of design workflows and the flexibility of IP models written in different programming languages. These features make the approach more suitable than traditional methods, especially for complex SoC designs. This method is suitable for using in industries where fast design cycles and high performance are important, such as telecommunications, automotive, and aerospace. In the field of telecommunications, faster validation

helps new technologies reach the market sooner. Similarly in the automotive sector, early error detection can reduce expensive design changes in safety-critical systems.

5 Discussion

This section focuses on the study's results, their importance, and how they compare with previous research. It also highlights the study's limitations and offers suggestions for future work.

5.1 Context and Overview

Modern System-on-Chip architectures are becoming increasingly complex, which makes early-stage validation more important than before. QEMU are commonly used for software emulation and early testing but it has limitations. Because QEMU doesn’t support the latest ARM core architectures, these limitations can delay hardware/software integration and reduce the accuracy of validation process. As a result, there is a growing need for more advanced simulation platforms that offer better compatibility with new processor technologies and support early hardware software co-simulation.

5.2 Interpretation of Results

The results of this integration confirms that the integrated system operates as expected. The TSS and ARM RDN2 platforms can transfer data properly with the help of the rdn2-tss-wrapper. This wrapper is responsible for converting transaction format between two systems. To verify this setup, IP model tick span test performed several times.

First, IP model tick span test confirmed the CSR write and read operations. In this test, a value was written to a specific register. Then, wrapper tried to read the same value and sent it to RDN2 platform. At that time, the ARM RDN2 platform has two values, one is sent one is received. The test compares those result to confirm that the data transfer between the two platforms was working correctly and without errors.

Second, IP model tick span test evaluates timing synchronization. During the simulation, the time progress of both platforms was compared to each other as a tick span. After

many steps, the difference between the platform tick spans was measured. The results proved that the time difference remained below 1 microsecond. This means that both systems were running in synchronization.

In conclusion, IP model tick span test demonstrates that the integrated simulation system functions properly. Data transfer is accurate, and the timing between the platforms is synchronized. This kind of performance is important for real-world SoC development, where reliable and early testing is required.

5.3 Comparison with Previous Studies

In earlier studies, many researchers used simulation environments separately or combine them in different integration setups. One of the most important advantages of TSS is its flexibility in adding new IP models. TSS support various IP blocks written in different languages such as SystemC, RTL, or MATLAB. However, TSS includes QEMU as a DPU and QEMU doesn't support newer processor architecture. On the other hand, the RDN2 platform supports new ARM cores and includes useful performance analysis tools. But RDN2 is not as flexible as TSS when it comes to working with custom IP models or different design languages.

Compared to current approaches, this thesis shows that integrating TSS with ARM RDN2 provides a better solution. Integration allows developers to build and test systems that include custom IP models on the newest ARM core architectures. In older methods, developers often had to choose between flexibility and testing their design on new processor architectures. With this new method, both goals can be achieved in one environment.

5.4 Limitations

In earlier studies, many researchers used simulation environments separately or combine them in different integration setups. One of the most important advantages of TSS is its flexibility in adding new IP models. TSS support various IP blocks written in different

languages such as SystemC, RTL, or MATLAB. However, TSS includes QEMU as a DPU and QEMU doesn’t support newer processor architecture. On the other hand, the RDN2 platform supports new ARM cores and includes useful performance analysis tools. But RDN2 is not as flexible as TSS when it comes to working with custom IP models or different design languages.

5.5 Recommendations for Future Research

Based on the results and current limitations, there are multiple recommendations that could potentially be implemented in future research. First, the simulation system should be tested with more complex SoC architectures. This will improve the understanding of the system's functionality and advantages in more complex designs.

Second, the data transfer process between TSS and ARM RDN2 can be improved. More powerful processors implementation in the server would reduce booting time. Additionally, it may prove advantageous to collaborate with hardware manufacturers. Their feedback could demonstrate the practical application of this system in the development of actual products.

Third, interrupt support should be added in future versions. This feature is important for many practical systems and would make the simulation environment more complete. In addition, if a project does not require very detailed results but needs faster simulation, using TSS with QEMU configuration might still be a better option. That setup is quicker to start and easier to manage, especially for simpler validation tasks.

5.6 Conclusion

This study has demonstrated that by integrating early hardware and software validation, the integration of the TSS and ARM RDN2 platforms greatly improves the efficiency of the SoC design process. This study focuses on the direct integration of the RDN2 and TSS

platforms and highlighting its advantages over QEMU-based application of TSS. This integration helps to use the main advantages of the TSS platform more effectively. For example, it allows testing of ARM's next-generation cores in a simulation environment. It also makes it possible to collect ASTF data, which is useful for improving system performance. Moreover, using the RDN2 platform not only allows testing of ARM's next-generation cores but also provides access to ARM's debugging tools that supports more efficient validation process.

Considering all these benefits, using the RDN2 platform together with TSS clearly improves the overall functionality of the TSS system. In addition, the proposed system has several advantages when compared to using only the RDN2 platform. One of the key benefits is the flexible design of the TSS platform that also supports IP modules written with different programming languages. This makes the development process easier and more efficient for engineers.

The proposed approach gives a better validation process in compared to traditional SoC design methods. Earlier in the development cycle, it allows engineers to find and fix design errors. This helps to reduce delays and lowers the overall cost for design process. The successful results of the applied tests demonstrates that the concept of the integrated system functions correctly.

In conclusion, this system brings variety of influential advantages, such as increased flexibility, the latest ARM cores support, and compatibility with IP modules that are written in different programming languages. These features offer a novel and efficient solution for SoC design projects.

6 References

Accellera. (2009, July). *OSCI TLM-2.0 Language Reference Manual: Accellera Web site.* Retrieved May 15, 2025, from Accellera Web site: https://www.accellera.org

Aho, J. M. (2023). *Inter-Processor Communication in Virtualized Environment [Master's thesis, University of Oulu].* Oulu. Retrieved from https://urn.fi/URN:NBN:fi:oulu-202312143784

Andrews, J. R. (2007). *Co-verification of Hardware and Software for ARM SoC Design.* Newnes: sciencedirect. doi:10.1016/B978-075067730-1/50007-4.

ARM. (2021, May 21). *Arm Neoverse N2 Platform: A Significant Uplift in Cloud-to-Edge Performance Efficiency.* Retrieved from ARM Web site: https://www.arm.com

ARM. (2022, January 15). *Arm Neoverse N2 reference design Technical: ARM Developers.* Retrieved from ARM Developers Web site: https://developer.arm.com/

ARM. (2025, February 19). *Fast Models Reference Guide, ARM.* Retrieved May 14, 2025, from Developer ARM Web site: https://developer.arm.com/documentation/100964/1128

Bailey, B., Grant, M., & Andrew, P. (2007). *ESL design and verification : a prescription for electronic system-level methodology.* Boston: Morgan Kaufmann.

Chen, Y.-K., & Kung, S. (2007, July 17). Trend and Challenge on System-on-a-Chip Designs. *Journal of Signal Processing Systems, 53*(1-2), 217-229. doi:doi:10.1007/s11265-007-0129-7

Drechsler, R. (Ed.). (2004). *Advanced Formal Verification* (1 ed.). New York, NY: Springer . doi:10.1007/b105236

Ferrara, P., Vincenzo, A., & Agostino, C. (2024, August 30). Challenges of software verification: the past, the present, the future. *International Journal on Software Tools for Technology Transfer, 26*, 421-430. doi:https://doi.org/10.1007/s10009-024-00765-y

Karmitsa, V. (2023). *Pre-validation of SoC via Hardware and Software Cosimulation [Master's thesis, University of Oulu].* Oulu: University of Oulu.

Kern, C., & Greenstreet, M. R. (1999, April). Formal verification in hardware design: a survey. *ACM Trans. Des. Autom. Electron. Syst., 4*(2), 123–193. doi:10.1145/307988.307989

Koivula, M. (2023). *Virtual Prototype Based SoC Co-Verification [Master's thesis, University of Oulu].* Oulu: University of Oulu. Retrieved May 2, 2025, from https://urn.fi/URN:NBN:fi:oulu-202312153872

Krolnik, A., & Foster, H. (2008). *Creating Assertion-Based IP* (1 ed.). New York: Springer. doi:10.1007/978-0-387-68398-0

Krula, J. (2024). *A framework for early system-on-chip co-validation [Master's thesis, University of Tampere].* Tampere: University of Tampere. Retrieved from https://trepo.tuni.fi/handle/10024/158609

Kumar, A., Litterick , M., & Candido, S. (2024). Efficient Verification of a RADAR SoC Using Formal and Simulation-Based Methods. *ArXiv, abs/2404.15371*. Retrieved May 5, 2025, from https://arxiv.org/abs/2404.15371

Marcian Cirstea ,Khaled Benkrid, Andrei Dinu,Romeo Ghiriti, Dorin Petreus. (2024, February). Digital Electronic System-on-Chip Design: Methodologies, Tools, Evolution, and Trends. *Micromachines, 15*(2), 1-24. doi:https://doi.org/10.3390/mi15020247

Orlov, A., & Syschikov, A. (2012). High-level system-on-chip simulator. *11th Conference of Open Innovations Association (FRUCT)* (pp. 136-142). St. Petersburg, Russia: IEEE. doi:10.23919/FRUCT.2012.8253117

The QEMU Project Developers. (2025). *User Mode Emulation QEMU Org.* Retrieved May 14, 2025, from QEMU Org. Web site: https://www.qemu.org/