CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Energy Efficient Open-Source RISC-V Processor for Automotive Workloads

Master's thesis in Computer science and engineering

Fredrik Jansson, Luyao Chang

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Energy Efficient Open-Source RISC-V Processor for Automotive Workloads

Fredrik Jansson, Luyao Chang

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Energy Efficient Open-Source RISC-V Processor for Automotive Workloads
Fredrik Jansson, Luyao Chang

Supervisor: Ioannis Sourdis, Computer Science and Engineering
Co-supervisor: Ahsen Ejaz, Mehrzad Nejat, Computer Science and Engineering
Advisor: Nicholas Mucci, Volvo Cars
Examiner: Pedro Petersen Moura Trancoso, Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2025

Energy Efficient Open-Source RISC-V Processor for Automotive Workloads
Fredrik Jansson Luyao Chang
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Modern vehicles integrate a growing number of electronic control units to perform key tasks, such as driver assistance, smart braking, and steer by wire. This presents a challenge for the transition towards battery electric vehicles as any power consumed by embedded systems cannot be used for driving, directly reducing vehicle range. As a result, energy efficiency is of central concern as automotive software grows more complex. Another factor is ease of development, with closed systems requiring expensive and specialized software for development, making them expensive and difficult to maintain. Moreover, processors traditionally used in the automotive space are proprietary which enables vendor lock in. Exploring alternative processor architectures has great potential for meeting the evolving requirements. The open and flexible nature of RISC-V makes it particularly well-suited to address the performance, safety, and energy-efficiency challenges of modern automotive systems. The instruction set being open allows anyone to design a compatible processor which can enable competition and result in cheaper and more efficient designs. Additionally, the design can be tailored to the specific performance and power requirements of the application. Finally, software development can leverage open-source tools, reducing the need for expensive licenses. This thesis investigates the suitability of the NOEL-V, an open-source RISC-V processor, for use in automotive systems. It is compared to a commercial automotive ECU based on Infineon TriCore TC399XP in terms of performance and power consumption by porting a climate system to RISC-V and evaluating it on an FPGA implementation of NOEL-V. The performance is assessed through cycle count and schedulability of real-time tasks, while power consumption on a 45 nm process is estimated using EDA tools. For the ECU, the performance metrics of the TriCore are collected on the real hardware through Lauterbach debug tools, and power consumption is estimated indirectly via the system-level power increase during workload execution. The results show that although the NOEL-V platform does not achieve the same raw performance as the TC399XP, it is sufficient for the climate application and requires fewer clock cycles per instruction. The project also identifies major power hotspots in the RISC-V processor under the target workload, and proposes directions on optimizing power dissipation for future RISC-V-based automotive processors.

Keywords: RISC-V, NOEL-V, TriCore TC399XP, automotive ECU, performance, power consumption, real-time systems, FPGA implementation, automotive workloads.

Acknowledgements

We would like to express our sincere gratitude to Pedro Petersen Moura Trancoso, Ioannis Sourdis, Mehrzad Nejat, and Ahsen Ejaz for providing academic feedback throughout the thesis process. We also extend our special thanks to Nicholas Mucci, Johan Ankarberg, and Johan Lindquist Holmberg at Volvo Cars, whose technical guidance enhanced our understanding of ECU performance and energy efficiency. Furthermore, we appreciate the warm welcome and support provided by Volvo Cars for hosting our thesis work in their office.

Fredrik Jansson, Luyao Chang, Gothenburg, 2025-09-26

Contents

List of Figures

List of Tables

1

Introduction

The increasing demand for high-energy efficiency and real-time capable systems in the automotive industry has intensified the exploration of alternative processor architectures to meet strict requirements in performance, power consumption, and reliability. Modern vehicles rely on *Electronic control units* (ECUs) to enable critical functionalities across diverse systems. The rise of Battery Electric Vehicles (BEVs) presents a unique challenge in the field of automotive computing, as any energy that is consumed by the computer system cannot be used for propulsion, reducing the range of the vehicle. Meanwhile, these vehicles rely entirely on electronic control systems, meaning that the range, handling, and performance of the vehicle can be improved through the use of computer systems [1]. Finally, the importance of traditional metrics for embedded computer systems, such as performance, chip area, cost, and heat, does not simply disappear because the system happens to be located in a BEV.

Another transformative development in the field of embedded computing is the emergence of RISC-V, an open-source *Instruction Set Architecture* (ISA) based on reduced instruction set computer (RISC) principles. Unlike proprietary architectures such as Arm and TriCore, RISC-V allows anyone to make a compatible computer system without costly license acquisition. Further, the open nature of the RISC-V architecture allows manufacturers to tailor the system to specific applications. These benefits enable significant improvement in embedded and high-performance computing, making RISC-V a perfect candidate for addressing the challenges in automotive applications.

This thesis aims to explore the potential of the NOEL-V [2] RISC-V processor in the automotive domain with an emphasis on performance and power consumption, compared to an existing commercial ECU, Infineon TriCore. By evaluating the NOEL-V processor under automotive-specific workloads, this thesis work provides insights into the feasibility of integrating RISC-V-based processors into automotive systems and identify its requirements and challenges.

1.1 State-of-the-art

In recent years, the rapid development of the RISC-V architecture has attracted wide attention from academia and industry due to its openness, modularity, and low cost. Lazzeri et al. [3] systematically compared three high-performance RISC-V

processors (BOOM, NOEL-V, and CVA6), evaluating their performance, power, and area efficiency. The results showed that although NOEL-V performs worse than BOOM in terms of performance, it provides better power efficiency and smaller area usage, making it more suitable for embedded and resource-constrained systems. The study also pointed out that most RISC-V cores are still behind commercial Arm processors in handling complex workloads. However, the open and customizable nature of RISC-V makes it a promising option for future energy-efficient designs, including automotive systems. While this work provides a general performance overview of different RISC-V cores, including NOEL-V, the testing is done using general-purpose, synthetic benchmarks (Dhrystone, CoreMark, and CoreMark-pro). However, the performance characteristics of real-world automotive software may differ significantly. In addition, all processors were implemented on the same field-programmable gate array (FPGA) board, and each core's power consumption was estimated by measuring the board's overall power and calculating the difference between the baseline and the total usage. This indirect method lacks precision and may lead to misleading conclusions when selecting or optimizing processors for energy-sensitive applications.

To address power consumption issues, International Business Machines (IBM) researchers suggested that power modeling should start early in the microarchitecture design phase. They proposed combining module-level estimation with performance simulation to identify power hotspots [4]. This study analyzed power distribution across key blocks such as caches and registers, and emphasized techniques like clock gating and dynamic resource management. Different energy-efficiency metrics like power-delay product (PDP) and million instructions per second per watt (MIPS/W) were compared. These methods provide support for power optimization in embedded systems and guide the energy-efficiency modeling flow used in this project.

Chatzopoulos et al. [5] focused on analyzing the energy efficiency of the SonicBOOM RISC-V out-of-order processor. They used SimPoint to identify microarchitectural hotspots and implemented register-transfer level (RTL) power estimation using Chipyard, Cadence Joules, and the ASAP7 predictive technology library. Their results showed that branch predictors and instruction scheduler units are major contributors to power consumption, and evaluated performance-per-watt of several configurations under many workloads. This work provides useful methods for hotspot analysis and microarchitecture optimization in energy-sensitive applications. Again, these works consider the power and energy use of processor cores running general workloads.

Claudio et al. [6] built a mixed-criticality platform based on the CVA6 RISC-V core, running AUTomotive Open System ARchitecture (AUTOSAR) Classic and Linux together. Their work emphasized the integration of open-source hardware and software to create a multi-operating system (multi-OS) architecture suitable for automotive systems. Their work shows that, with software optimizations in the ERIKA real-time operating system (RTOS) and the integration of an open-source interrupt controller, the real-time performance of RISC-V in terms of interrupt handling and communication can approach that of the Arm Cortex-R5. This confirms that RISC-V has the potential to serve as the main controller in future real-time

and energy-efficient automotive ECUs.

However, their research focuses on highly specific OS level events, such as task activation latency, time to enter or exit the interrupt handler, and time to terminate a task [7]. Their research does not consider the target application the ECU will run which ultimately determines the performance requirements of the platform. What is missing in their work is the consideration of the application software that an automotive ECU typically runs, such as control or diagnostic algorithms, which are more representative of real-world use cases. Evaluating performance at the application level is essential to determine whether the target MCU is suitable for practical deployment in automotive systems.

Although many studies have evaluated the performance and power characteristics of RISC-V processors, it remains unclear how these processors behave under real-world automotive workloads. To the best of our knowledge, existing evaluations are predominantly based on general-purpose benchmarks like CoreMark and Dhrystone, which cannot reflect the realistic behavior of automotive workloads. The workloads of interest in this study, representative of Volvo Cars' applications, typically consist of tight control loops and heavily utilize floating-point operations, resulting in different performance characteristics than general-purpose applications. Additionally, in the case of Volvo Cars, the source code for the ECU software is generated from Simulink models rather than written by a human programmer, which is another potential source of performance differences. Using such general-purpose benchmarks then risks selecting suboptimal processors, as the importance of microarchitectural features may be misjudged.

Furthermore, to the best of our knowledge, no prior work has been conducted on a cross-platform comparison between commercial ECUs like Infineon TriCore and open-source RISC-V cores (NOEL-V) under the same workload. Differences in architectures, ISA implementation, and technology libraries create uncertainty on whether RISC-V processors could be a good replacement for commercial ECU cores. Therefore, such comparisons are crucial for evaluating their suitability in automotive domains.

1.2 Purpose and Goal

This thesis bridges the gap between RISC-V's theoretical potential and automotive industry requirements, aiming to validate RISC-V's suitability for automotive systems. In particular, the contributions of this work are as follows:

- We measure the performance and power consumption of TriCore under automotive-specific workloads.
- We implement the NOEL-V processor on an FPGA, and compare its performance to the TriCore ECU under the same workloads.
- We identify the power hotspots of NOEL-V through gate-level power analysis using a comparable 45nm technology, and propose optimization directions.

- We analyze the performance characteristics of the workload to determine the performance required to run the workload, identify how different microarchitecture features affect performance, and assess scheduling feasibility.

1.3 Thesis Outline

The remainder of this thesis report is organized as follows. Chapter II provides some background on two platforms, NOEL-V and ECU (Infineon TriCore), and introduces power estimation fundamentals for the application-specific integrated circuit (ASIC) design flow. Chapter III presents the system architecture and design choices (platforms, software stacks, workload selection, and implementation). Chapter IV details the measurement framework and metrics of performance and power estimation on both platforms. Chapter V compares the performance results collected from two platforms and analyzes the power hotspots of the NOEL-V processor. Chapter VI discusses the results, including performance evaluation, limitations of the power analysis, and potential optimization directions. Chapter VII draws the main conclusions and discusses future work.

2

Background

This chapter introduces the fundamental knowledge and technical context for this thesis. First, it presents an overview of an open-source NOEL-V processor, which serves as the RISC-V implementation evaluated in this project. Next, it details the ECU in automotive systems, focusing on software architecture, safety requirements, hardware architecture, and some common debug tools. Finally, it outlines power estimation methodology, including sources of power consumption, the estimation flow in ASIC design, and the role of technology libraries and electronic design automation (EDA) tools. This background lays a solid foundation for the experimental work in later chapters.

2.1 Benefits of RISC-V

The main benefit of RISC-V is that it is an open standard, allowing anyone to develop a compatible system free of cost. Today, the automotive industry primarily relies on proprietary solutions for their ECU's creating vendor lock-in where a cheaper or more performant system from a competitor cannot be used. This results in more expensive ECUs and thus ultimately cars, and to make matters worse, these proprietary systems also require proprietary compilers from the same suppliers. These compilers then have to be supported for the lifetime of the produced car, which creates an opportunity for the vendor to extract rent for years after the car has been produced and sold. Furthermore, an ECU may require a specific version of the compiler, and as modern cars contain many different kinds of ECU's the manufacturer then has to license compilers for all of these.

The openness of RISC-V circumvents this type of lock-in as different vendors may compete on both hardware and software, which can enable large cost savings.

This openness also brings flexibility. With an open design, the central processing unit (CPU) can be tailored to meet the demands of the application, which can further reduce cost.

2.2 NOEL-V

NOEL-V is an implementation of a RISC-V system by Frontgrade Gaisler. It is a highly modular and configurable processor system with a dual-issue, 7-stage pipeline,

support for RISC-V 64-bit (RV64) and 32-bit (RV32), and an optional, area-efficient floating-point unit (FPU). NOEL-V is available for free under the GNU General Public License (GPL) or a commercial license with additional features such as a higher performance FPU and fault tolerance. It is designed to be a relatively high-performance system for aerospace.

The NOEL-V processor uses the Advanced Microcontroller Bus Architecture (AMBA) 2.0 Advanced High-performance Bus (AHB) and can optionally include a level-2 cache and an Advanced eXtensible Interface 4 (AXI4) backend. It supports plug-and-play integration of hardware components through Gaisler's GRLIB intellectual property (IP) library, facilitating rapid integration of automotive peripherals. Designed for flexibility, it can be implemented efficiently on both FPGA and ASIC platforms [8], using standard memory cells for caches and registers.

2.3 ECU in automotive systems

An ECU is an embedded computer system that controls electrical systems in the car. A modern car can contain as many as 100 ECUs, which enable significant functionality such as emergency braking, electronic stability control, and automatic cruise control. As embedded computer systems, ECUs need to be efficient with respect to power and cost, but also have an important safety perspective. If an error occurs in an ECU, it could in the worst case result in injury or death, meaning that rigorous testing has to be done in order to minimize residual risk.

2.3.1 Software

ECUs typically run an AUTOSAR stack. AUTOSAR is a layered software architecture that is commonly used in the automotive industry. In the AUTOSAR model, the software is divided into three layers: basic software (BSW), runtime environment (RTE), and application. The basic software is the hardware-dependent layer and provides drivers and hardware abstractions. The RTE provides a real-time operating system and a hardware-independent interface enabling the development of application software components without knowledge of the underlying ECU.

AUTOSAR also defines the AUTOSAR XML (ARXML) format. This eXtensible Markup Language (XML) schema describes the software component (SWC) interface to the system and provides a complete description of how to integrate the component with the rest of the system. The application software consists of runnable entities, which are executable sections of code, and events that trigger the execution of such entities. There are many different types of events, but the most important ones for this project are Mode switch events and timing events. Mode switch events are used to trigger execution when the component switches modes, for example from inactive to active, while timing events enable periodic tasks. In AUTOSAR, software components use ports to communicate with other components. There are 2 kinds of ports: Required and Provided ports. Required ports represent input data and are required for the application to function. Provided ports, on the other hand, represent

output data and thus provide data that may be required by another component. The AUTOSAR standard does not describe how these ports should be implemented.

2.3.2 Requirements

The requirements of the system can broadly be divided into 2 categories: performance requirements and feature requirements. Feature requirements describe what is required for the system to work as expected. In automotive computing systems, safety is a critical aspect of feature requirements and is defined by the Automotive Safety Integrity Level *ASIL* specification.

The ASIL process assesses events in 3 different categories: severity, exposure, and controllability. The severity of an event refers to what injuries the event may cause, with the lowest severity level S0 expected to cause no injuries, while an event in the highest severity class S3 may be life-threatening. Exposure level refers to how likely an event is to occur, while controllability refers to how likely a driver is to be able to control the event should it occur. Taken together, these factors define what integrity level the system must conform to. The integrity levels start at ASIL-D, which is the combination of maximum severity, exposure, and controllability, down to ASIL-A. Additionally, there is the Quality Management (QM) classification for systems that have no safety relevance. These levels then dictate the safety requirements of the system and facilitate the development of safe hardware and software systems, as suppliers can certify their components to a certain level which automatically makes it comply with the lower levels too.

2.3.3 Real-time system

An automotive application consists of real-time tasks. Real-time tasks are tasks whose correctness depends not only on its result but also on what time that result is produced. Real-time applications are often modeled as a set of periodic tasks where each task is defined by a deadline, at which point the execution must have completed, and a period, when the next task instance arrives. Deadlines are frequently implicit, meaning they are equal to the period, and in that case *Rate monotonic scheduling* produces the optimal, static schedule. This assigns priority to the tasks based on their periods, with shorter periods given higher priority. Then, it has been shown that all tasks in the set are schedulable if condition 2.1 holds

$$n(2^{1/n} - 1) \geq \sum_{i}^{n} \frac{C_i}{T_i} \tag{2.1}$$

where n is the number of tasks in the set, C their cost, and T their period. Because of its optimality and simplicity, rate monotonic scheduling of real-time tasks is prevalent in the automotive industry.

2.3.4 Performance model

To understand the requirements of the workload, a performance model is needed. The ultimate metric of interest is execution time T, which is broken down into its

contributing factors. The most fundamental performance model is $T = T_c * CPI * I$ where T_c is the clock period, I number of instructions in the program, and CPI is the average number of cycles per instruction. This simple model is complicated by the fact that not every instruction takes the same number of cycles to execute. In a single issue pipelined processor, instructions have a maximum throughput of one instruction per cycle, and the relationship can be rewritten as $T = T_c * (I + S)$ where S is the number of stall cycles. Pipeline stalls arise when an instruction needs to wait for a value before it can continue executing. This can happen when an instruction references memory that has to be retrieved or if it depends on the result of another instruction, for example a branch. The S term is expanded to distinguish what events cause stall cycles and to separate the number of event occurrences from the cost of each event. Total execution time can then be modeled as

$$T = T_c * (I + \sum_{e \in E} (N_e * C_e))$$

where E is the space of all events that can cause stall cycles in the model, N_e is the number of times event e occurs, and C_e the number of stall cycles caused by that event. The value of these parameters depends on the processor architecture and design while the cost of cache misses also depends on the speed of the memory being used.

2.3.5 Hardware architecture of ECU

ECUs typically consist of a microcontroller, on-chip memory, peripheral interfaces, and communication modules. They are designed to meet real-time performance, safety, and cost requirements. The core is a microcontroller unit (MCU), which executes control logic and processes data from various components. Depending on the application, the processor can be 16-bit, 32-bit, or 64-bit. For example, body control modules, which manage vehicle functions such as lighting and door locks, often use simple 16-bit MCUs, whereas autonomous systems require more powerful 32-bit or 64-bit processors. The memory system typically includes flash memory for storing programs, and static random-access memory (SRAM) for runtime data.

ECUs integrate various communication interfaces such as Controller Area Network (CAN), Local Interconnect Network (LIN), and Ethernet to connect with other ECUs in the vehicle. These interfaces are managed by specific communication controllers to reduce the load on the main processor. To meet real-time and safety requirements, hardware modules include watchdog timers and interrupt controllers. AUTOSAR-based systems rely on the Microcontroller Abstraction Layer (MCAL) to manage hardware resources and provide a standardized interface to the upper software layers [9].

In general, the hardware and software of the ECU are closely related, which enables the design to meet requirements related to safety, real-time, and automotive environmental constraints. The next section introduces the TriCore TC399 from Infineon, a typical example of an automotive processor.

2.3.6 TriCore TC399XP

TriCore TC399XP is a high-performance automotive microcontroller from the Infineon AURIX TC3xx series. It uses a multi-core architecture with up to six 32-bit cores running at up to 300MHz [10]. The chip meets the International Organization for Standardization (ISO) 26262 ASIL-D safety standard and supports features such as encryption, authentication, and secure boot. TC399 also includes many communication interfaces, such as multiple CAN with Flexible Data-rate (CAN FD) channels and Ethernet. Equipped with high-speed analog-to-digital converters (ADCs) and a dedicated signal processing unit, it demonstrates strong performance in demanding automotive control scenarios.

2.3.7 Debug tools

Lauterbach is a leading global manufacturer of microprocessor development tools for embedded designs. Its TRACE32 tools provide a widely used debugging solution in the embedded system environment, which consists of PowerView as a universal graphical user interface and PowerDebug tools for debugging [11]. One representative product, PowerDebug PRO, is widely used in industrial and automotive embedded systems for its powerful debugging capabilities.

TRACE32 tools are compatible with more than 80 widely used microprocessor architectures, such as TriCore, Arm, PowerPC, and RISC-V. Lauterbach hardware tools follow a modular design, consisting of a standard base module and an architecture-specific cable. The base module works with all architectures, while the specific cable connects to the target device. This allows switching to a new processor architecture by easily changing the cable.

Most modern cores include a debug port that allows tools to connect and control the processor, access its data, and perform debugging operations, including single-stepping, setting breakpoints, tracking variable values, triggering real-time events, and reading registers and memory. This helps developers identify software bugs and memory issues and correct the system to ensure expected behaviors.

2.4 Power modeling fundamentals

Power is a key concern in automotive systems because it directly affects energy use and system stability, so an accurate and consistent power estimation method is necessary. This section outlines the power consumption sources, procedures, and tools during workload execution.

2.4.1 Sources of power dissipation

Power consumption in complementary metal–oxide–semiconductor (CMOS)-based processors consists of dynamic and static components. Dynamic power results from switching activities, while static power is due to leakage currents in transistors that

are turned off. Both types are influenced by the processor's architecture, the executed workload, and process technology.

The switching power, P_{sw}, consistently occurs due to the charges transferring between the power supply and ground to charge and discharge a load capacitance [12]. For a netlist containing N logic cells with multiple internal nodes, the equation is

$$P_{\text{sw}} = f{V_{DD}}^2 \sum_{i=1}^{N} (C_i \alpha_i).$$

Here, f is the clock frequency, V_{DD} is the supply voltage, and C_i, α_i are the capacitance and switching activity, respectively, of the i-th node. The capacitance C_i of the node is the sum of the gate, diffusion, and wire capacitance on the node. The activity factor α_i is the probability that the circuit node transitions from 0 to 1 because that is the only time the circuit consumes dynamic power. Hence, dynamic power dominates in high-performance circuits due to its dependence on frequency and voltage. The switching activity factor is typically extracted from Value Change Dump (VCD) files and varies depending on the workloads.

Static power, also known as leakage power, is dissipated even when a chip remains inactive. Typically, subthreshold leakage is the main source of static power.

Additionally, short-circuit power is dissipated during signal transitions when both P-channel and N-channel metal–oxide–semiconductor (PMOS and NMOS) transistors conduct simultaneously. However, it contributes very little, less than 5% of the total power.

2.4.2 Power estimation flow in ASIC design process

Power estimation is performed at multiple stages of the ASIC design flow, with accuracy improving as the design moves from higher-level RTL descriptions toward detailed physical implementation. At the gate level, power analysis relies on synthesized netlists and switching activity obtained from simulation, providing a balance between accuracy and computational cost. Fig. 2.1 illustrates the estimation flow at this stage.

Functional verification

After completing the RTL design, functional verification is needed to ensure it behaves correctly. For example, if a processor is supposed to issue bus transactions to fetch instructions from memory, this behavior must be checked at this point through verification [13]. This is done with the help of EDA simulators that are used to model the design and apply different stimuli to it. A testbench and test cases are developed to test the functionality of the RTL code. If any test fails, it might indicate a design bug, so the design team can fix it before retesting until there is a good level of confidence in the functional correctness of the design.

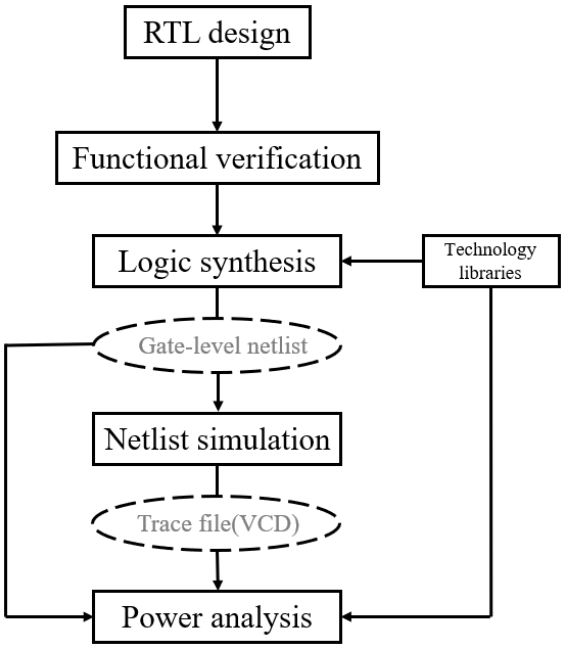

Figure 2.1: ASIC power estimation flow at the gate-level.

Logic synthesis

Once the hardware description language (HDL) code is functionally verified, it is synthesized into a gate-level netlist composed of actual hardware elements such as logic gates and flip-flops. This step is known as synthesis. Logic synthesis tools enable the conversion of RTL description in HDL to a gate-level netlist by technology mapping [14]. This netlist represents a description of the circuit using gates and connections. The tools also generate reports on key factors such as timing, area, and power, allowing designers to review and make necessary changes before the creation process. Early corrections at this stage help save time, cost, and effort.

The synthesis process typically involves three steps: translation, mapping, and optimization. Translation is the way to convert HDL code, such as Verilog or VHDL, into a structural representation. During mapping, structural design, or generic cells could be mapped to technology-independent cells that contain functional and timing details. After that, the design gets optimized to meet specific specifications in the optimization step. Here, technology libraries offer standard cells, gates, and other components tailored to the target process technology.

Power analysis

Then, power analysis is performed to evaluate the power dissipation of the circuit. Simulation-based power analysis is employed to estimate switching activity, which can be accurately captured through netlist simulation. Once the simulation is completed, a VCD file [15] was generated detailing toggle events for each clock cycle. This data can be visualized using waveform viewers, which allow designers to analyze the behavior of their digital designs. It can also be used to obtain the actual power consumption. The size of the trace file depends on the workload, which ranges from a

few Megabytes to hundreds of Gigabytes of data. Finally, the power estimation flow is performed by introducing the mapped netlist in conjunction with the transition of each signal to determine the switching, static, and total power consumption of the design.

2.4.3 Impact of technology libraries

As mentioned earlier, logic synthesis tools can work with different technology node processes and libraries to meet specific design goals. These libraries, provided by semiconductor foundries, include detailed data like rise/fall times for flip-flops and gate input/output delays, and significantly influence the power estimation.

There are several considerations for technology libraries. For process node scaling, smaller technology nodes reduce dynamic power through lower C_i and V_{DD}, but leakage increases due to shorter channel lengths, while larger technology nodes exhibit higher dynamic power but better leakage control. Another aspect is library characterization. This means cells are characterized for delay, power, and area across PVT (Process, Voltage, Temperature) corners [16]. Multi-voltage libraries enable power gating and dynamic voltage scaling [17]. Additionally, academic process design kits (PDKs) provide simplified models for research but may not match the accuracy of production-grade cases. In contrast, commercial libraries contain foundry-validated data but typically require a license to access.

GPDK045 is a 45nm generic PDK provided by Cadence. As a demonstration PDK, it is widely used in education, academic research, and EDA tool validation. The kit is based on a simulated 45nm CMOS process and includes a complete standard cell library and technology files. The standard cells cover basic logic gates such as INV, NAND, NOR, and DFF, and the technology files define rules for design rule checking (DRC), layout versus schematic (LVS), and parasitic extraction (EXTRACT). Key library files such as `.lib` (for synthesis and timing analysis), `.lef` (for layout abstraction), and `.db` (compiled libraries) are also provided.

GPDK045 is compatible with the full Cadence toolchain, including Virtuoso for layout, Genus for synthesis, Innovus for place and route, and Spectre for simulation, supporting a complete RTL-to-Graphic Data System II (GDSII) flow. However, it is not provided by a real semiconductor foundry, so it cannot be used for actual chip fabrication.

2.4.4 EDA tools

Power estimation relies on a suite of EDA tools, which are software tools used in chip design. These tools streamline the electronic design process, including various tasks like schematic capture, simulation, layout, and verification. This toolchain ensures consistent power metrics throughout the design flow, ensuring designers meet performance goals and can reliably manufacture them. Each EDA tool serves a specific purpose in the ASIC flow, with synthesis tools and simulators commonly employed at distinct stages within the electronic design process. In the following, we list two commonly used EDA tools.

- NCsim [18] is a Verilog simulator from Cadence, used to verify the correct timing behavior and logic functionality of a digital circuit design, ensuring that it meets specifications and works properly.
- Genus [19], also by Cadence, is a synthesis tool that converts RTL code into equivalent gate-level netlists. It handles design constraints to ensure that circuits meet design specifications and optimizes the netlists to improve circuit metrics such as performance, power, and area.

3

Design

This chapter presents the system architecture and design choices of this project. The TriCore TC399XP, as a representative commercial automotive core, is introduced to provide a baseline for platform comparison with the open-source NOEL-V processor. To ensure a fair evaluation, the same automotive workload is selected and adapted for both platforms. The chapter covers the system overview and the design decisions made for porting the software and execution environment.

3.1 Overview of system design

The main purpose is to evaluate the suitability of the NOEL-V processor for real-world automotive workload. The design motivation is driven by providing a realistic and comparable environment between an open-source RISC-V processor and a commercial automotive ECU. The two platforms represent state-of-practice and state-of-research in automotive domains, respectively.

On the Volvo Cars side, the Climate control application was selected as the software to be ported and evaluated. This system was chosen because it is a relatively simple and self-contained module that does not communicate much with other applications but still does useful work. This ensured the workload was both realistic and simple enough to port and analyze. The target application consists of periodic tasks that are scheduled and executed on an AUTOSAR stack, which runs on an Infineon TC399XP ECU.

Considering the significant difference in terms of architecture and software between NOEL-V and TriCore ECU, several key determinations were made. On the NOEL-V platform, we replaced AUTOSAR with Real-Time Executive for Multiprocessor Systems (RTEMS) to implement periodic scheduling and enable task monitoring. On the TriCore side, we isolated application logic when obtaining performance metrics, decreasing variability between the platforms.

To make the evaluation fair and reproducible, the system was designed to align the control task structure, periodic scheduling, workload, and metrics across both platforms. Table 3.1 summarizes two platforms in terms of workload, software stack, and methods later used for performance and power consumption evaluation. A detailed description of these evaluation methods is given in Chapter 4.

The following sections detail the design choices of both platforms.

Table 3.1: Comparison of the two platforms.

Platform	Workload	Software stack	Performance evaluation method	Power evaluation method
TC399XP ECU (TriCore)	Climate control application	AUTOSAR	TRACE32 (hardware trace interface)	Current-delta power measurement
NOEL-V (RISC-V)	Climate control application	RTEMS	Cycle counter (FPGA implementation)	Gate-level power modeling (ASIC design flow)

3.2 ECU platform

This section describes the baseline commercial automotive ECU, Infineon TriCore TC399XP. This section covers the hardware architecture of TC399XP and the software environment in which the automotive workload is executed. To focus the evaluation, one ECU containing its associated application is selected from the large set of controllers used in Volvo Cars.

3.2.1 Software

First, the software component to be investigated was decided. The SPA2 platform that modern Volvo Cars are built on contains many ECUs and not all of them can be investigated. Many ECUs also employ a *hotel* like approach, where multiple unrelated software components are assigned to execute on the same ECU in order to save on cost. In order to simplify the comparison, this study isolates one software component. For the purpose of this study, the Front input/output controller (*FIOC*) is chosen. Among other components, this one contains the climate system of the car and that will be the focus of this study.

3.2.1.1 Application model

The TC399XP is a 6-core CPU that runs an AUTOSAR stack. The first 4 of these cores are lockstep cores and each of them is assigned one application to execute, independent of the other cores. Each of these applications consists of a set of periodic tasks that are executed using rate monotonic scheduling.

This study evaluates the climate control system. This system is responsible for controlling temperature and humidity in the car and is defined in Simulink models that are then used to generate AUTOSAR-compliant C code. That code is then built using the proprietary Tasking compiler along with AUTOSAR BSW and RTE stacks to create the final application. The generated software consists of 3 periodic tasks, each with a period of 10 milliseconds, and is a relatively simple and self-contained system. Task 0 in this system reads configuration data and calculates a bit mask that is then used by the other tasks. Task 1 reads sensor values and calculates input data to task 3, which finally writes data to maintain the desired temperature and humidity in the car.

The different software components the application consists of, their required and provided ports, what events may occur in the system, and what code block should be executed on those events are described through ARXML files.

The FIOC application is built using a proprietary TriCore compiler and relies on the AUTOSAR stack to run. In addition to ports and timing events, the software also contains operation-invoked events. These events are used in client-server communication to trigger the execution of runnable code in the server.

3.3 NOEL-V platform

This section introduces the NOEL-V processor implemented on an FPGA board, including the hardware implementation, the software adaptation for automotive workload execution, and the chosen power estimation approach.

3.3.1 FPGA implementation of NOEL-V

To evaluate the NOEL-V processor in a real hardware environment, it is implemented on an FPGA platform. The bitstream generated from RTL code is programmed onto the Digilent ARTY-A7-100T development board [20].

This board is based on an Xilinx Artix-A7 FPGA, and is widely utilized in academic and prototyping projects because of its affordability and compatibility with a wide range of commonly used toolchains. It offers sufficient hardware resources, such as memory blocks and input/output (I/O) interfaces for system-on-chip (SoC) designs, and also supports the implementation of RISC-V soft cores along with Universal Asynchronous Receiver/Transmitter (UART) and other peripherals. These features make it a suitable platform for performance testing.

Using this FPGA-based implementation, NOEL-V can be interacted with at runtime, enabling the collection of performance metrics such as task execution time and cycles under automotive workloads. Details of the performance measurement and evaluation method are presented in Section 4.4.

3.3.2 Platform adaptation and workload porting

Since AUTOSAR and its associated compiler toolchain are not available for RISC-V, these components need to be replaced. In this project, GNU Compiler Collection (GCC) is used to compile the software, and RTEMS is used to achieve periodic execution. RTEMS is another embedded operating system that supports real-time tasks. For each software component, one RTEMS task is created. RTEMS rate monotonic scheduling is then used to schedule execution of the application tasks. The RTEMS system then monitors the execution of the application tasks and collects statistics on any deadlines that have been missed.

3.3.3 Power estimation approach

For NOEL-V, direct power measurement on the FPGA is not representative, because most of the power on an FPGA comes from the programmable logic and interconnect overhead, rather than from the processor core itself. Moreover, the architecture and

technology of an FPGA are very different from those of a target ASIC, which makes the results less meaningful for real silicon.

To obtain more accurate estimates, an ASIC design flow was chosen. In this way, technology libraries can be used to evaluate both dynamic and static power at the gate level, providing results closer to a real implementation. In automotive, many MCUs or ECUs still use mature nodes (from 40 nm to 28 nm) for reliability, high-temperature operation, and long lifetime. We use a 45 nm PDK as a process equivalent to the target ECU, which is around 40 nm. Using a neighboring node keeps the device type and supply-voltage range similar, so the comparison is fair. Although the library is not the same as an industrial automotive process, it still enables consistent comparison and reveals possible power consumption trends. Therefore, using 45 nm is close to the target ECU technology and does not affect our bottleneck analysis or the optimization direction. This study does not involve actual fabrication but aims to explore the potential of NOEL-V in automotive scenarios. Therefore, the ASIC flow was selected as a balanced approach. The detailed steps are described in Section 4.5.

4

Experimental methodology

This chapter specifies the measurement framework and metrics used for evaluating the performance and power consumption of both platforms. Performance and power are assessed separately on the ECU and NOEL-V, each with methods suitable for the platform.

4.1 Metrics

To evaluate the workload execution across different platforms, a set of performance and power metrics was defined.

The primary performance metric is Cycles Per Instruction (CPI) which captures the average number of cycles required to execute a single instruction, and cycles, which is the total number of cycles required to complete the task. CPI reflects execution efficiency and is used throughout the evaluation to compare the NOEL-V processor with the TriCore ECU. Taken together with the implementation dependent frequency and memory technology, these metrics can then be used to determine execution time.

Additionally, on the NOEL-V platform, the performance metric Schedulability is assessed via missed deadlines using the RTEMS kernel monitor. This reflects whether each task can be completed within its assigned period.

Due to architectural and tooling differences, the power consumption of the two platforms is evaluated using distinct methods and at different abstraction levels, making direct comparison infeasible.

The ECU board does not provide access to internal power rails. Therefore, the entire system's power consumption is estimated by measuring the current difference between idle and active states, under a fixed 13.0V supply. This method provides a practical view of the overall power consumption during workload execution.

For NOEL-V, gate-level power estimation is performed using EDA tools, and power is broken down by hardware modules. This approach captures only the CPU's power and does not account for board-level or peripheral components. It enables analysis of power hotspots.

These metrics form the basis of the evaluation presented in Chapter 5, where the execution behavior and power characteristics of both platforms are compared.

4.2 Performance measurement on ECU

Performance of the ECU is evaluated using the Lauterbach LA-3707 debugger and accompanying TRACE32 software, the hardware setup illustrated in Fig. 4.1.

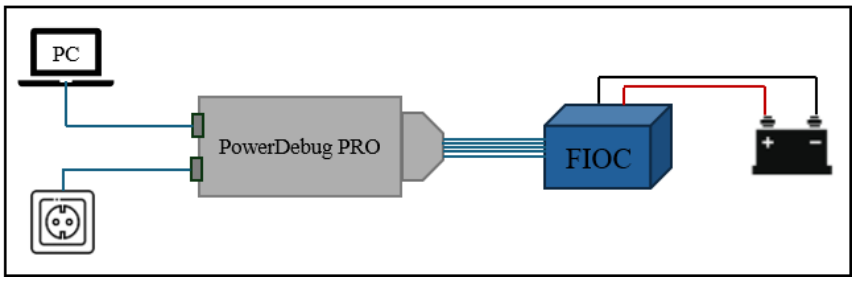

Figure 4.1: Hardware setup for ECU performance measurement. Lauterbach PowerDebug PRO connected to the host PC and the FIOC board (Infineon TC399XP ECU), providing access to the hardware trace interface for measurement of cycle counts and program execution.

First, the ECU system is booted and starts to periodically execute its application tasks. Then the debugger is attached to the session which inhibits execution. Breakpoints are then set at the entry and exit points of the task to be measured. When execution is resumed, the runtime system invokes all periodic tasks, including the one of interest. The debugger then counts all performance events from the first breakpoint being hit until the second one is reached. From this data, which includes the total number of clock cycles and the number of instructions, the performance of the task is determined. However, this approach has a major shortcoming in that it includes the impact of sub-function calls. That is a problem because the runtime of those involves calls into the AUTOSAR stack, which will not run on the NOEL-V and thus cannot be evaluated. They also include communication with other components and reading sensor data, which will not be implemented as part of this study.

To create an accurate performance comparison between the systems, these sub-function calls are isolated to focus on the core application logic. The Lauterbach debugger used does not support this type of tracing without additional hardware. For that reason, the impact of those sub-functions is determined manually, by first identifying all such function calls and then evaluating and determining their performance using the same method. The performance events from these measurements are then subtracted from that of the task of interest.

This method still has a flaw. Because the Lauterbach environment only allows one benchmark counter to be active at any time, the benchmarks of the sub-functions will come from different runs than that of the main task. This is alleviated by sampling each task and sub-function 50 times and comparing the averages, but it does reduce the precision of the benchmark.

4.3 Power measurement on ECU

To make a fair comparison of the core power consumption to that of NOEL-V, we need to get access to the chip-level voltage and current. Because there are no accessible chip power rails, we can just measure the system-level of the ECU during workload execution. Therefore, a practical method is to record the difference in total current between idle and active states. Since the power supply is stable at $V = 13.0V$, the power increase is calculated by the formula: $P_{ECU} = P_{ECU-active} - P_{ECU-idle} = V \times (I_{active} - I_{idle})$.

4.4 Performance measurement on NOEL-V

To determine the suitability of using an NOEL-V-based platform, its performance is evaluated. This is done with respect to 2 metrics: execution time and schedulability. Schedulability refers to whether it is possible to schedule all tasks in the system such that none of them miss any deadlines. The analysis can either be performed theoretically or empirically by applying a scheduling algorithm and running the system. Real-time scheduling in uniprocessor systems based on task utilization is a well-studied field. It consists of first determining the worst-case execution time of all the different tasks in the system and then applying a simple formula depending on the scheduling algorithm used. This approach does not work well with modern computing systems however. Microarchitectural improvements to processor design, such as caches, advanced branch predictors, and issuing multiple instructions per cycle has greatly improved average performance while affecting worst case performance significantly less, meaning that there can now be orders of magnitude difference between them. As a result, analysis based on worst-case execution time is highly pessimistic and demands much more powerful computer systems than would be necessary in any real-world scenario. Because of this, execution time is instead measured empirically in this study.

To facilitate performance measurements, an FPGA-based implementation of the NOEL-V is used. Because this implementation will have a different clock frequency than an ASIC implemented in equivalent technology to the TC399, the total number of clock cycles is used together with the frequency obtained from simulating the ASIC to get the total execution time. This method is still not perfect however, as the performance ratio between CPU and memory may be different in an ASIC implementation. This is accounted for by counting the total number of memory accesses and adjusting by the access time of a typical memory system.

The measurements are performed by building RTEMS with the climate application for NOEL-V and executing it for a long period of time. For each period task, the RTEMS system monitors every invocation of the task, every missed deadline (if any), and processor utilization (average and worst case). This data is then used to analyze schedulability. This project assumes rate monotonic scheduling, as that is a simple and well-studied scheduling algorithm for uniprocessor systems that is commonly used thanks to its simplicity and sufficient performance. Rate monotonic scheduling

Table 4.1: Monitored events on NOEL-V.

Cycles
Instructions
Single issue
Dual issue
Branches
Branch misses
Data cache accesses
Data cache misses

can also be used for partitioned scheduling of multiprocessor systems, allowing this analysis to scale beyond the systems considered here.

In addition to schedulability, execution time is also measured for each task in the application in order to identify performance bottlenecks. Instead of scheduling each task is now executed a large number of times and its execution time recorded. Additionally, hardware performance counters are utilized to measure the number of cycles each task takes, as well as what type of instructions contribute the most to the total execution time. While the schedulability analysis reveals if the application can be scheduled on the NOEL-V, this data establishes the performance characteristics of the application on this CPU and thus why it is more or less suitable than the TC399, and what is required of a RISC-V core in automotive. The monitored events are presented in Table 4.1.

In addition to these events, the number of cycles spent in the FPU is of interest as the climate application makes extensive use of floating-point arithmetic. As the NOEL-V does not have hardware support for collecting FPU events, this is achieved by replacing all floating-point operations with integer equivalents, and measuring the difference in performance.

4.5 Power analysis of NOEL-V

To quantify the power consumption of the NOEL-V processor under a realistic automotive workload, we followed a simulation-based power estimation method using the Cadence Genus synthesis tool and GPDK45 technology library, ensuring accurate modeling of dynamic and static power dissipation. The power consumption is estimated using an ASIC design flow at the gate-level, as illustrated in Fig. 4.2. It starts with RTL source code, which is synthesized using the Cadence tool with the PDK files. Some TCL scripts are used to automate this flow. During post-synthesis netlist simulation, the signal switching activity is saved into a VCD file, which is then used to estimate power consumption under workload execution. Finally, a power report is generated, which shows how the power consumption is distributed in the design.

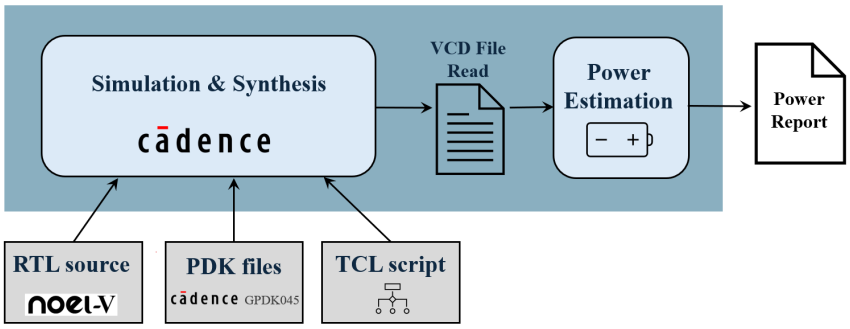

Figure 4.2: The ASIC workflow of the power analysis at the gate-level.

Technology library setup

The design is synthesized using the GPDK045 PDK library files, including Liberty, LEF, and QRC, which represent a generic 45nm CMOS process. The standard cell libraries contain a variety of basic gate types such as INV, NAND, NOR, DFF, and are characterized under fast process corner at a nominal voltage of 1.0V and temperature of 25°C. These libraries contain timing, area, and power information required for technology mapping and accurate gate-level modeling.

Due to the need to validate the RTL-to-gate-level netlist implementation flow and power analysis methodology, and considering the restricted access to commercial foundry PDKs, GPDK045 is chosen as the platform. As the target ECU's processor is fabricated in a 40nm technology node, the 45nm GPDK045 provides a comparable process, serving as a reasonable approximation for evaluating performance and power consumption under similar technology conditions.

RTL elaboration and netlist preparation

The RTL design is elaborated through Cadence Genus with the top-level module `noelvmp`, including all necessary GRLIB and platform-specific RTL files. A synthesized clock constraint with 10000 ps, corresponding to a 100 MHz clock, is applied. This value was chosen as a trade-off between synthesis runtime and timing closure. When using a higher target frequency 200 MHz, the slack becomes very small, and the synthesis process is significantly more time-consuming. By setting the constraint to 100 MHz, the synthesis finishes faster while still meeting the clock constraint. Subsequently, the design was mapped to standard cells from the GPDK045 process to generate a gate-level netlist file. Verilog modules of the technology library were included to enable accurate estimation.

Switching activity capture and power estimation

A simulation-based method is adopted to extract switching activity for power analysis through RTL simulation using the Cadence NCsim simulator. The simulation was used not only to verify the functional correctness, but also to execute a workload cross-compiled from real-world automotive climate control software into an S-record (SREC) file named `ram.srec`, loaded into the processor's random-access memory

(RAM) via testbench.

The `ram.srec` file follows the Motorola S-record format, and it encodes the binary program using hexadecimal representation and specifies the type of record, the address at which the data field is to be loaded into the memory, and memory loadable data or representative information [21]. During simulation, these records are parsed by the testbench and directly written into the RAM, allowing the processor to execute the pre-compiled automotive workload.

To capture switching activity, a VCD file was generated, which will then be used in the Cadence Genus synthesis tool for switching activity annotation. In the simulation configuration, the timescale is set to nanoseconds, allowing for precise signal tracing. Signal probing depth set to all under the `cpu` hierarchy to cover all signal switching. Simulation runtime of 57 ms covers several periods of task execution in a control loop. The resulting VCD file, later imported into Genus through the command `read_vcd`, provided detailed transition information relevant to instruction execution, control flow, and floating-point operations.

We feed the synthesized netlist file, PDK files, and the VCD file annotating the toggling activity to Genus, along with the Tool Command Language (TCL) script that orchestrates the power estimation flow. With the `report_power` command, we can obtain a detailed breakdown of dynamic and static power dissipation at hierarchical and instance levels. Additionally, we simply grouped the instances of the NOEL-V CPU based on manual identifications. This will highlight the contribution of key hardware components, such as the branch predictor, caches, register file, and the floating-point units. These insights enabled the identification of power hotspots and provided the basis for microarchitectural analysis for future optimizations.

5

Results

In this chapter, we evaluate and compare the performance metrics collected from two platforms. We also identify the power hotspots of the NOEL-V processor and analyze the cause of power consumption for major hardware modules.

5.1 NOEL-V platform

This section presents the performance results of the RISC-V processor on an FPGA. Also, results at the synthesis and power analysis stages through the ASIC flow are discussed.

5.1.1 Performance

Here, the results of the performance data from the NOEL-V platform are presented. We first look at the periodic execution of the Climate and Thermal management systems where each task is periodically scheduled using RTEMS.

Table 5.1 illustrates the performance of 60 seconds of execution with RTEMS. Count refers to the total number of invocations of the tasks, while missed indicates how many of those invocations missed their deadline. Wall time measures the time from when a task has been scheduled for execution until it completes, and CPU time is the portion of that time when the task is actively executing. For each of these metrics the minimum, maximum, and average value are recorded. The last column shows the maximum measured CPU utilization in a single invocation of the task, calculated by dividing the CPU time of the task by its period.

It is immediately evident that no deadlines have been missed which is the main requirement of this application. Further, The total utilization of all application tasks in the system does not exceed 8%, which is well below the 70% threshold for rate monotonic schedulability as determined by 2.1. These results strongly suggest that the climate application is schedulable on NOEL-V.

Next, the performance of the individual tasks in the climate application is evaluated. This is done by executing 3 tasks within the climate application, here referred to task A, B, and C for simplicity and recording the performance events that occur during execution. Table 5.2 shows what events occur during 1000 iterations of task A. The last row of the table shows CPI calculated by dividing the number of cycles by

Table 5.1: The performance of 60 seconds of execution with RTEMS.

Task	Count	Missed	Cpu min	Cpu max	Cpu avg	Wall min	Wall max	Wall avg	Utilization
MASK	6059	0	0.000040	0.000095	0.000067	0.000092	0.001855	0.000366	0.00959
Snsr	6066	0	0.000209	0.000220	0.000215	0.000289	0.002068	0.000569	0.02174
Clim	6074	0	0.000077	0.000082	0.000078	0.000364	0.002143	0.000645	0.00790
Clrr	608	0	0.000332	0.000343	0.000340	0.000352	0.000363	0.000360	0.00345
PMPD	609	0	0.000084	0.000086	0.000085	0.000417	0.000434	0.000423	0.00086
RlyD	610	0	0.000083	0.000086	0.000085	0.000497	0.000508	0.000504	0.00086
Tsnr	610	0	0.000325	0.000329	0.000327	0.000813	0.000825	0.000821	0.00332
CVDV	611	0	0.000280	0.000287	0.000281	0.001089	0.001102	0.001096	0.00286
FAND	612	0	0.000138	0.000139	0.000138	0.001219	0.001234	0.001227	0.00141
FANR	612	0	0.000080	0.000081	0.000080	0.001296	0.001308	0.001304	0.00082
Ficc	613	0	0.000055	0.000057	0.000056	0.001337	0.001354	0.001347	0.00057
Safe	613	0	0.000085	0.000086	0.000085	0.001414	0.001431	0.001423	0.00087
Shtr	614	0	0.000251	0.000255	0.000253	0.001661	0.001679	0.001670	0.00259
tmtP	615	0	0.000142	0.000144	0.000143	0.001802	0.001821	0.001809	0.00147
core	615	0	0.000222	0.000246	0.000227	0.000245	0.000271	0.000250	0.00233
Elec	616	0	0.000150	0.000152	0.000151	0.000376	0.000398	0.000382	0.00155
tmsk	617	0	0.000049	0.000052	0.000051	0.000421	0.000442	0.000424	0.00052
Fdig	617	0	0.000100	0.000102	0.000101	0.000516	0.000533	0.000518	0.00104
Batt	618	0	0.000158	0.000161	0.000159	0.000666	0.000684	0.000668	0.00164
Hvac	619	0	0.000173	0.000179	0.000175	0.000838	0.000853	0.000841	0.00181
Cond	619	0	0.000226	0.000236	0.000230	0.000245	0.000261	0.000252	0.00237
MNTR	1	0	0.577619	0.577619	0.577619	0.701982	0.701982	0.701982	0.00963
Total									0.07919

the number of instructions. This workload is small enough that the entire workload fits in the instruction and data caches of NOEL-V, resulting in a very high hit rate and few accesses to main memory, improving performance. Further, for this simple periodic task, the branch predictor achieves a near 100% hit rate as the workload consists mostly of taken branches that are common in control systems software. Despite few performance degrading events this workload achieves a CPI of 3.6, which is not explained by the performance model. When the workload is re-executed with all floating point operations replaced with integer equivalents, the picture becomes clearer. The CPI now drops to 2.24 which is a remarkable improvement. This behavior is the result of combining a slow, non-pipelined FPU with a floating-point heavy automotive workload. Additionally, NOEL-V has dual-issue capabilities when floating-point instructions are not involved, which further improves performance. Interestingly, this change also causes the branch miss rate to increase from close to 0% to more than 50%, which presumably is related to the dual issue change.

Table 5.3 displays the events occuring during the execution of task B. While still relatively small and simple, this task does more work and thus causes more events than the previous one. It is evident that this workload does not fit in the instruction cache, as there are a significant number of misses. There are also more branches in this task and, as a result, more miss predictions. The data cache still appears to be sufficient however, with much fewer misses than 1 per iteration of the task. In this task, the FPU has a much more limited impact on performance. With integer instructions execution time does decrease somewhat and CPI decreases from 5.2 to 4.6, but the number of pipeline stalls almost doubled and the instruction cache misses also increased enough to make the speedup insignificant.

The results of task C presented in Table 5.4. Once again, floating-point has a

Table 5.2: Illustrates the performance characteristics observed during 1000 iterations of periodic task 1. This simple task calculates a bit mask that is used by the other tasks.

Event	**Count (FP)**	**Count (Integer)**
Cycles	192430	119444
Instructions	54025	53026
Instruction cache misses	20	18
Data cache accesses	18009	14001
Data cache misses	40	35
Branches	2000	2000
Branch mispredictions	7	1006
Single issued instructions	34040	27041
Dual issued instructions	10000	13000
Pipeline stalls	1419	441
Cpi	3.6	2.24

Table 5.3: Performance characteristics of the climate sensor task.

Event	**Count (FP)**	**Count (Integer)**
Cycles	4217872	3953552
Instructions	808528	848035
Instruction cache misses	14130	26140
Data cache accesses	379015	387018
Data cache misses	31	32
Branches	113000	116002
Branch mispredictions	11625	14169
Single issued instructions	672547	604052
Dual issued instructions	67998	121999
Pipeline stalls	579972	1124738
Cpi	5.2	4.6

Table 5.4: Performance characteristics of the third periodic task in the climate application.

Event	Count (FP)	Count (Integer)
Cycles	541598	377259
Instructions	192027	178027
Instruction cache misses	31	29
Data cache accesses	98016	87023
Data cache misses	25	32
Branches	35000	31000
Branch mispredictions	2049	2040
Single issued instructions	130042	108042
Dual issued instructions	31000	35000
Pipeline stalls	13367	5002
Cpi	2.8	2.2

dramatic impact on performance, resulting in a 31% speedup and a decrease in CPI from 2.8 to 2.1 when integer is used instead. Furthermore, while the number of dual-issued instructions increased in this task just as with the other tasks, this did not result in an increase in branch misses and the number of pipeline stalls actually decreased. This suggests that while the issue pattern of instructions affects the branch predictor, it is not the case that dual issue always increases the miss rate.

5.1.2 Synthesis results

The design was synthesized using the Cadence synthesis tool Genus with the GPDK45 technology library. A clock period of 10ns, which is a clock frequency of 100MHz, was specified. The design has met all timing constraints without violations.

The synthesized netlist for the CPU core consists of approximately 1.46 million standard cells, with a total area of 6.5 mm^2. This area includes both combinational logic and sequential elements. Table 5.5 shows the area contribution of major hardware modules. It should be noted that *Pipeline* entry refers to the integer execution pipeline, including the ALU, branch logic, CSR control, exception handling, and the registers connecting each stage. In other words, it covers the full execution path of integer instructions, not only the pipeline registers.

From Table 5.5, it can be seen that the L1 cache occupies over two-thirds of the total CPU area, which is expected because it is based on flip-flop implementation. Execution units such as the pipeline and arithmetic unit together account for around 11%, reflecting the design goal of a simplified pipeline. The remaining area is distributed across control units like the memory management unit (MMU) and the branch predictor. Overall, this area distribution is aligned well with embedded designs.

Table 5.5: Area breakdown of major hardware modules in the synthesized CPU core.

Instance	Total area/mm^2	Percentage
L1 cache	4.466	68.75%
MMU	0.324	4.99%
Pipeline	0.532	8.19%
Mul/Div Unit	0.123	1.89%
Trace Buffer	0.474	7.30%
Integer Register File	0.277	4.26%
FP Unit	0.074	1.14%
Branch Predictor	0.190	2.93%
FP Register File	0.035	0.54%

5.1.3 Power consumption analysis

To analyze the power dissipation characteristics of the RISC-V processor when executing a specific automotive workload (climate system software), this section presents detailed results of leakage, dynamic, and total power for key hardware modules at the RTL level. Table 5.6 lists the measured power data and the corresponding percentage of the total power dissipation.

Table 5.6: Power consumption and percentage contribution of each module.

Instance	Leakage power/mW	Dynamic power/mW	Total power/mW	Percentage
L1 cache	0.108	42.829	42.937	39.28%
MMU	5.708×10^{-3}	22.038	22.043	20.17%
Pipeline	8.420×10^{-3}	19.664	19.672	18.00%
Mul/Div Unit	1.491×10^{-3}	10.035	10.037	9.18%
Trace Buffer	11.107×10^{-3}	4.119	4.130	3.78%
Integer Register File	6.230×10^{-3}	3.922	3.928	3.59%
FP Unit	1.014×10^{-3}	3.820	3.821	3.50%
Branch Predictor	4.361×10^{-3}	2.363	2.367	2.17%
FP Register File	0.779×10^{-3}	0.371	0.372	0.34%

The L1 cache contributes the most to the system, approximately 43 mW, which accounts for 39.28% of the total CPU power. This indicates a high frequency of data and instruction access during workload execution. However, the dynamic power of this component is much higher than the leakage power, which might be due to the implementation of the cache built with flip-flops rather than static random-access memory (SRAM) cells. Following that, MMU and the pipeline consume 20% and 18%, respectively. The above three modules are the primary power hotspots because of an overall 78% contribution.

In addition, the Mul/Div unit shows a significant power usage, up to 10 mW, which means an amount of multiplication and division operations in the control logic, although this unit only serves specific arithmetic. On the other hand, the FP Unit and FP Register File have relatively lower power. The total percentage of integer-related modules, including pipeline, Mul/Div Unit, and Integer Register File, is 8 times higher than that of FP-related modules. This ratio suggests that integer

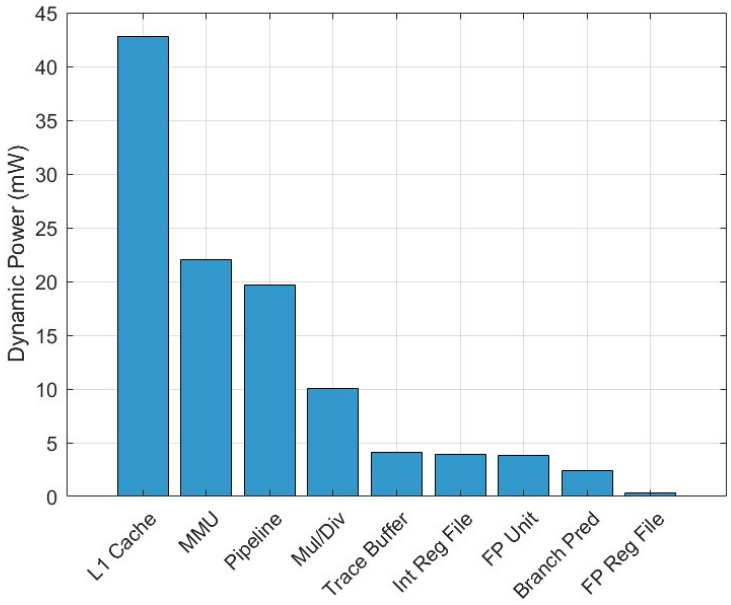

Figure 5.1: Dynamic power consumption per hardware module.

operations are frequent in the workload. Among the control-related modules, the branch predictor accounts for 2.17% and the trace buffer for 3.78%, but they play essential roles in pipeline performance and debugging.

We further break down the power composition of major modules and analyze the causes of power consumption based on their functionality and hardware implementation. To enhance the understanding of the power consumption, Fig. 5.1 and 5.2 illustrate the dynamic and leakage power dissipation of each module, respectively. Together with Table 5.6, these bar charts jointly provide both numerical precision and visual clarity. While the table represents exact values, the diagrams reveal patterns such as overwhelming dominance and relative ranking of each module's contribution to the total power.

Under the climate system software workload, the processor frequently accesses both instructions and data, making the L1 Cache active during most clock cycles. Therefore, its dynamic power reaches 42.8 mW, accounting for 39.28% of the total processor power. At the RTL level, the L1 Cache is implemented as a register array, which is synthesized into a memory structure composed of flip-flops (FFs), rather than using SRAM macros, since the academic PDK does not provide SRAM cells. As a result, the cache takes more area than it would with SRAM in an industrial design. During each read or write operation, entire rows or columns may be toggled, leading to widespread switching activity. Moreover, the lack of clock gating further amplifies dynamic power consumption. Additionally, the static power of the L1 cache is 107.7 μW, accounting for only 0.25% of its total power. This relatively low ratio is due to the dominance of dynamic switching activity during workload execution, as the cache is implemented using standard FFs. However, as shown in Fig. 5.2, L1 cache has the highest leakage power among all modules because of its large physical area and high density of registers, and continuous power-on status, which together

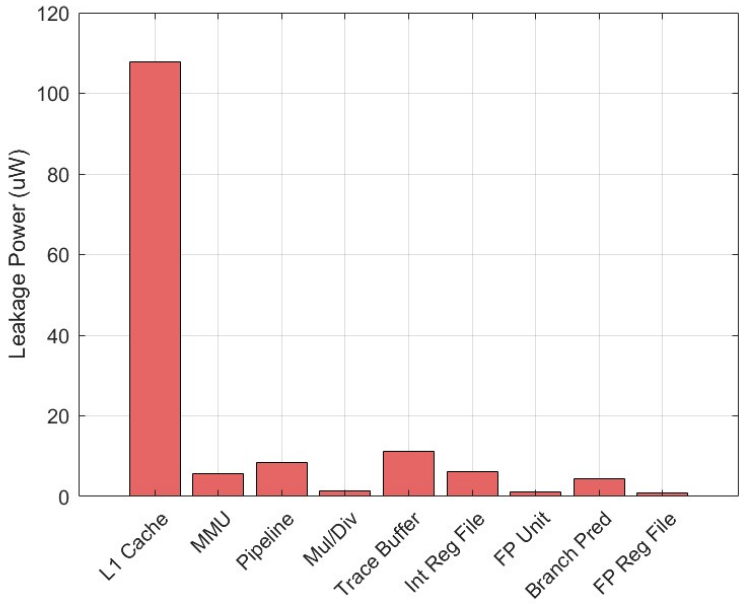

Figure 5.2: Leakage power consumption per hardware module.

result in a substantial amount of leakage current during workload execution. As a result, L1 cache becomes the dominant contributor to the total leakage power.

The MMU is responsible for handling refill, write-back, and external memory interface management. The RTL implementation shows that these operations are accomplished through several state machines. When a cache miss happens, the cache controller transitions through multiple states, such as from detection to issuing read requests to writing data back into the cache. These transitions drive frequent switching in control signals, resulting in significant dynamic power. Since clock gating is not enabled, many control and data signals remain active even when not in use, further increasing power consumption. Another notable contributor is the extensive use of multiplexers, used for cache way selection or write buffer selection, which are essentially combinational logic blocks composed of numerous logic gates. Even if outputs remain the same, changes in inputs can still cause internal switching activities, leading to unnecessary power dissipation. Overall, the combination of frequent finite state machine (FSM) transitions, lack of clock gating, and high combinational activity results in a dynamic power consumption of 22 mW, or 20.17% of the total.

The Pipeline module serves as the processor's execution core, handling all integer instructions, including arithmetic and logic unit (ALU) operations, branching, control and status register (CSR) access, and exception handling. It implements a classic multi-stage pipeline with decode, execute, memory access, and write-back stages, connected by a series of pipeline registers. These registers receive clock edges and are updated every cycle, regardless of whether the instruction is valid. Similarly, components such as the ALU, shifter, and branch logic execute operations on every clock cycle, even for invalid instructions, leading to high switching activity. For instance, when each instruction enters the execution stage, the ALU performs computations every cycle, even if the results are ultimately unused. Branch logic

also evaluates conditions and target addresses on every branch instruction, resulting in frequent transitions. The module also includes exception detection and CSR control logic that checks if illegal instructions and trap conditions exist on every cycle. Altogether, numerous consistently active registers and control paths, combined with the lack of clock gating, result in dynamic power reaching 19.7 mW, or 18% of the total.

The Mul/Div Unit handles integer multiplication and division. The relatively high power consumption of this module suggests that the target workload involves frequent arithmetic operations. The multiplication unit adopts a calculation bit-by-bit structure, suggesting computing partial products bit by bit and accumulating the result. The division unit utilizes an iterative structure that requires dozens of clock cycles to complete each operation, involving shifts, subtractions, and sign checks. Since each operation takes multiple cycles and activates wide combinational paths and internal registers, this module consumes significant power even at moderate usage frequencies.

The FPU controls floating-point operations such as addition, multiplication, and comparisons. It runs as a sequential unit rather than a pipeline, using a state machine to control each step. Each instruction takes approximately 2 to 4 cycles to complete. Although the module is not frequently used, each time it activates several functional blocks for tasks like exponent alignment and rounding, which causes bursts of switching activity. Combined with the FP Register File, which is accessed during each floating-point instruction, the overall contribution of FPU stays around 3.5% of the total.

The Branch Predictor consists of the branch history table (BHT), branch target buffer (BTB), and return address stack (RAS). These components are used in the instruction fetch stage to predict branch behavior and target addresses, helping to reduce pipeline stalls. In terms of hardware structure, they are implemented as small lookup tables and simple stack logic. Due to their compact size and limited access frequency, signal activity is low. Although they play a key role in performance, their power consumption is minimal. In this workload, the total branch prediction logic consumes less than 2.4 mW, or about 2.2% of total processor power.

The trace buffer is the part of the processor that keeps track of what instructions are being executed. It logs information like the current instruction address and whether a branch was taken, which is really helpful during debugging or simulation. The design itself is pretty simple, basically a circular buffer made of registers. Every time an instruction gets committed, some information gets written into it. Since it works in synchronization with the processor's instruction flow, almost every instruction can trigger a write, indicating lots of signal switching. Because the amount of data is small and the structure is straightforward, the overall power consumption stays quite low. Based on measurements, it uses about 4.1 mW, just occupying 3.78% of the total power.

In summary, the main power hotspots lie in the memory system and execution pipeline, indicating that this workload execution is dominated by intensive memory access and complex control logic. Modules such as the branch predictor and trace

logic, while consuming relatively little power, serve important functional roles in performance. These findings in this section provide a solid basis for exploring targeted module-level power reduction strategies in the next chapter.

5.2 ECU platform

This section presents the performance results of the TriCore and comparison to RISC-V processor, as well as the power consumption of the ECU.

5.2.1 TriCore performance and comparison to RISC-V

In Table 5.7 the performance of the TC399XP at 300MHz is compared to that of NOEL-V. The FPGA implementation of the RISC-V achieves consistently better CPI than the TriCore but its much lower frequency 40MHz makes it lose out on execution time. It is still impressive that NOEL-V manages to perform this well against the TC399XP in the type of task the TriCore was designed for.

Table 5.7: Performance of the different tasks on TriCore and NOEL-V.

TriCore	Task 0	Task 1	Task 2
Instructions	73	991	367
CPI	3.85	6.5	5.27
Cycles	281	6418	1936
Time (ns)	**0.94**	**21**	**6.4**
NOEL-V			
Instructions	54	808	192
CPI	3.6	5.2	2.8
Cycles	192	4217	541
Time (ns)	**4.6**	**105**	**13.5**

5.2.2 Power measurements

In this setup, the ECU is powered by a supply $V = 13.0V$. When the ECU begins executing the climate system software, the current increases from the idle state $I_{idle} = 0.12A$ to the active state $I_{active} = 0.18A$. Based on the power formula, we calculate that the increased power represents the ECU power consumption during the target workload execution, which is $P_{ECU} = P \times (I_{active} - I_{idle}) = P \times (0.18A - 0.12A) = 13.0V \times 0.06A = 0.78W = 780mW$.

Noticeably, the power increment reflects the entire ECU, which includes not only the processor core but also other peripheral components like communication interfaces and sensors.

Due to the lack of accessible chip power rails, we are unable to extract the CPU-only power usage. Thus, it is unavailable to make an equivalent comparison with the NOEL-V's CPU-only power dissipation.

6

Discussion

This chapter analyzes the performance and power results obtained from both platforms. It discusses the observed execution behaviors, highlights architectural factors that influence efficiency, and identifies limitations in the measurement methods. Finally, it explores potential directions for optimizing NOEL-V's power consumption, based on insights gained from the evaluation.

6.1 Performance

From the results obtained, it is evident that automotive workloads can indeed run on RISC-V. This is not surprising as the application was written in C.

Schedulability of the application on NOEL-V using rate monotonic scheduling is also strongly implied. This result alone is not conclusive proof as NOEL-V does not exhibit deterministic performance. While hardware features such as dual issue, branch predictors, and caches dramatically improve average performance, the hard real-time schedulability analysis must account for the case when none of these features can be relied upon, making it overly pessimistic.

But while this does not constitute a formal proof of schedulability, the actual CPU utilization remains far below the schedulability threshold at all times during the test, which at least strongly implies schedulability.

Compared to the TriCore-based ECU, NOEL-V achieves consistently lower instruction count and lower CPI, which at equal clock speeds should translate into better performance. The reduction in instruction count may be a result of the RISC-V ISA, but it is also likely affected by the compiler, where a modern version of GCC was used to target NOEL-V, while the ECU system uses the proprietary tasking compiler. As the tasking compiler is proprietary, it is unfortunately difficult to assess the impact of compiler choice. In either case, there is no official version of GCC or LLVM for TriCore, so even if the ECU could in theory perform better with another compiler, that will not happen in practice, which is a shortcoming RISC-V does not have. NOEL-V also achieves consistently lower CPI in the type of workload that the TC399XP was specifically designed for. This likely results in part from NOEL-V being a modern, high-performance (for the intended use case) design, which uses a 7-stage, dual-issue pipeline, branch prediction capabilities, and a sophisticated cache and system that can sustain one write per cycle. As the TC399

is a proprietary design, not much is known about its implementation details, but the design is over 10 years old at this point and has likely not kept up with the most recent microprocessor innovations. The TriCore does have some documented tricks in its favor however. It supports digital signal processing (DSP) instructions which allow for higher arithmetic performance and single instruction, multiple data (SIMD), and each core has a small amount of scratchpad memory which could be used to minimize memory access latency. It is unclear if the Climate application utilizes these features however. The CPI may be artificially limited by the low clock speed of the FPGA however. With a higher clock speed, the relative performance difference between CPU and RAM would increase, making cache misses cost more cycles thus increasing CPI.

It is evident that floating point performance is hugely consequential in this application, with CPI dropping between 0.6 to 1.36 cycles in the most extreme case when only integer instructions are used, which amounts to a 40% reduction. This is the maximum theoretical improvement that could be achieved with a better FPU. While an improvement of this magnitude is not expected from switching to a higher-performing FPU, it is evident that there are still significant performance gains possible. Better performing, pipelined FPU's are available, for example, within the commercial release of GRLIB, and as the FPU was only using a small fraction of the chip power, this is clearly a worthwhile investment. A larger issue is that OS overhead is not accounted for. This is a reasonable assumption when the tasks do more work and overhead becomes a negligible fraction, but in this case, that assumption does not hold.

6.2 Limitations of power evaluation

When it comes to power evaluation, both platforms have some limitations. For the RISC-V system, one of the issues is that we used a non-commercial technology library, which is not an accurate reflection of real silicon. Also, the behavior at RTL and gate-level can be different, which affects accuracy.

On the ECU side, we can only measure system-level power and do not know how much power the TC399 core alone consumes. Also, the resolution of the measurements is quite limited because the current is directly measured from the power supply.

Finally, the two platforms differ in both CPU frequency and process node, which makes a direct comparison impossible.

6.3 Power optimization directions of NOEL-V

There are two directions for improvement: reducing NOEL-V's power consumption through architectural optimizations, and increasing the accuracy of the power measurement methodology.

Regarding architectural optimizations, one of the most significant contributors to NOEL-V's total power consumption is L1 cache, accounting for 39%. This is mainly

due to its implementation using FFs rather than SRAM macros, which results in high switching activities across a large number of registers. Since SRAM macros are commonly used for energy-efficient memory storage with a dense physical area, replacing FFs with synthesized SRAM macros will reduce both physical area and dynamic power under frequent cache access. Additionally, enabling clock gating within the cache system can further decrease unnecessary switching when certain parts are idle. This helps further reduce dynamic power without affecting performance.

In terms of measurement accuracy, several enhancements to the power analysis flow can be applied. Performing power analysis after place and route would better capture the real circuit characteristics. Furthermore, using a commercial process node instead of a predictive PDK provides more realistic power and timing characteristics, making the results closer to the actual silicon. As an alternative reference, estimating the power consumption in a developed RISC-V implementation like GR765 from Gaisler also provides a reasonable method to get accurate power data.

In summary, L1 cache optimization is a key direction for reducing NOEL-V's power consumption. By replacing the memory structure with SRAM macros and introducing clock control alongside improved power measurement methodology, significant power savings would be achieved without compromising performance.

7

Conclusion and future work

This thesis has explored whether an open-source RISC-V processor, NOEL-V, can be used for realistic automotive applications with promising performance and power consumption. The study compared NOEL-V against a commercial ECU based on Infineon TriCore TC399XP. A climate control application from Volvo Cars was executed on both platforms, using specific methods and tools to evaluate performance and power consumption.

The results demonstrate that NOEL-V can successfully execute all tasks within the deadlines. The RISC-V processor achieved a consistently lower CPI than the ECU, indicating good instruction-level execution efficiency despite running at a lower clock speed. However, the FPU remains a bottleneck due to its low throughput, as shown by the significant CPI reduction when floating-point operations are removed. Furthermore, power estimation identified the L1 cache as a power hotspot, mainly due to its FF-based implementation. While the TriCore ECU offers faster overall execution, its power consumption could only be measured at the system level, making CPU-level comparison with NOEL-V infeasible.

To improve NOEL-V's performance and efficiency, two main hardware-level optimizations are proposed. First, replacing the current FPU with a pipelined implementation can accelerate floating-point operations. Second, redesigning the L1 cache using SRAM macros and clock gating can reduce switching activity and lower power dissipation. These improvements could significantly reduce power without sacrificing performance.

Future work could explore several directions. First, this study only tested single-core execution. Future work could explore how to use multiple cores at the same time and how to divide tasks across cores. Second, power estimation could be improved by using commercial tools and post-layout simulation. Third, software compatibility remains a challenge. Future research could investigate how to enable RISC-V platforms like NOEL-V to support automotive standards such as AUTOSAR, which are currently not natively supported but are essential for automotive systems.

In summary, this study demonstrates that open-source RISC-V processors like NOEL-V hold strong potential as energy-efficient and flexible alternatives for automotive embedded systems. With targeted architectural improvements, they could soon catch up with commercial ECUs in both performance and how practical they are for real-time control tasks.

Bibliography

[1] S. Chakraborty, M. Lukasiewycz, C. Buckl, *et al.*, "Embedded systems and software challenges in electric vehicles," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2012, pp. 424–429. DOI: 10.1109/DATE.2012.6176508.

[2] Cobham Gaisler. "NOEL-V high-performance RISC-V processor." Accessed: 2025-01-24. (2024), [Online]. Available: https://www.gaisler.com/products/noel-v.

[3] E. Lazzeri, B. E. Forlin, G. Furano, M. Ottavi, and L. Cassano, "An experimental comparison of RISC-V processors: Performance, power, area and security -special session paper-," in *2024 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2024.

[4] D. M. Brooks, P. Bose, S. E. Schuster, *et al.*, "Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors," *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000. DOI: 10.1109/40.888701.

[5] O. Chatzopoulos, M. Trakosa, G. Papadimitriou, W. S. Wong, and D. Gizopoulos, "Simpoint-based microarchitectural hotspot & energy-efficiency analysis of RISC-V OoO CPUs," in *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, 2024, pp. 120–131. DOI: 10.1109/ISPASS61541.2024.00021.

[6] L. Cuomo, C. Scordino, A. Ottaviano, *et al.*, "Towards a RISC-V open platform for next-generation automotive ECUs," in *2023 12th Mediterranean Conference on Embedded Computing (MECO)*, 2023.

[7] C. Scordino, I. M. Savino, L. Cuomo, *et al.*, "Real-time virtualization for industrial automation," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, IEEE, 2020, pp. 353–360. DOI: 10.1109/ETFA46521.2020.9211890.

[8] F. G. AB, *GRLIB IP library user's manual*, Version 2025.1, Frontgrade Gaisler AB, 2025.

[9] AUTOSAR. "Classic platform - AUTOSAR." Accessed: 2025-03-24. (2024), [Online]. Available: https://www.autosar.org/standards/classic-platform.

[10] Infineon Technologies AG. "Microcontroller products." Accessed: 2025-02-06. (2024), [Online]. Available: https://www.infineon.com/cms/en/product/microcontroller/.

[11] L. GmbH, *Laucherbach development tools product overview*, Available online: https://www.lauterbach.com, Lauterbach GmbH, 2015.

[12] N. M. E. Weste and D. M. Harris, "Chapter 5 - Power," in *CMOS VLSI Design, A Circuits and Systems Perspective*, M. Hirsch, M. Goldstein, C. Bell, and J. Holcomb, Eds., 4th, Boca Raton: CRC Press, 2016, ISBN: 9780321547743.

[13] D. Chinnery, L. Stok, D. Hathaway, and K. Keutzer, "Chapter 1 - Design flows," in *Electronic Design Automation for Integrated Circuits Handbook, Vol 2*, L. Lavagno, I. L. Markov, G. E. Martin, and L. K. Scheffer, Eds., 2nd, Boca Raton: CRC Press, 2016, ISBN: 9781315215112.

[14] S. P. Khatri, N. V. Shenoy, J.-C. Giomi, and A. Khouja, "Chapter 2 - Logic synthesis," in *Electronic Design Automation for Integrated Circuits Handbook, Vol 2*, L. Lavagno, I. L. Markov, G. E. Martin, and L. K. Scheffer, Eds., 2nd, Boca Raton: CRC Press, 2016, ISBN: 9781315215112.

[15] R. Chadha and J. Bhasker, *Power Analysis in ASICs*. Springer, New York, NY, 2012.

[16] N. H. E. Weste and D. M. Harris, "Chapter 7 - variability," in *CMOS VLSI Design, A Circuits and Systems Perspective*, M. Hirsch, M. Goldstein, C. Bell, and J. Holcomb, Eds., 4th, Boca Raton: CRC Press, 2016, ISBN: 9780321547743.

[17] J. Monteiro, R. Patel, and V. Tiwari, "Chapter 3 - Power analysis and optimization from circuit to register-transfer levels," in *Electronic Design Automation for Integrated Circuits Handbook, Vol 2*, L. Lavagno, I. L. Markov, G. E. Martin, and L. K. Scheffer, Eds., 2nd, Boca Raton: CRC Press, 2016, ISBN: 9781315215112.

[18] C. D. Systems, *Virtuoso NC-Verilog environment user guide*, Version IC6.1.8, Cadence Design Systems, Inc., 2021.

[19] C. D. Systems, *Genus user guide*, Version 18.1, Cadence Design Systems, Inc., 2018.

[20] Digilent Inc. "Arty A7 reference manual." Accessed: 2025-01-25. (2024), [Online]. Available: `https://digilent.com/reference/programmable-logic/arty-a7/`.

[21] Ubuntu Manpage. "Srec(5) — motorola s-record file format manual page." Accessed: 2025-03-24. (2014), [Online]. Available: `https://manpages.ubuntu.com/manpages/trusty/man5/srec.5.html`.