Benchmarking of Control Kernels on Open-Source RISC-V Processors

Narayan Ghimire

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 20.12.2022

Supervisor

Prof. Jussi Ryynänen, Aalto University

Advisor

MSc (Tech.) Subash Puri, Nokia

Aalto University
School of Electrical Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Narayan Ghimire

Title Benchmarking of Control Kernels on Open-Source RISC-V Processors

Degree programme School of Electrical Engineering

Major Micro- and Nanoelectronic Circuit Design **Code of major** ELEC3036

Supervisor Prof. Jussi Ryynänen, Aalto University

Advisor MSc (Tech.) Subash Puri, Nokia

Date 20.12.2022 **Number of pages** 56 **Language** English

Abstract

In recent years, the RISC-V Instruction Set Architecture has emerged as an open-source alternative in the processor market which is dominated by proprietary architectures. The modern telecommunication industry has adopted the RISC-V architecture for accelerating communication data paths. In a telecommunication system, control kernels play a crucial role in managing the underlying hardware to direct the flow of information between devices. A control kernel typically configures the underlying infrastructure for the system and provides services such as scheduling, resource management, and data processing. Such tasks are heavily dependent on configurations of the 5G systems. This thesis presents a study of the telecommunication related control kernel's performance and power efficiency on open-source RISC-V processors.

The open-source RISC-V implementation, CV32E40P, maintained by OpenHW Group is selected for benchmarking against Nokia's in-house processor core, NRISCV. The processors are synthesized at 1 GHz frequency for 7nm TSMC technology, and power is estimated on the synthesized cores using PowerArtist.

The study finds that control kernels' performance and power consumption are largely influenced by the underlying microarchitecture of the RISC-V processor, with some control kernels achieving significantly better performance and power efficiency for specific implementations. This study provides insight into the strengths and weaknesses of different RISC-V processors for control kernel applications and can guide the design and implementation of future telecommunication systems.

Keywords RISC-V, CV32E40P, PPA Analysis, Control Kernels, Benchmarks

Contents

Abbreviations

3GPP	3^{rd} Generation Partnership Project
4G LTE	Fourth-Generation Long-Term Evolution
5G-NR	Fifth-Generation New Radio
ABI	Application Binary Interface
ALU	Arithmetic Logical Unit
AMD	Advanced Micro Devices
ARM	Advanced RISC Machine
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction Processor
CMOS	Complementary Metal-Oxide Semiconductor
CQI	Channel Quality Indicator
CSR	Control and Status Register
DC	Design Compiler
DPM	Dynamic Power Management
FCPA	Fast Control Plane Algorithm
FSDB	Flatfile Streaming Database
HDL	Hardware Description Language
ILP	Instruction Level Parallelism
IP	Internet Protocol
IPC	Instructions Per Cycle
ISA	Instruction Set Architecture
ISS	Instruction Set Simulator
JAL	Jump and Link
JALR	Jump and Link Register
LTE	Long-Term Evolution
MCS	Modulation and Coding Scheme

MIPS	Millions of Instructions Per Second
MFLOPS	Millions of Floating-point Operations Per Second
MIMO	Multiple-Input Multiple-Output
OFDM	Orthogonal Frequency Division Multiple Access
O-RAN	Open Radio Access Network
PC	Program Counter
PMI	Pre-coding Matrix Indicator
PPA	Power Performance and Area
PRCS	PMI/RI/CQI Selection
RCD	Register Change Dump
RI	Rank Indicator
RISC-V	Reduced Instruction Set Computer – Five
RRM	Radio Resource Management
RTL	Register Transfer Level
SMCME	Shrinkage Method for Covariance Matrix Estimation
UE	User Equipment
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
VLIW	Very Long Instruction Word
WNS	Worst Negative Slack

Symbols

α	Switching Activity
f	Switching Frequency
C_{eff}	Effective Capacitance
V_{dd}	Supply Voltage
P_{dyn}	Dynamic Power Consumption
I	Number of Instructions in the Programs
CPI	Average Cycles per Instruction
T	Clock Cycle Time

1 Introduction

The rapid development in telecommunication technology has necessitated the use of more advanced hardware and software in the telecommunication system. Modern telecommunication devices require processing an enormous amount of data as the demand for real-time video transmission has increasingly grown over the low data rate voice services. The rate of growth of the data traffic often outperforms the enhancement in the hardware and software technology. In order to achieve an appropriate processing performance and power consumption, networking devices need architectural enhancement and efficient software to implement the numerous network attributes.

In telecommunication, software contributes a lot in terms of performance and energy consumption compared to its underlying hardware platform. It is a well-established fact among practitioners that optimized software on efficient hardware can dramatically increase performance as well as reduce power consumption. Control Software is integral to telecommunication networks to conduct system configurations, large data movements between memories, initiation, and the monitoring of running processes [1]. Such control software depends on the state of the system and more often needs to process data that is difficult to vectorize or exploit energy and latency-efficient features of the hardware. Fifth-Generation New Radio (5G-NR) adds complexity to the control system and makes the system take a larger processing time which otherwise could have been allocated to acceleration tasks. Therefore, with efficient control software, 5G can achieve a boost in performance and optimization in power consumption.

In recent years, power dissipation and processing performance have been crucial design constraints in the design of 5G-NR systems. In order to meet the requirements, telecommunication industries are evaluating various hardware architectures. RISC-V, which stands for 'Reduced Instruction Set Computer - five', is one of the most recently adopted Instruction Set Architectures (ISAs). The RISC-V has emerged as an open-source alternative to the mainstream proprietary ISAs. The RISC-V processor cores have massively attracted industries due to their simple and flexible hardware implementations.

1.1 Problem Statement

In practice, control kernels are run on a complex core such as ARM processors due to the general nature of the operation performed by the software mostly involving large data movement depending on the network configurations. Therefore, it is an interesting area to observe the behavior of the control kernels on RISC-V implementations.

The main purpose of the thesis is to assess the performance and power efficiency of industry-standard control software on open-source RISC-V platforms and under-

stand the efficacies and inefficiencies of the novel architecture from the standpoint of 5G control software. To accomplish the objective, the thesis will compare the popular RISC-V open-source processor against Nokia's in-house RISC-V processor and evaluate their usability in networking systems in terms of computing intensity and energy efficiency.

1.2 Thesis Structure

The remainder of this thesis is divided into six chapters. Chapter 2 introduces the RISC-V ISA and the existing extensions, followed by a brief description of a generic processor implementation. This chapter also discusses an overview of control software in 5G telecommunication. It further explains the power, performance, area, and their correlation in the software-hardware ecosystem. Chapter 3 discusses an overview of previous RISC-V benchmarking efforts. Chapter 4 proposes the benchmarks, selected processors, and the methodology exercised during this thesis. Chapter 5 presents the results and their analysis and finally Chapter 6 concludes this thesis.

2 Background

This chapter introduces the fundamental concepts and related topics to evaluate RISC-V processors across standard telecommunication benchmarks. The first section begins with the description of RISC-V ISA specification. A particular emphasis is given to the base integer instruction set as well as the available extensions on the selected cores. The second section briefly explains the generic implementation of the processor based on the RISC-V instruction set, emphasizing the pipelined architecture. The third section explores the role of control software in the 5G telecommunication system. The final section covers the background on Power, Performance, and Area (PPA) analysis and their correlation in the 5G ecosystem.

2.1 RISC-V ISA Overview

The RISC-V ISA was developed at the University of California, Berkeley, and is now maintained by the RISC-V foundation [2]. It was initially developed for computer architecture research and educational purpose. It has now evolved as a standard open-source architecture also for commercial implementation. Due to the open and simple nature of ISA, the RISC-V has received significant support from the open-source community, with adaptation and development of programming tools, such as a GNU compiler toolchain [3], C libraries (newlib, glib) [3], LLVM toolchain [4], GNU MCU Eclipse [5], and an official ISA simulator (Spike) [6]. The primary goal of a RISC-V is to be a completely open ISA that is suitable for native hardware implementation for a wide range of microarchitecture styles or implementation technology.

Typically, RISC-V ISA refers to a base ISA, which is essential in all implementations. The base ISA is optimized to a minimal set of instructions (i.e., unprivileged instruction set [7]) adequate to deliver a sensible target for compilers and modern operating systems. The base integer ISA contains instructions for integer arithmetic, integer loads, integer stores, and control flow. Additionally, a set of privileged instructions [8] is available that can only be executed when the processor is in a privileged mode, such as supervisor mode or machine mode. This separation of privilege ensures that applications cannot perform sensitive tasks or access protected resources without the explicit permission of the operating system.

Although RISC-V is a convenient term to represent ISA, it is rather a family of related ISAs. RISC-V currently has four different versions of base ISAs. Each base ISA is tailored for a specific purpose and can be distinguished by the number of integer registers, their width, and the corresponding size of address space, which is illustrated in Table 1.

Base Integer	Width of Integer Registers	Size of the User Address
RV32I	32	32-bit
RV64I	32	64-bit
RV32E	16	32-bit
RV128I	32	128-bit

Table 1: Variants of RISC-V ISAs. [7]

RV32I and RV64I are two prominent variants of base integer ISA which offer 32 integer registers, each having 32 bits and 64 bits address space respectively. RV32E is another variant that provides 32 bits address space with half the number of registers to reduce the hardware cost over RV32I. The RV32E better fits as an alternative to inexpensive, low-power embedded devices. Lastly, RISC-V has an RV128I variant that provides a larger flat 128 bits address space for future support. [7]

2.1.1 Structure of RV32I Base Architecture

Table 2 lists the RV32I registers and their names as defined by the RISC-V's Application Binary Interface (ABI). The architecture has 32 32-bit wide registers, namely, x0−x31. The larger the number of architectural registers, the larger the effect on computing performance, power consumption, and code size [7]. Register x0 is hardwired to constant 0, also called 'zero register'. A store to the zero register has no effect, and a read always provides 0. Besides this, the integer base RISC-V specification has 32 pseudo-instructions that rely on the zero register [9]. A pseudo-instruction is an instruction that is mapped by the assembler to another real instruction. For instance, branch-if-equal-to-zero is mapped to branch-if-equal. This means without a zero register, RISC-V would occupy 32 more opcodes for those instructions. Less number of opcodes simplifies the instruction decoder, which directly affects the performance and power consumption. Figure 2 illustrates one additional register other than the state registers known as the program counter (PC). The PC holds the byte address of the current instructions. [10]

31 0		31 0	
x0 / zero	Hardwired zero	x16 / a6	Function argument
x1 / ra	Return address	x17 / a7	Function argument
x2 / sp	Stack pointer	x18 / s2	Saved register
x3 / gp	Global pointer	x19 / s3	Saved register
x4 / tp	Thread pointer	x20 / s4	Saved register
x5 / t0	Temporary	x21 / s5	Saved register
x6 / t1	Temporary	x22 / s6	Saved register
x7 / t2	Temporary	x23 / s7	Saved register
x8 / s0 / fp	Saved register, frame pointer	x24 / s8	Saved register
x9 / s1	Saved register	x25 / s9	Saved register
x10 / a0	Function argument, return value	x26 / s10	Saved register
x11 / a1	Function argument, return value	x27 / s11	Saved register
x12 / a2	Function argument	x28 / t3	Temporary
x13 / a3	Function argument	x29 / t4	Temporary
x14 / a4	Function argument	x30 / t5	Temporary
x15 / a5	Function argument	x31 / t6	Temporary
16		32	

31 0
pc

Table 2: The architectural registers of RV32I. [9]

The base integer architecture does not have any register committed for the link register or stack pointer. Any register can be freely used for this purpose. However, there is a standard software calling convention, which generally defines the location. Each of the 32 registers has a different name as determined by the RISC-V's ABI based on its intended usage. The convention defines the register ra for holding the return address of the function call. Register sp is defined for holding the address of the boundary of the stack. Registers $t0 - t6$ are defined for holding temporary values that do not exist after function calls. Registers $s0 - s11$ are defined for holding values that persist after function calls. Registers $a0 - a1$ are defined for holding the first two arguments to the function or the return values. Registers $a2 - a7$ are defined for holding any remaining arguments. [11]

2.1.2 RV32I Instruction Format

The base RV32I ISA has six instruction formats: R-type (register to register operations); I-type (short immediate and loads); S-type (stores); B-type (conditional branches); U-type (long immediate); and J-type (unconditional jumps). Figure 1 demonstrates the detail of each kind of instruction format. The instruction format provides three register operands: $rs1$ and $rs2$ for dedicated source registers, and rd for a destination register. Besides these register operands, the instruction format also provides immediate fields which are always sign-extended. The sign bit always

present in the most significant bit of the instruction. [9]

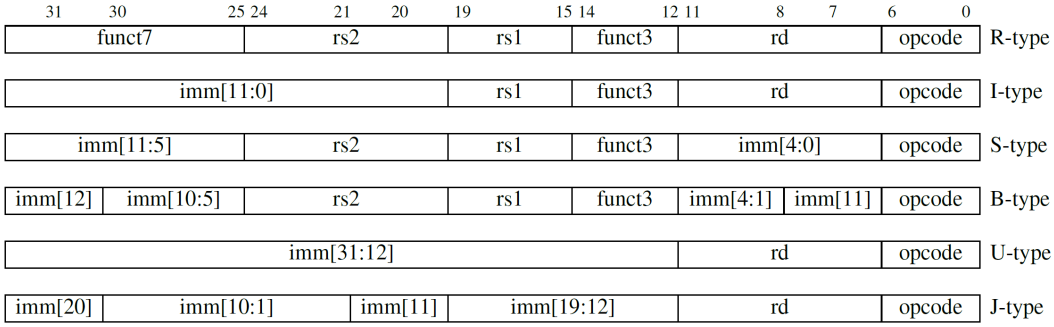

Figure 1: RISC-V base instruction formats. [9]

The simple nature of the RV32I instruction format plays a crucial role in performance and power efficiency. The RV32I has a fixed instruction length architecture, and all the instructions must be stored naturally aligned on a four-byte boundary in the memory in little-endian byte order. Furthermore, the register operands always occupy the same positions in the instructions. All these features of the instruction format overly simplify instruction decoding and also leverage to proceed with the register fetch in parallel, improving a critical path. [9]

2.1.3 Base Instructions

Most Integer arithmetic instructions are encoded in either R-type or I-type instruction format. Instructions such as ADD, OR, AND, and so on fall under the R-type format. These instructions operate on two source registers $rs1$ and $rs2$. Whereas instructions such as ADDI, ORI, ANDI, and so on fall under the I-type format. These types of instructions have 12 bits of immediate constants that operate with source register $rs1$. Both R-type and I-type instructions write the results to destination register rd. [7]

In addition to R-type and I-type formats, there is a U-type format that describes the instructions such as Load Upper Immediate (LUI) and Add Upper Immediate to Program Counter (AUIPC). Even though the LUI and AUIPC belong to the integer computation subgroup, they have different datapaths from the rest of the integer computation instructions. Figure 2 represents one of the possible datapaths for the LUI instruction. LUI places the immediate value of U-type format in the top 20 bits and the remaining lowest 12 bits are filled with zero. The "Fill 0" module at the top of the Register File present in the diagram is used to fill the bottom 12 bits with zeros. The result is stored in the destination register rd. AUIPC works similarly except it is used to build PC-relative addresses. For this, the AUIPC instruction forms an intermediate value by concatenating the 20-bit immediate value and 12-bit zeros. This intermediate value is then added to the current AUIPC instruction being

executed and then stored in the register rd. [12]

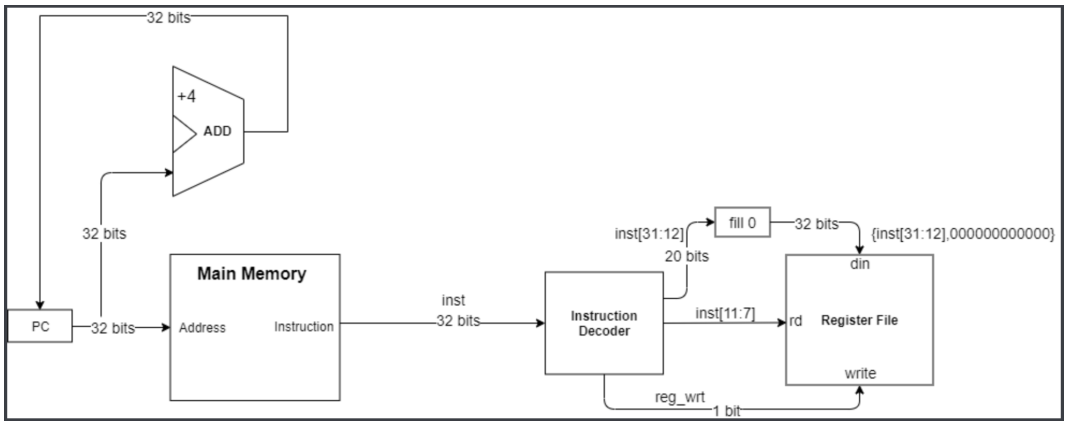

Figure 2: LUI Datapath [12]

Besides the integer computational instructions, RV32I features load and store instructions as prominent base instructions. The architecture itself is commonly known as load-store architecture, since only load and store instructions access memory, and arithmetic instructions only operate on CPU registers. Load uses the I-type format and copies a value from memory to register rd, whereas store uses the S-type format and copies the value in register $rs2$ to memory. Load and store instructions both form the address by adding the contents of the register $rs1$ to the sign-extended 12-bit offset. [7]

The RISC-V ISA provides two forms of control transfer instructions, namely, unconditional jumps and conditional branches. Unconditional jumps primarily contain Jump and Link (JAL) and Jump and Link Register (JALR) instructions. JAL instruction, whose datapath is depicted in Figure 3, uses a J-type format. In J-type format, immediate value bits are jumbled around. These bits are sorted primarily and the 0^{th} bit is set to 0. The value is then sign-extended to the size of an address making an integer value of approximately ± 1 million. Thus, the jumps can target function calls within a 1 MB range from the calling instruction. The sign-extended integer is finally added to the address of the jump instruction to create the jump target address as shown in the diagram. On the other hand, JALR instruction uses an I-type instruction format. In I-type instruction format, the sign-extended 12-bit immediate value is first added to the value stored in register $rs1$. Then the jump target address is acquired by setting the least-significant bit (bit 0^{th}) of the addition to 0. Both JAL and JALR write the address of the following instruction (current PC + 4) to register rd. [7]

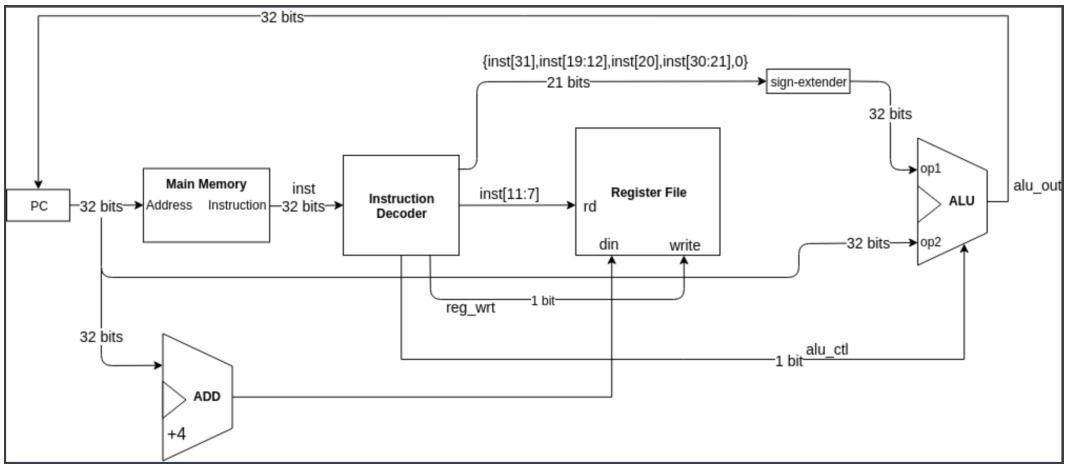

Figure 3: JAL Datapath [12]

In addition to the unconditional branch, the RISC-V also provides conditional branch instructions, whose datapath is presented in Figure 4. These instructions follow the B-type instruction format, where the target address is obtained similarly to the J-type instruction format. However, the B-type instruction contains only 12-bit immediate, making the jump target in the ± 4 KB range. Conditional branch instructions first compare the two source registers $rs1$ and $rs2$. Based on the comparison, these instructions evaluate the conditions, such as 'equal', 'not equal', 'less than', 'less than or equal', 'greater than', and 'greater than or equal', and lead to the decision of branch selection. As illustrated in the diagram, the outcome of the arithmetic-logical unit (ALU) is chopped to its least significant bit, which is used to regulate the multiplexer to select whether the branch is taken or not. If the comparison operation returns false, the PC value is incremented normally. [12]

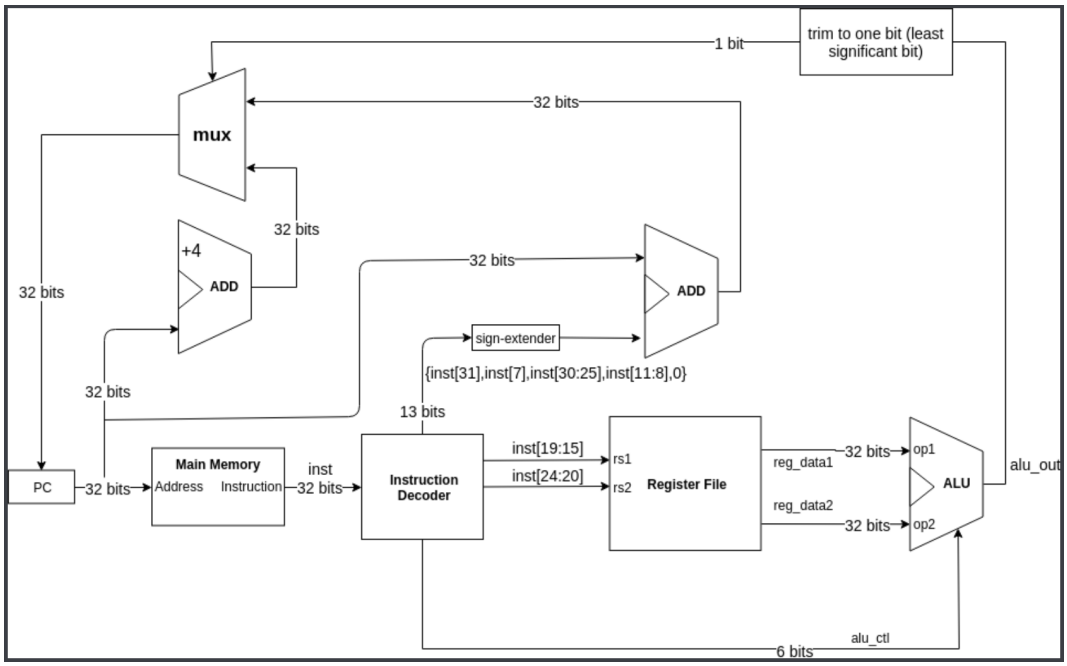

Figure 4: Conditional Branch Datapath [12]

The RISC-V defines the instruction set and register model for a processor, but it does not specify the exact microarchitecture or implementation of the processor. This allows designers to create processors with different microarchitectures and performance characteristics that are still compatible with the RISC-V ISA. Therefore, the diagrams presented in the Figure 2, 3, and 4 are one possible representation of LUI, JAL, and Conditional Branch Datapath respectively.

2.1.4 RISC-V Standard Extensions

RISC-V has several standard extensions to support extensive customization and specialization. Table 3 presents the list of standard extensions that are already approved by the RISC-V foundation. In addition to the RISC-V standard extensions, the ISA allows for custom third-party ISA. Custom extensions intend to allow designers to implement additional specialized instructions. These extensions are independent and can be developed in parallel without affecting the base ISAs. [7]

Table 3 illustrates several extensions that are marked as Ratified, Frozen, and Draft. Modules marked as Ratified have been already ratified by the time of this documentation. The modules marked as Frozen are not supposed to diverge substantially before being put up for ratification. The extension modules marked as Draft are premature, and deviations are expected in the future. [7]

Base		
Name	**Version**	**Status**
RVWMO	2.0	Ratified
RV32I	2.1	Ratified
RV64I	2.1	Ratified
RV32E	1.9	Draft
RV128I	1.7	Draft
Extensions		
M	2.0	Ratified
A	2.1	Ratified
F	2.2	Ratified
D	2.2	Ratified
Q	2.2	Ratified
C	2.0	Ratified
Counters	2.0	Draft
L	0.0	Draft
B	0.0	Draft
J	0.0	Draft
T	0.0	Draft
P	0.2	Draft
V	0.7	Draft
Zicsr	2.0	Ratified
Zifencei	2.0	Ratified
Zam	0.1	Draft
Ztso	0.1	Frozen

Table 3: RISC-V Extensions [7]

The base integer ISAs prefixed by 'I', such as RV32I or RV64I have instructions for integer handling, loading/storing, and control flow, such as addition/subtraction, bitwise logic operation, and shifting. The standard extension 'M' adds multiplication and division instructions for the integer values. 'A' adds instructions that atomically read, modify, and write memory allowing inter-processor synchronization. 'C' extension adds a compressed subset of instructions to allow for smaller code size. 'F', 'D', and 'Q' extensions add support for native floating-point operations for single, double, and quad precision respectively. [7]

The open-source processor selected for this thesis study has a few additional extensions on top of the base instructions set, such as Multiplication and Division Instructions, Compressed instructions, Performance Counters, Control and Status Register Instructions, and Instruction-Fetch Fence extensions. These instructions are given particular attention in the upcoming sections.

Multiplication and Division Instructions (M)

All multiplication operations MUL, MULH, MULHU, and MULHSU operate on two source registers. However, only MUL places the lower 32 bits in the destination register, whereas the other operations place the upper 32 bits according to the sign of the operands. For both high and low bits of the same product, code sequence MULH[S][U] rdh, $rs1$, $rs2$; MUL rdl, $rs1$, $rs2$; is used to fuse the two separate operations into a single multiply operation.

Division operation also has four instructions, namely, DIV, DIVU, REM, REMU, where instruction with the letter 'U' at the end operates on unsigned operands. Registers $rs1$ and $rs2$ store the dividend and divisor respectively. The division operation rounds the quotient towards zero and the quotient always possesses the sign of the dividend. For both quotient and remainder of the operation, code sequence DIV[U] rdq, $rs1$, $rs2$; REM rdr, $rs1$, $rs2$; is used to fuse the two separate operations into a single division operation. The RISC-V ISA does not trigger any exception for division by 0, this situation is detected by inserting the branch instruction right after the division. [7]

Compressed Extension (C)

The compressed extension is introduced in RISC-V ISA to reduce static and dynamic code size. The goal of the extension is to present short 16-bit instructions that can perform the most common operations and to make it compatible with all the other standard extensions. According to the RISC-V specification [10], typically, the compressed instructions can substitute 50%-60% of the RISC-V general instructions, reducing 25%-35% in code size. The compressed instruction is made compatible with the 32-bit instruction encodings by setting the lower two bits of 32-bit instructions to 11, with all other values reserved for 16-bit instruction words. [9]

Control and Status Register Instructions (Zicsr)

According to the RISC-V unprivileged specification [7], the ISA provides a distinct address space of 4096 Control and Status Registers (CSRs) in each hart and a full set of CSR instructions that operate on the CSRs. The CSR instructions, namely, CSRRW atomically reads and writes the CSR, CSRRS atomically reads and sets bits in CSR, and CSRRC atomically reads and clears bits in CSR. The CSRRWI, CSRRSI, and CSRRCI variants are like CSRRW, CSRRS, and CSRRC respectively, except they update the CSR using a 5-bit unsigned immediate value from register $rs1$, after zero-extending to 32-bit.

Performance Counter (Zicount)

The RISC-V ISA offers an arrangement of 32 performance counters and timers, each of 64-bit length. These counters and timers are accessible via a 12-bit CSR address

space and accessed by using CSRRS instructions. The first three counters are reserved for cycle count, real-time clock count, and instructions-retired count, while others can be used as programmable event counters depending on the implementation. The counter instructions such as RDCYCLE, RDTIME, and RDINSTRET reads the lower 32-bit of the cycle CSR, time CSR, and instret CSR respectively, whereas the instructions RDCYCLEH, RDTIMEH, and RDINSTRETH reads the bits 63-32 of the respective counters. The underlying 64-bit counter should never overflow in practice. [7]

Instruction-Fetch Fence (Zifencei)

Zifencei extension is presented in the RISC-V ISAs to support explicit synchronization amongst writes to an instruction memory and instructions fetches on the same hart. This operation is carried out by the FENCE.I instruction. Only the FENCE.I instruction guarantees the stores to instruction memory on a RISC-V hart will be made visible to instruction fetch on the same RISC-V hart. [7]

2.2 Processor Implementation

This Section covers a simple version of the generic RISC-V processor implementation. The implementation includes a subset of the RISC-V instructions, such as integer arithmetic-logical instructions, memory-reference instructions, and branch instructions described in Section 2.1.

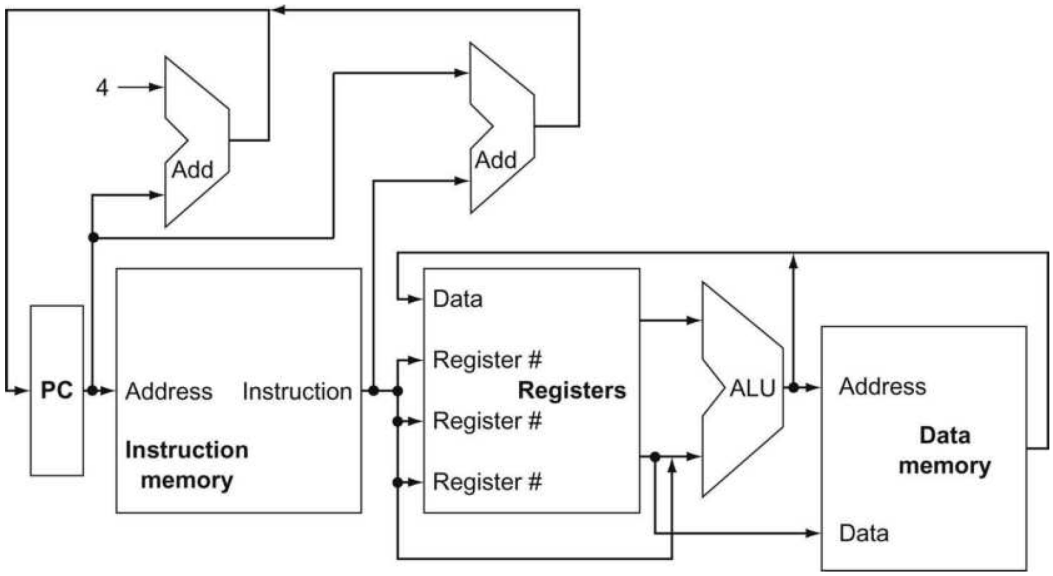

Figure 5: An overview of the implementation of the RISC-V subset presenting various functional units and connections among them. [13]

Figure 5 presents a simple RISC-V implementation with several functional units and their interconnects. In RISC-V implementation, the first two steps for all the instructions are similar. Every instruction begins by sending the address of the program counter to the instruction memory. The instruction that corresponds to the address of the program counter is then fetched. This instruction specifies the register operands for reading and writing register files. Usually, the instruction requires reading two registers but the load instruction requires reading a single register. After these two identical steps, the instruction classes, such as the integer arithmetic-logical instructions, the memory-reference instructions, and the branch instructions all perform the specific action to complete the instruction. Even these specific actions have some similarities in all three instruction classes. For instance, after reading the register, all the instruction classes use the ALU. The arithmetic-logical instructions use the ALU for the operation execution, the memory-reference instructions for an address calculation, and conditional branches for the equality test. These similarities and regularity in the execution of many of the instruction classes have largely simplified the implementation of the RISC-V processor. [13]

After the ALU operation, different instruction classes perform different actions to complete the instruction. An arithmetic-logical instruction writes the data from the ALU into a register. A memory-reference instruction either reads data from memory or writes data to memory. A conditional branch instruction changes the following instruction address depending on the comparison outcome. Otherwise, the PC value is incremented by four to obtain the address of the following instruction. [13]

Figure 5 is a single-cycle implementation. Though the single-cycle design works accurately, it is uncommon in modern design due to its inefficiency. In this design, the clock cycle is decided by the longest feasible path in the implementation. This makes it redundant to use methods that minimize the delay of the common cases other than worst-case cycle time. [13]

The next section presents the implementation technique known as pipelining, which uses a similar datapath to Figure 5, but has much higher throughput.

An Overview of Pipelining

Pipelining refers to an implementation technique where multiple instructions are overlapped during execution. It does not decrease the time it takes to complete an individual instruction, known as latency. It rather improves the instruction throughput. Instruction throughput is a critical metric because real programs perform billions of instructions.

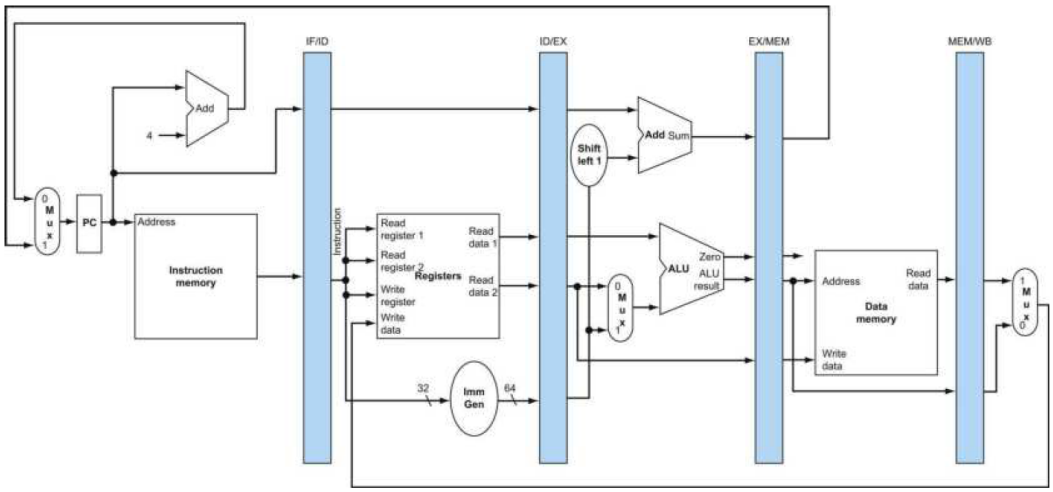

Figure 6: The pipelined version of the datapath. [13]

Considering the RISC-V implementation described in Section 2.2 is categorized into five steps, namely, Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Data Access (DA), and Write Back (WB) as depicted in Figure 6. Hence, the pipeline described in this section has five stages. All the pipeline stages take a single clock cycle. Therefore, the clock cycle is determined by the slowest operation.

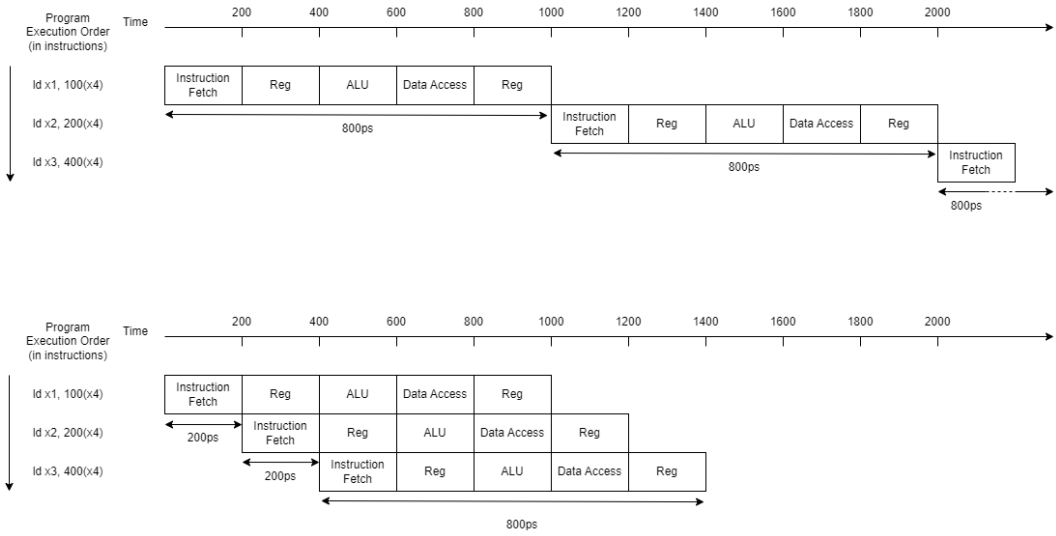

Figure 7: Nonpipelined execution (top) versus pipelined execution (bottom). [13]

Figure 7 illustrates a significant performance improvement in pipelined execution versus the single-cycle nonpipelined execution. Under an ideal situation and with a larger number of instructions, performance increase by the pipelining is equivalent to the number of pipeline stages. This implies that a five-stage pipeline nearly increases the performance five times, which can be formulated as in Equation 1.

$$Time\ between\ instructions_{pipelined} = \frac{Time\ between\ Instructions_{nonpipelined}}{Number\ of\ pipeline\ stages} \quad (1)$$

However, the pipeline stages are not ideal and may be imperfectly balanced. Moreover, pipelining involves additional registers to communicate between two consecutive stages. All this results in an increment in the minimum possible time it takes to execute single instructions in the pipelined implementation. This further justifies the performance increment that is less than the number of stages, which is predicted in Equation 1.

Despite offering plenty of advantages to modern architecture, longer pipelines have some downsides too. There are events in pipelining when the execution of the subsequent instructions is hindered in the succeeding clock cycles, known as hazards. There are three kinds of hazards, namely, structural hazards, data hazards, and control hazards.

A structural hazard is an event that is caused due to lack of hardware support for the combination of instructions that are set to perform in the same clock cycle. Data hazards happen due to the reliance of one instruction upon previous instruction that remains still in the pipeline. Data hazards can be mitigated with the help of compilers but stacking additional hardware to recover the missing data initially from the internal resources, called forwarding or bypassing, is the most efficient solution. The third type of hazard, control hazard, also known as branch hazard, occurs due to the necessity of making a decision based on the results of one instruction while others are executing. There are primarily two solutions to the control hazard: stall and predict. The stall is the conservative option of waiting for the result of one instruction before fetching the next instruction in the pipeline. Stall certainly works, but it is slow. While predict does not slow the pipeline when the prediction is correct, otherwise the whole pipeline needs to be flushed. The method of resolving the branch hazard by prediction is known as branch prediction. [13]

2.3 Control Software in 5G Telecommunication

The fifth-generation (5G) of cellular networks is a technological leap over a fourth-generation long-term evolution (4G LTE), introducing significant software innovations. This trend of software advancement has been a consistent process over each generation since the first commercially deployed cellular network, known as 1G. 1G was primarily focused on analog voice communication which was later adapted for very low-rate data communication. The software in this generation was specific to low-level embedded hardware. In comparison to 1G, the second-generation cellular networks (2G) offered native support for digital data due to which the 2G system became the first one to support encryption for both user and control traffic. The software in this generation was already responsible for critical control and management tasks, including monitoring and billing. Compared with 2G, the third-generation cellular

networks (3G) delivered improvements in the data rates, quality of voice communication, efficiency in using RF spectrum, and integration with internet protocol (IP) networks. As the infrastructure in this generation became more complex, the software became increasingly relevant for organizing the increased volume of data communication.

The fourth-generation cellular network (4G) is the first one designed to have data and software as relevant elements. The 4G system added Orthogonal Frequency-Division Multiple Access (OFDMA) for supporting multiple access to network resources and Multiple-Input Multiple-Output (MIMO) for a more stable wireless connection. Besides this, various releases under the 3GPP 4G LTE added some notable features, such as Evolved Packet System (EPS), Self-Organizing Network (SON), Multimedia Broadcast and Multicast Service (eMBMS), small cells and network densification, Device-to-Device (D2D) communication, Machine Type Communication (MTM), and addition of new spectrum bands for LTE operation [14]. From the software perspective, 4G displays considerable evolution over previous generations which is mainly driven to properly orchestrate these added features for efficient network control. For instance, each radio cell in a 4G LTE network is estimated to have up to 3000 different parameter settings that need to be configured, updated, and controlled to guarantee the correct functionality of the network [15]. Manual configuration of these parameters to operate the network is impossible. The control software can efficiently configure these parameters and enhance the performance of the system while optimizing power dissipation.

The arrival of 5G networks is foreseen to achieve infinite connectivity, extremely low latency, and faster data transfer. The system is being developed to achieve universal high-capacity radio for the evolution of technologies, such as artificial intelligence, automation, and the Internet of Things (IoT). Furthermore, the system extensively uses native cloud applications and cores to make the design more flexible and dynamic. In terms of the software, 5G is designed to make extensive use of some of the most powerful software paradigms, such as virtualization of services, Software-Defined Networking (SDN), and microservices. These software models are further organized and controlled by the control software. Additionally, the control software needs to efficiently manage the RF spectrum, performing a series of tasks, such as radio resource management, connection mobility control, dynamic allocation of resources to user Equipment (UE), and session management, among other functions in 5G networks.

The control software in 5G telecommunication is crucial in the proper orchestration of resource allocation and efficient network control. It also provides programmability, flexibility, and scalability to the system. 5G is a complex system made of a pool of numerous subsystems. Each subsystem is designed to perform specific features supported by a specific software-hardware component. These components need to be controlled appropriately for optimum efficiency with the help of control software. Besides this, the control software articulates these individual subsystems and organizes

the whole system for the intended purpose. One example of this behavior of control software can be taken from the base station where the control software manages the system-level control of radio resources, known as Radio Resource Management (RRM) [16]. RRM manages the radio resources, such as user scheduling, link adaptation, handover, and transmit-power. A user scheduler selects a time, frequency intervals, and spatial resources for each user to transmit data. The link adaptation includes the selection of the modulation and coding scheme (MCS) for each UE. Additionally, it also selects Pre-coding Matrix Indicator (PMI) and a pre-coder for UE. In a nutshell, Modern telecommunication systems are dependent on control software for their successful operation.

2.4 Power, Performance, and Area Analysis

Power, Performance, and Area (PPA) analysis is performed on an implementation of a processor. The PPA analysis is particularly significant because it allows designers to optimize and evaluate the processor implementation. It is desirable to do these optimizations before tapeout, as once the processor is manufactured its PPA characteristics are locked. The processor cannot be improved without incurring massive extra manufacturing costs.

A trade-off between PPA metrics is a well-established conundrum among circuit designers. Modern circuit designers are longing for flexibility in the design while paying substantially lower PPA costs. Historically circuit designers have been exploring the area and performance implementation space for optimizing processor design. Several studies have stated that a design with a larger area can complete a provided computation task in less amount of time. It has been mentioned that doubling a processing speed can expand die size by $2-4$ times [17].

However, area and performance are not the only parameters that affect the implementation of the design. Power is being given similar importance to area and performance considerations with the decreasing device size. Decreasing the device size enables increasing performance at an expense of a large electric field. This electric field can be mitigated by decreasing the supply voltage. However, shrinking the supply voltage by half shrinks the operating frequency by half [17]. As energy efficiency becomes a critical factor in the design, the trade-off between performance and power consumption forces the designers to use silicon area wisely to increase performance.

Despite of constant challenge between the PPA tradeoff, its optimization must be secondary to an actual realization of the design at reasonable design effort, unless the production cost of the design is extremely large.

2.4.1 Power

Considering the aim of this thesis study, it is reasonable to briefly review the mechanisms for power consumption in integrated circuits. Modern-day integrated circuit integrates numerous miniaturized digital systems due to the advancement of complementary metal-oxide semiconductor (CMOS) technology. CMOS circuits predominantly dissipate power in the form of dynamic power and static power. Dynamic power typically accounts for at least 90% of the total power dissipation in CMOS circuits [18], and it is the outcome of charging and discharging parasitic capacitances in the circuits [19]. The dynamic power consumption of CMOS circuits depends on switching activity (α), switching frequency (f), the effective capacitance (C_{eff}), and the supply voltage (V_{dd}) as presented in Equation 2.

$$P_{dyn} = C_{eff} \cdot V_{dd}^2 \cdot \alpha \cdot f_{clk} \tag{2}$$

As can be observed from Equation 2, the supply voltage has a quadratic influence on dynamic power. With the scaling of technology towards smaller nodes, supply voltage has dropped steadily, decreasing the dynamic power dissipation. Unfortunately, smaller designs intensify leakage, so static power starts to govern the power consumption equation in processor design. Leakage power or static power refers to the power that is consumed in the form of leakage currents and substrate injection currents when the system is in standby mode. Leakage power dissipation is proportional to area and temperature and hence to the density of transistors packed on a single chip. [18]

While insight into dynamic and leakage power provides a foundation to consider a low-power CMOS design, plenty of effort has been carried out in accomplishing lower power consumption at all abstraction levels of the design process. Table 4 lists the major abstraction levels of the design process and the techniques involved in the respective level for power optimization.

Abstraction Levels	Techniques	Opportunities
System level	Hardware/Software partitioning Power management	10 - 100 times
Architecture level	Parallelism Pipelining Voltage /frequency scaling Multi-supply voltage Power/clock gating Asynchronous design	10 - 90 %
Circuit/Logic/Gate levels	One-hot coding Ripple counter Bus inversion Word-length reduction Avoiding combinational loops Binary representations	15 -50 %
Transistor level	Substrate biasing Technology optimization Multi-oxide devices Layout optimization Oxide thickness reduction Capacitance minimization	2 - 10 %

Table 4: Level of design abstraction for power saving. [19]

Table 4 portrays the effectiveness of the power optimization techniques at each level of abstraction. It is illustrated in the table that the approaches used to optimize the power dissipation at the architecture and system level are more efficient than the approaches taken at the logic or gate level. The table depicts an opportunity for only 2−5% of power optimization at the transistor level. The opportunities dramatically increase as the hierarchy of the abstraction levels in the design process increases to the system level. The system level displays the colossal possibility of 10 to 100 folds of power optimization.

At the system level, hardware/software partitioning, i.e., organizing the implementation of the system components between hardware and software, is one of the crucial steps in power optimization when trade-off needs to be made between conflicting parameters, such as power, performance, and hardware size [20]. Additionally, system-level software at this level largely affects power consumption by coordinating several tasks as well as managing resources, and thus controlling the complete hardware platform. Furthermore, the choice of the algorithm for specified functionality and hardware architecture for rendering the functionality with the designated algorithm affects power consumption. In terms of power budget, the main purpose of this design level is to prepare the system with an energy-efficient run-time support system. Dynamic Power Management (DPM), which allows systems to adjust to time-varying workloads, can be considered an example of such a system. DPM is a design tech-

nique that dynamically reconfigures a system with the minimum number of active components or a minimum load on such components to deliver the requested services and performance levels [21].

All the approaches and techniques depicted in Table 4 ultimately reduce the power dissipation by lowering either the supply voltage, the voltage swing, the physical capacitance, the switching activity, or a combination of the above.

2.4.2 Performance

Performance refers to the speed of execution for a given task in a specific implementation. The size of the task is measured as the number of clock cycles it takes to execute a program. In other words, performance is typically associated with the number of computations in a specific amount of time, and it is often measured in Millions of Instructions Per Second (MIPS) or Millions of Floating-point Operations Per Second (MFLOPS). The classic performance equation of the processor is given as:

$$CPUTime = I \times CPI \times T \tag{3}$$

Where:
I $\rightarrow$ the number of instructions in the program,
CPI $\rightarrow$ average cycles per instructions, and
T $\rightarrow$ clock cycle time.

The overall performance of the system is affected by the programming language, the compiler, and the architecture. Architectural enhancement has been a steady process since the basic von Neumann architecture [22]. Architectural innovators have developed various enhancements, such as pipelined instruction processing (described in Section 2.2), superscalar, and Very Long Instruction Word (VLIW) architecture, to introduce a new form of parallelism into instruction processing. Superscalar architectures have at least two parallel pipelines for improving the instruction throughput. VLIW architectures are similar to superscalar in terms of extensive hardware redundancy for supporting the parallel processing of instructions. However, VLIW depends on the compiler to effectively schedule instructions to exploit the parallelism in the processor architecture.

The success of each architectural enhancement has tested the capability of compiler technology to deliver efficient language implementations on that architecture. The basic task of the compiler is to translate a programming language's source code into the machine code for the target machine. In addition to this, modern compilers have developed techniques, such as vectorization, instruction scheduling, and management of memory hierarchies, which makes the system compute-intensive [23]. The compiler's responsibility does not end with the production of a correct machine-language

translation of the source program, it must produce a suitable efficient program as well. A well-written source program is beneficial in achieving the required efficiency.

2.4.3 Area

In 1965, Gordon Moore detected that the number of transistors in a dense integrated circuit doubles every two years. This phenomenon is known as Moore's law. In recent times, as the evolution of semiconductor process technology is reaching molecular limits, the exponential benefits of Moore's law have realized some slowdown. Although silicon technology has substantial challenges to following Moore's law, leading some to predict its demise, the industry has managed to overcome the challenges with several innovative ideas other than decreasing the size of the transistor and is likely to do so in the foreseeable future. Some of these innovative solutions proposed by the International Technology Roadmap for Semiconductors (ITRS) [24] involves different transistor model like gate-all around transistor [25], vertical transistors [26], and 3D design choices [27].

For integrated circuits, the relationship between the silicon die area and product cost is straightforward. Increasing implementation area directly corresponds to higher packaging costs as well as reduced fabrication yield resulting in increased manufacturing costs. The cost factor has constantly pushed for the smaller silicon size for decades, especially for high-volume production.

However, it is observed that the designer's productivity has hindered the potential shrinkage of the die area. A study shows the transistors per die (die complexity) have increased by 58% per year, but designer productivity has merely grown by 21% per year [17] which is illustrated in Figure 8. Meanwhile, this productivity gap has provided opportunities for many design ideas in all the abstraction layers of the design hierarchy.

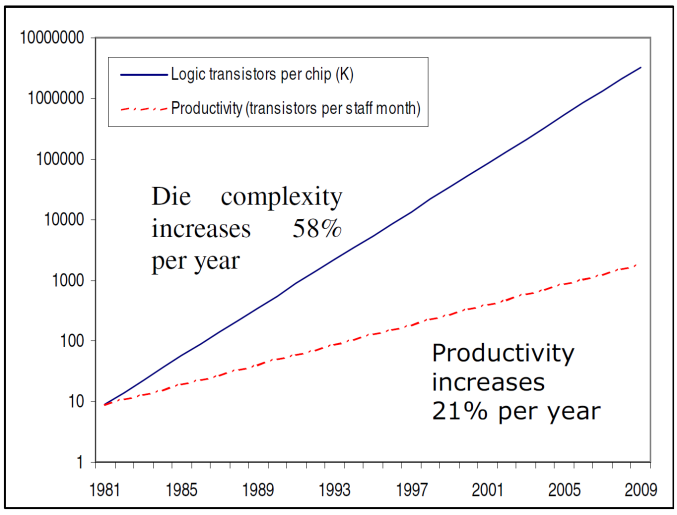

Figure 8: Design complexity (transistors per chip) and design productivity as a function of time. [17]

At the system level, hardware/software partitioning mentioned in Section 2.4.1 can be considered one of the most crucial steps to optimizing the hardware size by offloading the components to the software side. Besides this, the choice of logic styles for hardware implementation offers various alternative datapath design that aids in area optimization [28]. Additionally, system-level software present at this level can also play a vital role in area optimization. It defines the requirements of instruction memory size based on the code density. The memory size is also influenced by the amount of data flow which again can be controlled by the software. In a nutshell, designers can carefully construct an efficient program that exploits the architecture effectively, reducing the unnecessary hardware which ultimately reduces the overall die area.

2.5 Summary

Throughout this chapter, the state-of-art related to RISC-V ISA, a generic processor implementation based on RISC-V instructions, control software in a telecommunication system, and PPA analysis are explored. It begins by explaining the evolution of the RISC-V ISA, and the support this ISA received from the open-source communities, universities, and commercial industries. Regarding the instruction set, the chapter presents the structure of base architecture, instruction format, base instructions, and some standard extensions, such as 'M', 'C', 'Zicsr', 'Zicount', and 'Zifencei'.

The chapter also describes a generic RISC-V processor implementation based on the subset of the RISC-V instructions, presenting various functional units and their interconnects. It further elaborates on the advantage of pipelining and the potential hazards that may arise due to the pipelining. Then, the chapter explains the overview of control software in 5G telecommunication, exploring the evolution of the software

starting from the 1G. Finally, the chapter discusses the trade-off between power, performance, and area in terms of the hardware-software ecosystem.

In a nutshell, this chapter of the thesis serves to provide the necessary context for the power, performance, and area analysis of 5G-standard control software across RISC-V processor implementations.

3 State of the Art

Since the initiation of the RISC-V ISA, the number of processors that implement the ISA is steadily increasing. A list of available cores and SoCs have been maintained and consistently updated by the RISC-V community [29]. Nonetheless, the list appears far from complete, particularly since some smaller projects are not listed. The provided knowledge is an appropriate place for investigating an existing implementation for further analysis and comparison with in-house implementation.

Schiavone et al. in [30] assesses the ultra-low-power RISC-V cores, such as Riscy, Micro-riscy, and Zero-ricy. These cores are evaluated under different workloads for the analysis of changes in energy consumption due to different microarchitectures, timing constraints, operating frequency, and voltage. However, the assessment emphasizes merely processor cores aiming for low-power applications. Moreover, all three cores under evaluation possess architectural differences, and the workloads used for the evaluation are common benchmarks usually used to compare smaller RISC-V implementations.

A comparative survey of application-class RISC-V processor implementation [31] compares CVA6, Rocket, SHAKTI, and BOOM processors by executing similar benchmarks on common configuration settings. The results show large variations in processing performance, area coverage, energy efficiency, and resource utilization which highlights the importance of processor selection. However, the processors evaluated in this study vary in implementation due to their differences in the number of pipeline stages and their order of execution. Moreover, the benchmarks used for the evaluation are generic ones that are not designed to evaluate the networking processors.

In [32], a comprehensive evaluation of 32-bit RISC-V processors is executed by exploiting the TaPaSCo framework [33]. Eight 32-bit open-source processors are thoroughly evaluated across four FPGA platforms concerning resource utilization and processing performance. However, the TaPaSCo framework limits the evaluation of 32-bit architecture only to FPGA technology.

The authors of [34] analyze 10 RISC-V cores to find a suitable core for power- and cost-efficient IoT devices. The maximum clock frequency, energy efficiency, and area estimation of the processors are evaluated based on Vivado v2018.02 synthesis report. However, the comparison is solely based on FPGA technology and the study emphasizes only low-power and low-cost attributes, excluding criteria, such as performance and application coverage.

Unlike general-purpose processors, processors in telecommunication networks are enhanced to employ a control plane, data plane, packet processing networking functions, and large data movements [35]. Standard benchmarks for telecommunication processors do not exist yet, and available benchmarks created to assess the performance of

compute-intensive applications are not suitable for telecommunication processors.

This study is aware of some of the major benchmarking efforts for network processors, such as CommBench [36], NpBench [37], EEMBC [38], and NetBench [39]. However, these benchmarking approaches do not emphasize the system-level interface on which the performance of these processors is heavily dependent [35]. Therefore, this thesis investigates the performance and power efficiency of processors across Nokia's 5G standard control kernels which give particular emphasis to the system-level interface of the processor.

4 Benchmarking Framework

To accomplish the objective of the thesis discussed in Section 1.1, a precise set of requirements should be laid out. Figure 9 illustrates the framework designed to render the essential requirements. Chapter 2 familiarizes the theoretical concept crucial to this thesis work. This Chapter describes the methods used and the processes followed to realize the aim of this thesis study. The chapter begins with the introduction of control kernels/algorithms that were exploited for profiling the processors. It further describes the open-source processor and an in-house processor which are selected for benchmarking. Lastly, it explains the procedure followed during the simulation, synthesis, and power analysis of both processors.

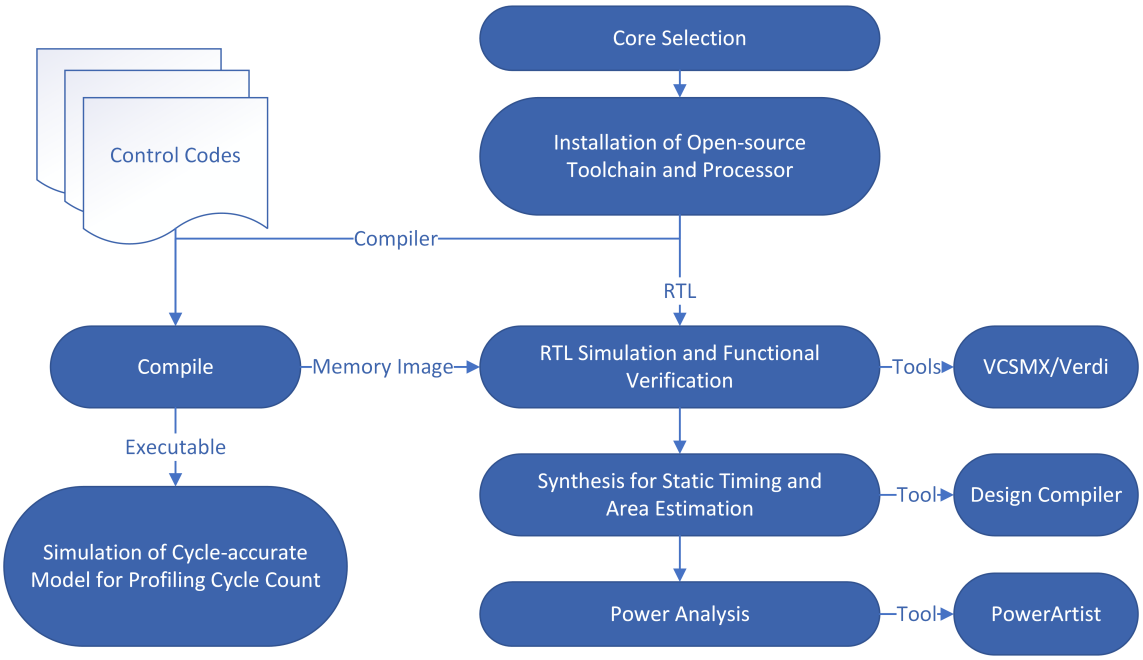

Figure 9: An overview of thesis methodology.

4.1 The Benchmarks

In recent years, the interaction between programming language and computer architecture has drawn enormous interest because this area provides huge opportunities for PPA optimization. Several pieces of research have been concentrated on benchmarking these PPA parameters to evaluate the existing microprocessor and microarchitecture design as well as to examine the area of development for further enhancement. Benchmark in this study refers to a control software/algorithm that reflects the frequency of source language construct in real programs and its corresponding machine language construct after compiled form. This benchmark will evaluate the efficiency of both processor architecture and the competence of the compiler to produce efficient code.

For any benchmark suite, a depiction of a wider application class in the field of interest is the most desired property. Processors evaluated in this study were intended to utilize in a telecommunication network, therefore 5G standard control kernels were used as benchmarks. Each benchmark suite is discussed separately in the upcoming sections.

4.1.1 Fast Control Plane Algorithm (FCPA)

FCP (Fast Control Plane) algorithm is designed to parse the Open Radio Access Network (O-RAN) messages. Specifically, the software is implemented to process only the section type 1 messages and extensions of type 1, described in the O-RAN fronthaul specification [40]. However, the software examines the available values for specific types and generates an error message in other instances. Typically the application parses the message, extracts the information from the message, and manages the storage of this information. The code also measures its performance in parsing packets of various sizes via cycles count by generating its own test data. [41]

4.1.2 PMI/RI/CQI Selection (PRCS)

The PRCS control code organizes the network system based on some of the parameters that affect the 5G performance. These parameters are the Pre-coding Matrix Indicator (PMI), a Rank Indicator (RI), and a Channel Quality Indicator (CQI). RI indicates the number of data streams or suitable transmission rank. PMI indicates the index of the codebook for pre-coding based on the selected rank. And CQI describes the amount of information carried by the physical channel defined by the 3GPP standard [42]. The 3GPP specification required a closed-loop feedback system where the UE provides these parameters to the base station. This control code in the base station orchestrates the network based on these parameters and network conditions [43].

4.1.3 Shrinkage Method for Covariance Matrix Estimation (SMCME)

The SMCME is a benchmarking suite for covariance matrix estimation. Covariance matrix estimation is a fundamental problem in signal processing in 5G systems. Signal processing largely relies on precisely estimated covariance matrices [44]. The purpose of the shrinkage method is to improve covariance matrix estimation for high dimensional covariance matrices with a small number of samples [45]. There are numerous configurations and various algorithms for the selection of the optimal covariance matrix. This control kernel organizes those algorithms based on different configurations and controls the network system.

4.2 The Processors

The preliminary step of this thesis work was the selection of an open-source processor to benchmark against Nokia's in-house processor, called NRISCV in this thesis. To meet the architectural similarities of the in-house processor, this thesis is interested in RISC-V open-source 32-bit cores suitable for Application Specific Integrated Circuit (ASIC) simulation and synthesis.

Features \ Cores	NRISCV	Rocket	BottleRocket	Potato	VexRiscv	CV32E40P
Achitecture	32-bit	32-bit	32-bit	32-bit	32-bit	32-bit
Extensions	I, M, C	I, M, A, F, D	I, M, C	I,	I, M, C, A	I, M, C, F
Pipeline Types	In-order	In-order	In-order	In-order	In-order	In-order
Pipeline Stages	5	5	3	5	5	4
HDL	System Verilog	Chisel	Chisel	VHDL	SpinalHDL	System Verilog
Documentation	Excellent	Insufficient	Insufficient	Insufficient	Insufficient	Excellent

Table 5: Reviewed RISC-V Cores Comparison.

Although the number of the RISC-V processor implementation is steadily increasing. Considering the processor for architectural similarity, licensing, proper documentation, and competitiveness reduces the option to only a handful of processors. Table 5 presents some of the RISC-V cores which have been reviewed as a candidate for benchmarking against NRISCV. Rocket core is a 5-stage in-order core, with support for the RV32I and RV64I ISAs. However, the rocket core is not the best solution for this study due to the lack of documentation that describes the modification of 32-bit implementation from 64-bit architecture. Moreover, the core is written in Chisel, which is not a well-known HDL by the community, creating barriers to understanding and modifying the code. BottleRocket is a 32-bit architecture built as a customized microarchitecture from components of the Free Chips Project Rocket core. However, the core only has 3 stages pipeline architecture. Potato core is a 32-bit implementation written in VHDL and supports the CSRs instructions. Unfortunately, the core is neither documented nor implemented for ASIC synthesis. VexRisc is a 32-bit core written in SpinalHDL whose pipelined architecture can be configured with 2 to 5 stages. The SpinalHDL is an open-source high-level description language regarded as a fork of Chisel, requiring a proper knowledge of the language for modification. Furthermore, the core architecture of the VexRisc is not described in the available documentation.

After an intensive investigation of available popular open-source cores, depicted in the Table 5, the CV32E40P core from the OpenHW Group was selected. Though the core has 4 stages pipelined architecture, it has excellent documentation and a well-explained verification procedure. The core also has its user manual that describes the architectures, supported features, existing peripherals, interfaces between components, and configuration options.

4.2.1 CV32E40P

CV32E40P [46] is a 4-stage in-order 32-bit RISC-V processor core that implements RV32IM[F]C ISA. The core supports floating point extension and a custom extension called Xpulp optionally. With the Xpulp extension, the core can achieve higher code density, greater performance, and lower energy consumption [47] [30]. The architecture of the core has been developed to perform at Near-Threshold voltage to increase the efficiency of transistors.

Primarily, CV32E40P was initiated as a fork of the OR10N CPU core that was based on the OpenRISC ISA. In 2016, the core changed its name to RI5CY under the PULP platform [48] team and it became a RISC-V compatible core. The core had been maintained as RI5CY until February 2020. Since then the core has been contributed by OpenHW Group as CV32E40P.

Figure 10 represents a block diagram of the CV32E40P. The diagram depicts the four stages: Instruction Fetch, Instruction Decode, Execute, and Write-Back, separated by the registers designed to communicate between neighboring stages. The pipeline design of the CV32E40P is fully independent i.e., each stage can complete its execution regardless of the state of its neighboring stage.

The CV32E40P core fully supports the Base Integer Instruction Set with additional Standard Extensions, such as M, C, Zicount, Zicsr, and Zifencei, described in Subsection 2.1.4. The MULT block present in the diagram represents the M extension, while the compress decoder block represents the C extension. The instance for the floating-point unit is unavailable in the diagram as it is optional in the CV32E40P core. Furthermore, the core also features PULP-specific extensions such as ALU extensions, hardware loops, post-incrementing load and stores, and multiply-accumulate extensions [46] [49].

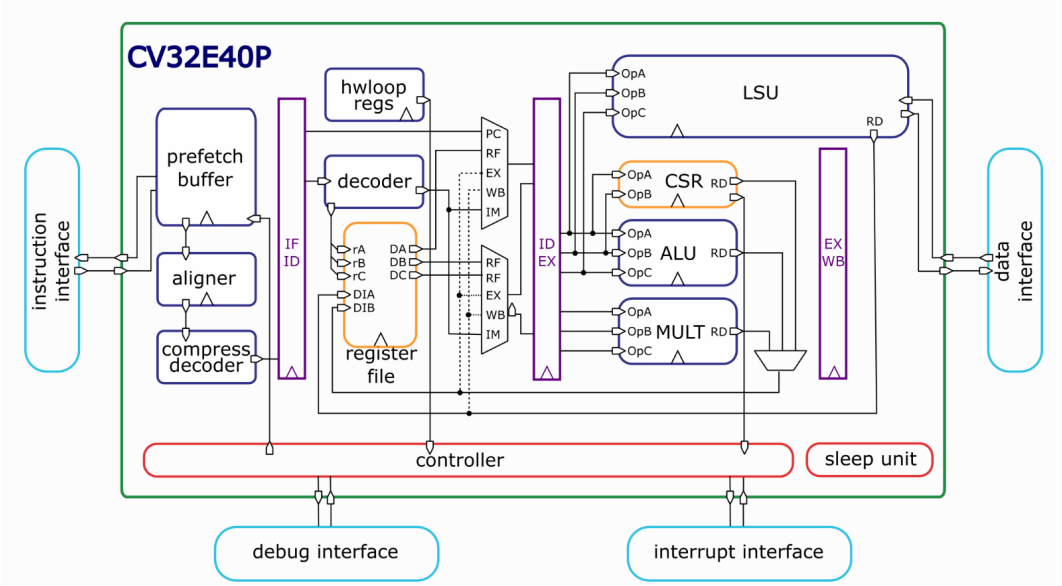

Figure 10: CV32E40P Core Overview. [46]

4.2.2 NRISCV

NRISCV is a Nokia proprietary core that was opted for the benchmarking. It is a 5-stage in-order 32-bit RISC-V core. The core supports standard extensions of Integer Multiplication and Division Instructions (M) and Compressed Instructions (C).

Processors / **Features**	**CV32E40P**	**NRISCV**
Data Width (Bits)	32-bit	32-bit
Pipeline Type	In-order	In-order
Pipeline Stages	4	5
Extensions	M and C	M and C
Hardware Loops	Yes	Yes
Sleep Unit	Yes	No
Debug Module	Execution-based	More Features
Trigger Module	Only to enable entry into debug mode	Yes
HDL	System Verilog (Handwritten)	System Verilog (Generated)
License	SolderPad	Proprietary

Table 6: An architectural overview of CV32E40P and NRISCV

Table 6 presents an overview of major architectural similarities and differences possess by CV32E40P and NRISCV processors. Both processors have 32-bit architecture and offer in-order pipeline type. However, the pipeline depth of the CV32E40P is 4 stages and the NRISCV is 5 stages. Other features, such as hardware loops and extensions like 'M' and 'C' are common to both processors. Although the CV32E40P offers the optional floating point 'F' and Custom 'Xpulp' extensions, these extensions were excluded for a fair comparison. Debug and Trigger Module present in the CV32E40P and NRISCV are different in nature. CV32E40P offers support for execution-based debug according to the RISC-V Debug Specification, version 0.13.2 [50]. It also supports a Trigger Module only to enable entry into debug mode on a trigger event. It has one trigger register and supports only instruction address match trigger type. On the other hand, Debug and Trigger Module of NRISCV supports more features. In NRISCV, a configuration of the triggers in the Trigger Module requires the use of the system instructions to update at least 3 or more registers. This Trigger Module supports match trigger, icount trigger, and chaining features. Despite having an RTL description of both cores in System Verilog, the HDL of CV32E40P was handwritten but for NRISCV it was generated from high-level synthesis (HLS) language. Lastly, the copyright and related rights of the CV32E40P processor are licensed under SolderPad Hardware License, whereas NRISCV is proprietary to Nokia.

4.3 Methodology and Procedure

This section explores the procedure and methods employed for this thesis work in four distinct phases. Section 4.3.1 describes the processes involved in the compilation of control software. Section 4.3.2 presents the procedure for RTL simulation and functional verification. Section 4.3.3 discusses the measure taken during synthesis. Lastly, Section 4.3.4 proposes the PowerArtist tool for power analysis.

4.3.1 Compilation of Control Software

Once the CV32E40P processor was selected for benchmarking, the toolchain that supports the CV32E40P processor i.e., RISC-V GNU compiler toolchain and the processor itself were installed. The installation procedure of the toolchain is explained in [3] and the processor CV32E40P is described in [51].

The control software, described in Section 4.1 was then compiled with both CV32E40P and NRISCV toolchains. The compiler then produced the executable and memory-image files. The executable was used for the simulation of the cycle-accurate behavior model and the memory-image file was used for Register Transfer Level (RTL) simulation. Since the Verilator was used to generate a cycle-accurate model of CV32E40P, the model was exploited for profiling cycle counts for each control software.

4.3.2 RTL Simulation and Functional Verification

The simulation process was performed by the VCSMX and the Verdi from Synopsys. VCSMX is a simulation engine for analyzing, compiling, and simulating a design described in VHDL, Verilog, mixed-HDL, OpenVera, SystemVerilog, and SystemC [52]. Verdi is an automated debug system for debugging digital designs and verification flows which increases design productivity [53]. VCSMX and Verdi work together to compile, simulate and debug the digital design, and both tools must be of the same version for compatibility. The RTL simulation process has three inputs: a testbench written in HDL, a memory image of a benchmark, and the actual HDL design.

In the case of the NRISCV processor, the Chess/Checkers retargetable tool suite [54] was used for instruction set simulator (ISS) simulation. The tool suite is popular for designing the architecture and the implementation of application-specific instruction set processors (ASIPs) as well as debugging and programming them. The working principle of the tool suite can be described in three steps: first, the tool suite transforms the source code into machine code for the target processor; second, it generates a retargetable ISS generator that constructs a cycle-accurate ISS for the target processor; third, it generates an HDL generator that generates a synthesizable RTL model of the target processor core. A bash script was created to automate the whole process so that the tool suite first generates the basic components needed for simulation and then simulates a control kernel on both the ISS and the RTL. Finally, Register Change Dump (RCD) files generated from both the RTL simulation and the ISS simulation for each benchmark were compared. The similarity in the output of both simulations validates the functional correctness of the core.

In the case of the CV32E40P processor, the testbench and the actual HDL design were obtained from the repository [55] maintained by the OpenHW Group. The memory-image of the benchmark was generated by the simulator provided by the same developer community. The existing Makefile that was obtained from the framework of NRISCV RTL simulation was modified to accommodate the CV32E40P core and the 5G standard control kernels. A Makefile is a program-building tool that helps in automating software building procedures and other complex tasks with dependencies. The compilation process converts the processor core from a source-level description into a binary database. Then the process reads and links the memory image subjected to the compilation process and produces the output to the console. The similarity in the output of the ISS simulator and the RTL simulator for each benchmark validates the functional correctness of the core.

The OpenHW Group does not provide the module ‘cv32e40p_clock_gate’, which has a clock gating cell, as part of the RTL design because these cells are exclusive to the selected target technology. Therefore, the clock gating cell module was designed according to the manual [46], and placed in the RTL design. Besides, the clock gating cell, the core was also missing ‘riscv_defines.sv’ and ‘riscv_config.sv’ files, which define the essential configuration for simulation. These files were taken from the RI5CY

core contributed by the PULPino platform. Since the core CV32E40P was modified from the RI5CY, these files were accepted by the simulation process without a change.

The simulator also produces a Flatfile Streaming Database (FSDB) file, which is used for storing simulation waveform data. However, this thesis is interested in RTL simulation only for the examination of the functional correctness of the processor cores. Therefore, it is sufficient from the perspective of this thesis that the similarity in the output of the RTL simulation and the output of the ISS simulation, validates the correctness of the logic in the design.

4.3.3 Synthesis

Typically, functional verification is followed by a synthesis process in the digital design flow. The synthesis process is significant in this study for the static timing analysis and area analysis. Synthesis is a process of generating an optimized gate-level representation from a high-level description of the design, given a standard-cell library and design constraints [56]. Standard-cell library is vendor specific. This thesis work used Synopsys's Design Compiler (DC) for synthesis. Synthesis only provides the information about the standard cells and their sizing, that will be used in the layout. Therefore, the characteristics of the actual layout might fluctuate from the ones reported by synthesis due to the actual placement and the actual route.

In the case of NRISCV, the synthesis process was straightforward, as this thesis exploited the framework that has been well-established and maintained by a team responsible for developing the core itself. The framework also contains a file that defines the constraints and the process corners. The entire synthesis process was automated by Makefile. The maximum processor core frequency was achieved by increasing the clock until timing violations were recorded. These timing constraint violations are known as setup/hold time violations and are associated with the critical path of the design. Setup time is the minimum amount of time required for the synchronous input data to be stable before capturing the active edge of the clock. Whereas hold time is the minimum amount of time required for the synchronous input data to be stable after capturing the active edge of the clock [57]. Any transition in data during either of these two time-margins is known as a timing violation.

In the case of CV32E40P, a similar framework with the same technology, constraints, and process corners was used for the synthesis. RTL files of the CV32E40P core offers two flavors of register files, namely, latch-based register file and flip-flop-based register file. CV32E40P document [46] suggests the use of a flip-flop-based register file for ASIC synthesis as it has been verified. The missing clock gate module discussed in the Chapter 4.3.2 was added to complete the RTL description of the design. The maximum processor core frequency was obtained by applying a similar approach to the NRISCV.

4.3.4 Power Analysis

In a digital design flow, power analysis is performed to examine the design in terms of essential power specification. Power analysis is performed before chip fabrication and usually follows the synthesis process. Power analysis in this thesis was executed by using Apache licensed ANSYS PowerArtist tool. Since the thesis is particularly interested in the power estimation values due to the different control software, a dynamic power analysis was performed.

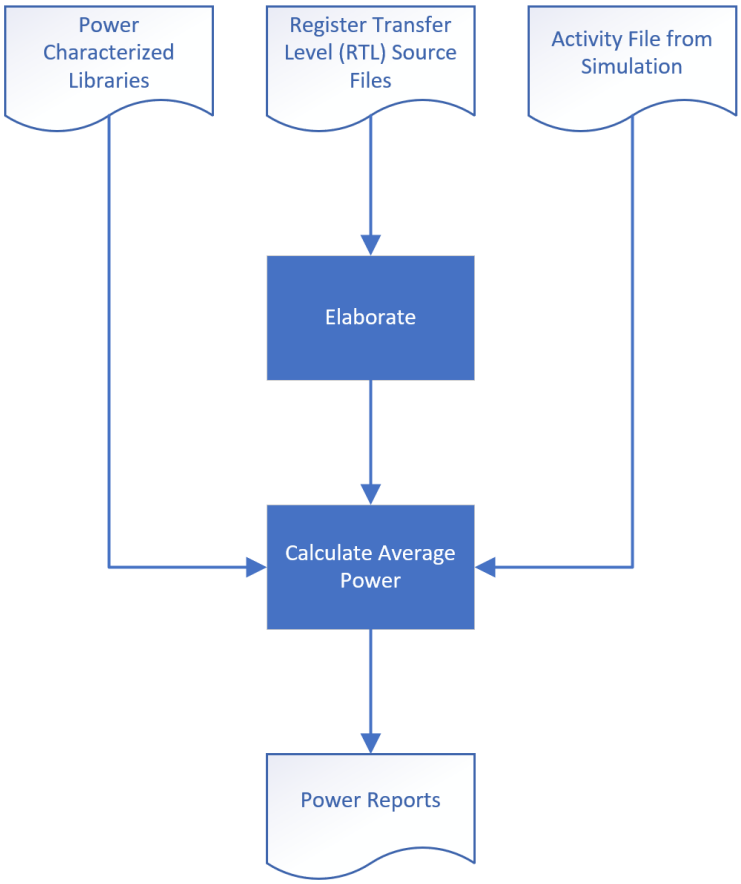

Figure 11: Flow chart explaining the process of power analysis.

Figure 11 illustrates the basic process for generating power reports with the help of the PowerArtist tool. First of all, the tool compiles the HDL description of the design into a binary format, called the scenario file. This process is known as the elaboration phase. After the elaboration, the tool analyses the activity file produced during simulation, especially monitoring the activity at the input clock pin of registers. Finally, the tool performs an average-based power analysis and generates the power reports.

In the case of NRISCV, generating a power report was fairly simple. The team responsible for the development of the core has also maintained a framework for

the power analysis with the help of Makefiles and python scripts. In the case of CV32E40P, the framework for the power analysis of NRISCV was modified to accommodate the CV32E40P core and a power report was generated. These power reports will be analyzed further in Section 5.3.

5 Results and Analysis

This chapter presents the benchmarking results accompanied by their analysis. The chapter is categorized into three sections for individual analysis and discussion on factors, such as processing performance, area estimation, and power consumption.

5.1 Performance

The performance of the processors was evaluated based on 5G standard control kernels. Benchmarking processors generally involve various attributes, and the influence of these attributes on the computing performance relies on the underlying processor architecture and associated compiler. This study has prioritized cycle count and synthesizable clock frequency as evaluation factors for the performance measurements.

Cycle Count is the first attribute this thesis was interested in regarding the performance evaluation. The performance of the processor can be measured in a number of clock cycles, which is a relative measure. Cycle count is a generic measure that is rather tied to the architecture of the processor. Every processor is accompanied by a physical clock that oscillates at a certain frequency. The amount of time between two pulses of an oscillation is a clock cycle. The clock cycle is independent of frequency rather it is a determining factor in dictating the minimum frequency. A cycle count is a popular method of benchmarking different algorithms or different platforms.

Figure 12 represents the number of instructions retired and Figure 13 represents the number of cycles retired while executing those instructions for different control kernels. The bar graphs depicted in Figures 12 and 13 are in logarithmic scale for better visibility. Figure 12 illustrates that the number of instructions retired while executing FCPA is marginally higher in NRISCV but for PRCS, the number is higher in CV32E40P. However, the number of instructions executed for SMCME in CV32E40P is almost twice of NRISCV. This huge difference in executed instructions for SMCME shows no significant effect on the average number of instructions as the total number of instructions retired for SMCME is substantially lower in comparison to FCPA and PRCS. The difference in the number of instructions retired in both processors while executing the same benchmarks describes the nature of the compiler.

Figure 13 displays that the cycle count exhibits similar characteristics as of instructions count for all the control kernels. The number of cycles taken while executing FCPA is around 1.5 times higher in NRISCV but for PRCS, the number is roughly 1.5 times higher in CV32E40P. Although the number of instructions executed for SMCME significantly varies in both processors, the number of cycles required to execute the SMCME in both processors is significantly close. Considering the influence of all the control kernels on both the processors as depicted in Figures 12 and 13, the average number of cycles taken to execute a similar amount of instructions in NRISCV is comparatively more than that of CV32E40P.

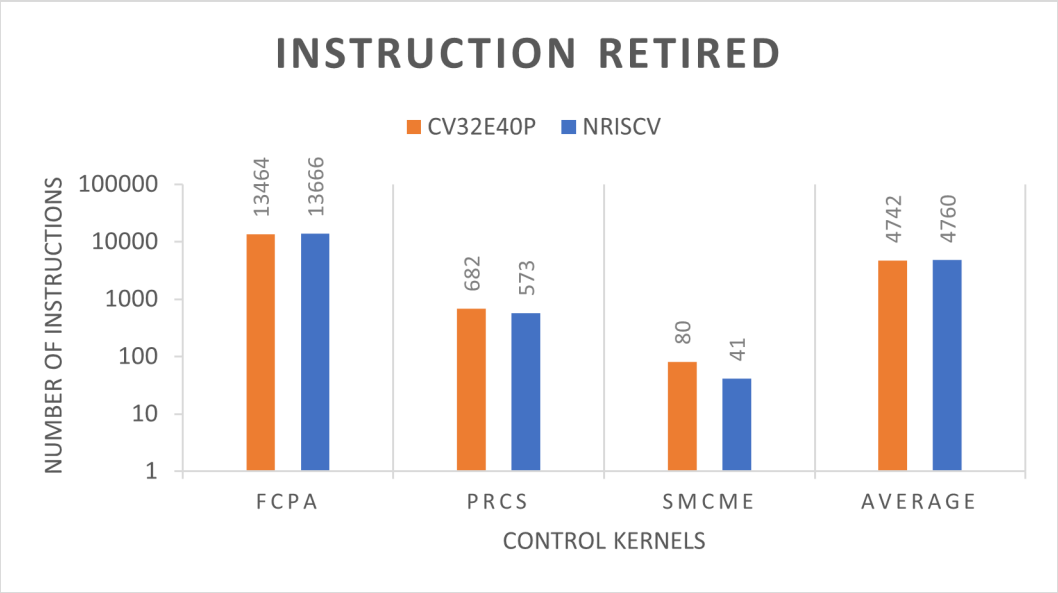

Figure 12: Instructions Count for various Control Kernels in different Processors.

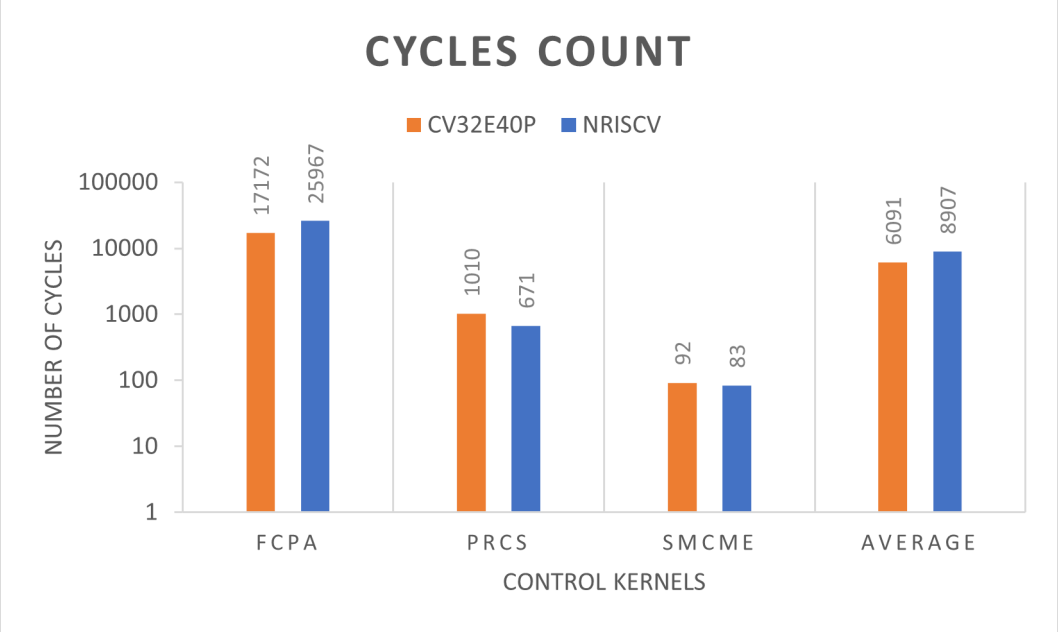

Figure 13: Cycles Count for various Control Kernels in different Processors.

The maximum synthesizable processor frequency was another attribute this thesis was interested in regarding performance evaluation. The maximum synthesizable frequency of a processor is a measure of its maximum operating speed. This is typically determined by the design of the processor and the technology used to manufacture it. In general, a processor with a higher maximum synthesizable frequency will be able to perform more operations per second, which can improve the overall performance of a system. The CV32E40P processor managed to perform synthesis at a maximum

clock rate of 1550 MHz and the NRISCV core managed to perform synthesis at 1950 MHz without timing violation. These results are technology dependent and specified for 7nm TSMC ASIC synthesis. These results indicate that the NRISCV has improved performance and efficiency over CV32E40P for a task that requires a high amount of processing power.

5.2 Area

The comparison of processor size reveals the intricacy of the core architecture. This thesis assumes that the area estimated for both processors based on the area report generated from the synthesis suffices the objective. However, the area report may differ from the actual layout due to the reason mentioned in Section 4.3.3. The ASIC synthesis was accomplished for the processors under relaxed clock constraints of 1 GHz. The cores exhibit minute area gain for tight timing constraints.

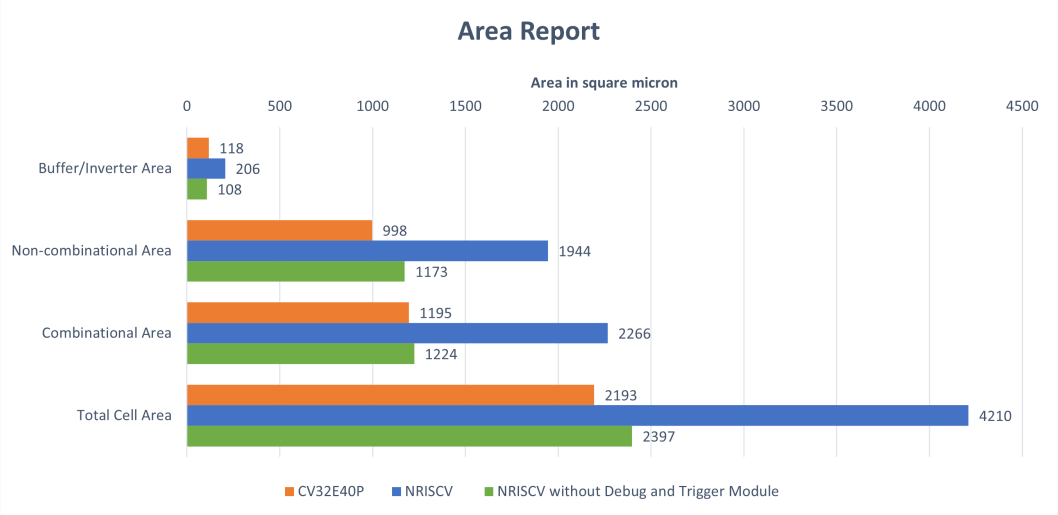

Figure 14: Area Report

The Synopsys DC reports the total area of the design, including an area breakdown of each of the contained modules. The report provides the total buffer/inverter area, non-combinational area, and combinational area separately as illustrated in Figure 14. All these separate module areas are added together to form a cell area. Area values listed in the diagram are rounded to the nearest decimal values and use square microns for units.

Figure 14 shows that the total cell area required for CV32E40P is nearly half of the NRISCV. This ratio is consistent in each separate module, such as buffer/inverter area, non-combinational area, and combinational area. This huge difference in area coverage is mainly due to the nature of Debug and Trigger Module implemented in the CV32E40P and NRISCV which is explained in Section 4.2.2. The Trigger

module in NRISCV nearly covers half the area of the overall NRISCV. The area cost of NRISCV while excluding Debug and Trigger Module is more comparable to CV32E40P, as can be seen in Figure 14.

However, there is no apple-to-apple comparison between these two processor cores. The NRISCV core does not have instructions for accessing everything (e.g. reading program memory). If this core opts for execution-based debug like CV32E40P, then the instructions for accessing program memory need to be added, which adds more area. But in the case of a perfect RISC-V core like CV32E40P, where everything can be accessed with instructions, an execution based debug takes less area compared to debug module that is directly connected to the registers and memories.

Another important reason behind the difference in area coverage of both the processors can be directly related to the HDL description of each core. Since the HDL description of the CV32E40P was hand-written, the core was carefully designed by accounting for the area cost. While the HDL description of the NRISCV was generated by the compiler from a high-level language, the generated HDL was found less efficient compared to the hand-written core in terms of area optimization.

5.3 Power Estimation

Though Synopsys DC provides the power estimate report of the design, this thesis opted for the ANSYS PowerArtist tool for more accurate power estimation. The report provides the internal power, switching power, and leakage power. The internal power corresponds to the cell's internal nodes. Switching power occurs at the cell's input and output nodes. These internal power and switching power combine to form a dynamic power.

Figure 15 illustrates the power consumption in both processors due to various control kernels. The total power consumption in CV32E40P is roughly 2.5 times lower than that of NRISCV for each control kernel. The figure also demonstrates the internal, switching, and leakage power as a constituent of the total power consumption for both cores. The figure illustrates that the leakage power dissipation contributes the highest to the total power dissipation, while the switching has the lowest contribution among the three components. The internal power dissipation of CV32E40P is around $5-7$ times lower than that of NRISCV depending on the specific control kernel.

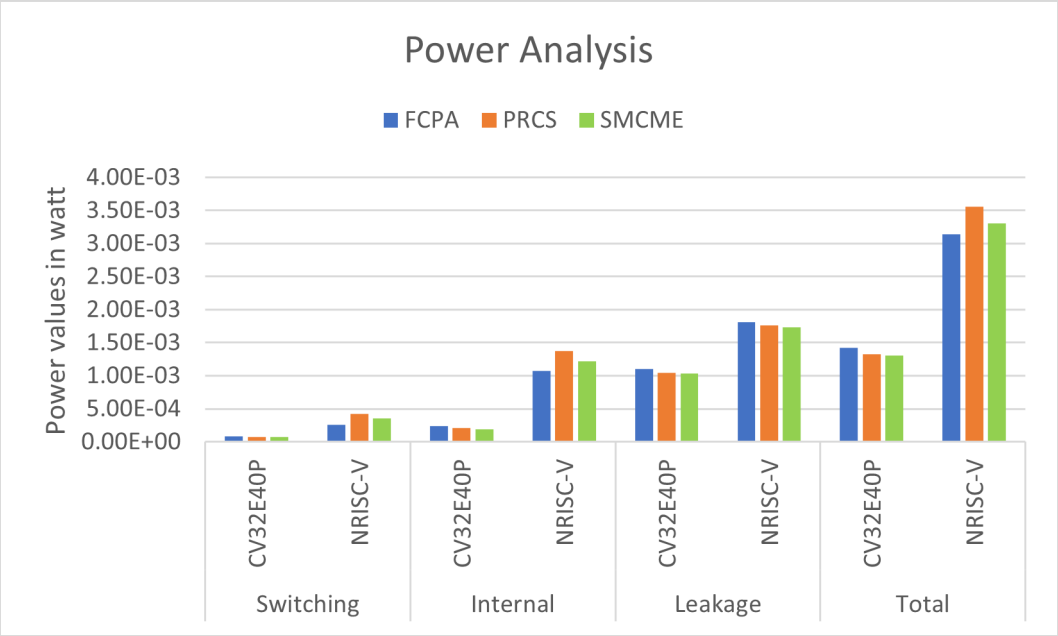

Figure 15: Power Analysis Report of NRISCV and CV32E40P.

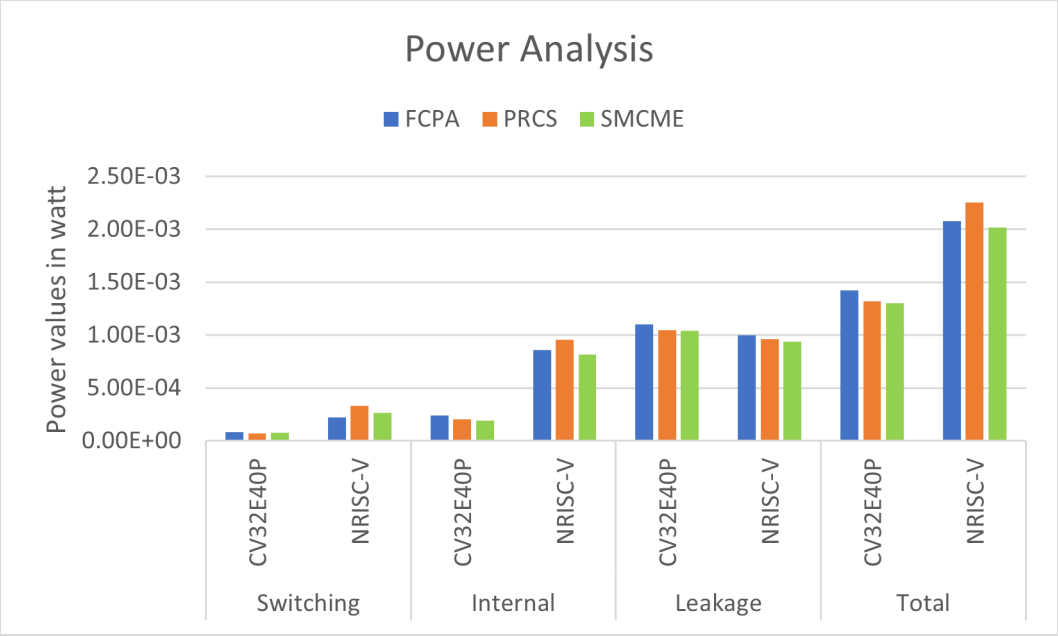

Figure 16: Power Analysis Report of NRISCV without Debug and Trigger Module and CV32E40P.

Comparing the power analysis reports of NRISCV without Debug and Trigger Module and CV32E40P seems more reasonable due to the reason discussed in Section 5.2, which is presented in Figure 16. The bar graph illustrates that the power consumption in CV32E40P is still less than that of NRISCV even after excluding Debug and Trigger Module from the design. However, the difference is reduced to roughly 1.5

times from the earlier difference of around 2.5 times when the Debug and Trigger Module was present in the processor design. Unlike Figure 15, the Figure 16 depicts that the CV32E40P dissipates more leakage power after excluding Debug and Trigger Module from NRISCV. The Debug and Trigger Module in the NRISCV appears to be dissipating roughly half the leakage power of NRISCV. The higher leakage power in CV32E40P means that the core has higher power consumption than that of NRISCV when they are not actively performing any task or executing instructions.

However, the switching and internal power dissipation appear to be reasonably larger in NRISCV. The internal power dissipation is still 3.5−5.5 times lower in CV32E40P depending on the control kernels. The overall power dissipation of the NRISCV processor is larger than that of CV32E40P. This difference in total power consumption is largely contributed due to the huge internal power dissipation of the NRISCV. Since both processors were synthesized with the same technology, the larger amount of power consumption in the NRISCV is mainly due to the larger size of the processor, which is shown in Figure 14. The larger the size of the processor the larger the number of transistors and complex architecture design, which affects the amount of power consumption.

Another important reason behind the difference in power consumption between the two cores is their microarchitecture. The NRISCV has a longer pipeline depth in comparison to CV32E40P, which leads to a longer datapath and a larger number of registers. This extra hardware present in the NRISCV leads to larger real-estate and higher power consumption. Besides this, the CV32E40P has a sleep unit that is absent in the NRISCV, benefiting the processor in power optimization. Furthermore, the handwritten HDL for CV32E40P over a generated HDL for NRISCV is another attributes that make the CV32E40P more power efficient.

6 Conclusion

This thesis evaluated the usability of an open-source RISC-V processor core, CV32E40P, in the telecommunication network. The CV32E40P core was assessed against Nokia's in-house core, NRISCV. The fair comparison is based upon common configuration settings and execution of the same workloads on an identical platform.

The result shows significant differences in terms of processing performance, power consumption, and area utilization. If cycle count is taken as the primary evaluation factor for the performance, it is hard to declare a clear winner. The behavior of both processor cores entirely varies depending on the nature of the control software. For SMCME and PRCS, NRISCV exhibits a lower cycle count, while for FCPA, CV32E40P exhibits a lower cycles count. In terms of power and area estimation values, the CV32E40P core appears to be power efficient and possesses a smaller footprint than the NRISCV core. However, these differences in area and power results were mainly due to the different microarchitectures of the two processors.

This paper differs regarding the benchmarking practices exercised to compare the RISC-V cores. Generally, the cores have been evaluated under generic benchmarks that are not explicitly from the targeted application field. This paper examined the processor core targeting telecommunication network system by using system-specific benchmarks.

The CV32E40P evaluated in this paper is actively strengthened by many contributors and the results presented here vary by each version. Therefore, the results discussed in this paper are a snapshot in time.

References

[1] A. H. Celdrán, M. G. Pérez, F. J. García Clemente, and G. M. Pérez, "Automatic monitoring management for 5g mobile networks," *Procedia Computer Science*, vol. 110, pp. 328–335, 2017, 14th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2017) / 12th International Conference on Future Networks and Communications (FNC 2017) / Affiliated Workshops. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050917312802

[2] Risc-v international. [Online]. Available: https://riscv.org/

[3] K. Cheng. Gnu toolchain for risc-v. [Online]. Available: https://github.com/riscv-collab/riscv-gnu-toolchain

[4] LLVM Community. Llvm 9.0.0 release notes. [Online]. Available: https://releases.llvm.org/9.0.0/docs/ReleaseNotes.html

[5] xpack gnu risc-v embedded gcc v8.2.0-3.1 released. [Online]. Available: https://xpack.github.io/blog/2019/07/31/riscv-none-embed-gcc-v8-2-0-3-1-released/

[6] Spike risc-v isa simulator. [Online]. Available: https://github.com/riscv-software-src/riscv-isa-sim

[7] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual, volume i: User-level isa, version 2.1," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-118, May 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html

[8] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanović, "The risc-v instruction set manual volume ii: Privileged architecture version 1.9.1," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-118, May 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html

[9] D. Patterson and A. Waterman, *The RISC-V Reader: An Open Architecture Atlas*, 1st ed. Strawberry Canyon, 2017.

[10] S. Kaxiras and M. Martonosi, *Computer Architecture Techniques for Power-Efficiency*, 01 2008, vol. 3.

[11] N. Riasanovsky, "Understanding risc-v calling convention," EECS Department, University of California, Berkeley, Tech. Rep. [Online]. Available: https://inst.eecs.berkeley.edu/~cs61c/resources/RISCV_Calling_Convention.pdf

[12] Symmathics. Risc-v datapath. [Online]. Available: https://www.symmathics.com/index.php/risc-v-datapath-part-3/

[13] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[14] K. V. Cardoso, C. B. Both, L. R. Prade, C. J. Macedo, and V. H. L. Lopes, "A softwarized perspective of the 5g networks," *arXiv preprint arXiv:2006.10409*, 2020.

[15] TEOCO. Automated network configuration management for today's complex wireless networks. [Online]. Available: https://www.teoco.com/blog/automated-network-configuration/

[16] H. Sahlin, "Channel prediction for link adaptation in lte uplink," in *2012 IEEE Vehicular Technology Conference (VTC Fall)*. IEEE, 2012, pp. 1–5.

[17] M. Flynn, "Area - time - power and design effort: the basic tradeoffs in application specific systems," in *2005 IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05)*, 2005, pp. 3–6.

[18] P. Havinga, *Design techniques for energy efficient and low-power systems*. Netherlands: University of Twente, 2000, pp. 2.1–2.52, dissertation, Chapter 2 - Major parts of this chapter will be published in the Journal of Systems Architecture, 2000 [25] and were presented at the IEEE International Conference on Personal Wireless Communications (ICPWC'97), 1997 [24].

[19] J. M. Rabaey and M. Pedram, "Low power design methodologies," 1996.

[20] P. Arato, S. Juhasz, Z. Mann, A. Orban, and D. Papp, "Hardware-software partitioning in embedded system design," in *IEEE International Symposium on Intelligent Signal Processing, 2003*, 2003, pp. 197–202.

[21] L. Benini and G. d. Micheli, "System-level power optimization: techniques and tools," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 5, no. 2, pp. 115–192, 2000.

[22] *Hardware for Real-Time Systems*. John Wiley & Sons, Ltd, 2011, ch. 2, pp. 27–77. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118136607.ch2

[23] K. Kennedy and J. R. Allen, *Optimizing compilers for modern architectures: a dependence-based approach*. Morgan Kaufmann Publishers Inc., 2001.

[24] B. Hoefflinger, "Itrs: The international technology roadmap for semiconductors," in *Chips 2020*. Springer, 2011, pp. 161–174.

[25] A. S. Verhulst, B. Sorée, D. Leonelli, W. G. Vandenberghe, and G. Groeseneken, "Modeling the single-gate, double-gate, and gate-all-around tunnel field-effect transistor," *Journal of Applied Physics*, vol. 107, no. 2, p. 024518, 2010.

[26] F. Giannazzo, G. Greco, F. Roccaforte, and S. S. Sonde, "Vertical transistors based on 2d materials: Status and prospects," *Crystals*, vol. 8, no. 2, p. 70, 2018.

[27] D. Velenis, M. Stucchi, E. J. Marinissen, B. Swinnen, and E. Beyne, "Impact of 3d design choices on manufacturing cost," in *2009 IEEE International Conference on 3D System Integration*. IEEE, 2009, pp. 1–5.

[28] T. Callaway and E. Swartzlander, "Optimizing arithmetic elements for signal processing," in *Workshop on VLSI Signal Processing*, 1992, pp. 91–100.

[29] Risc-v cores and soc overview. [Online]. Available: https://github.com/riscvarchive/riscv-cores-list

[30] P. Davide Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand, and L. Benini, "Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, pp. 1–8.

[31] A. Doerflinger, M. Albers, B. Kleinbeck, Y. Guan, H. Michalik, R. Klink, C. Blochwitz, A. Nechi, and M. Berekovic, "A comparative survey of open-source application-class risc-v processor implementations," 05 2021.

[32] C. Heinz, Y. Lavan, J. Hofmann, and A. Koch, "A catalog and in-hardware evaluation of open-source drop-in compatible risc-v softcore processors," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2019, pp. 1–8.

[33] J. Korinth, J. A. Hofmann, C. Heinz, and A. Koch, "The tapasco open-source toolflow for the automated composition of task-based parallel reconfigurable computing systems," in *ARC*, 2019.

[34] D. Gookyi and K. Ryoo, "Selecting a synthesizable risc-v processor core for low-cost hardware devices," *Journal of Information Processing Systems*, vol. 15, pp. 1406 – 1421, 12 2019.

[35] P. R. Chandra, F. Hady, R. Yauatkar, T. Bock, M. Cabot, and P. Mathew, "Chapter 2 - benchmarking network processors," in *Network Processor Design*, ser. The Morgan Kaufmann Series in Computer Architecture and Design, P. Crowley, M. A. Franklin, H. Hadimioglu, and P. Z. Onufryk, Eds. San Francisco: Morgan Kaufmann, 2003, pp. 11–25. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978155860875750020X

[36] T. Wolf and M. Franklin, "Commbench-a telecommunications benchmark for network processors," in *2000 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS (Cat. No.00EX422)*, 2000, pp. 154–162.

[37] B. Lee and L. John, "Npbench: a benchmark suite for control plane and data plane applications for network processors," in *Proceedings 21st International Conference on Computer Design*, 2003, pp. 226–233.

[38] J. A. Poovey, T. M. Conte, M. Levy, and S. Gal-On, "A benchmark characterization of the eembc benchmark suite," *IEEE Micro*, vol. 29, no. 5, pp. 18–29, 2009.

[39] G. Memik, W. Mangione-Smith, and W. Hu, "Netbench: a benchmarking suite for network processors," in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, 2001, pp. 39–42.

[40] O-RAN Alliance, "O-ran fronthaul working group. control, user and synchronization plane specification." Tech. Rep., May 2022. [Online]. Available: https://www.o-ran.org/specifications

[41] O. Södergren, "Risc-v based application-specific instruction set processor for packet processing in mobile networks," Master's thesis, Linköping University, 2021.

[42] D. Ogawa, C. Koike, T. Seyama, and T. Dateki, "A low complexity pmi/ri selection scheme in lte-a systems," in *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*. IEEE, 2013, pp. 1–5.

[43] A. E.-R. Nada and A. M. H. Mehana, "A comparative study of pmi/ri selection schemes for lte/ltea systems," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 2, pp. 1444–1453, 2017.

[44] R. Abrahamsson, Y. Selen, and P. Stoica, "Enhanced covariance matrix estimators in adaptive beamforming," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, vol. 2, 2007, pp. II–969–II–972.

[45] Y. Chen, A. Wiesel, Y. C. Eldar, and A. O. Hero, "Shrinkage algorithms for mmse covariance estimation," *IEEE Transactions on Signal Processing*, vol. 58, no. 10, pp. 5016–5029, 2010.

[46] OpenHW Group. Cv32e40p user manual documentation. [Online]. Available: https://docs.openhwgroup.org/projects/cv32e40p-user-manual/intro.html

[47] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[48] Pulp platform. [Online]. Available: https://pulp-platform.org/

[49] P. D. S. Andreas Traber, Michael Gautschi, *RI5CY: User Manual*, April 2019.

[50] SiFive, Inc., *RISC-V External Debug Support*.

[51] OpenHW Group. Core-v-verif quick start guide. [Online]. Available: https://docs.openhwgroup.org/projects/core-v-verif/en/latest/quick_start.html#core-v-verif-quick-start-guide

[52] Synopsys, *VCSMX/VCS MXi User Guide*.

[53] Synopsys, *Verdi3 User Guide and Tutorial*.

[54] G. Goossens, D. Lanneer, W. Geurts, and J. Van Praet, "Design of asips in multi-processor socs using the chess/checkers retargetable tool suite," in *2006 International Symposium on System-on-Chip*, 2006, pp. 1–4.

[55] OpenHW Group, "Cv32e40p," https://github.com/openhwgroup/cv32e40p.

[56] S. Gayathri and T. C. Taranath, "Rtl synthesis of case study using design compiler," in *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, 2017, pp. 1–7.

[57] R. Abdollahi, K. Hadidi, and A. Khoei, "A simple and reliable system to detect and correct setup/hold time violations in digital circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 10, pp. 1682–1689, 2016.