Power Profiling Model for RISC-V Core

Lin Zhang

School of Electrical Engineering

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

Espoo 28.04.2023

Supervisor

Prof. Jussi Ryynänen

Advisor

Dr. Marko Kosunen

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 Lin Zhang

Title Power Profiling Model for RISC-V Core

Degree programme Electronics and Nanotechnology

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

Supervisor Prof. Jussi Ryynänen

Advisor Dr. Marko Kosunen

Date 28.04.2023 **Number of pages** 58+4 **Language** English

Abstract

The reduction of power consumption is considered to be a critical factor for efficient computation of microprocessors. Therefore, it is necessary to implement a power management system that is aware of the computational load of the CPU cores. To enable such power management, this project aims to develop a power profiling model for the RISC-V core. TheSyDeKick verification environment was used to develop the power profiling models. Additionally, Python-controlled mixed mode simulations of C-programs compiled for A-Core were conducted to obtain needed data for the power profiling of the digital circuitry. The proposed methodology could employ a time-varying power consumption profiling for the A-Core RISC-V microprocessor core which depends on software, voltage, and clock frequency. The results of this project allow for the creation of parameterized power profiles for the A-Core, which can contribute to more efficient and sustainable computing.

Keywords RISC-V, Power Profiling, Power Management, TheSyDeKick

Aalto University
School of Electrical Engineering

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Lin Zhang

Työn nimi Tehon Profilointimalli RISC-V Corelle

Koulutusohjelma Elektroniikka ja nanoteknologia

Pääaine Mikro- ja nanoelektroniikkasuunnittelu **Pääaineen koodi** ELEC3036

Työn valvoja Prof. Jussi Ryynänen

Työn ohjaaja TkT Marko Kosunen

Päivämäärä 28.04.2023 **Sivumäärä** 58+4 **Kieli** Englanti

Tiivistelmä

Virrankulutuksen vähentäminen on kriittinen tekijä mikroprosessorien tehokkaassa laskennassa. Siksi on tarpeen toteuttaa virrankulutuksen hallintajärjestelmä, joka tunnistaa CPU-ytimien laskennallisen kuorman. Tämän projektin tavoitteena on kehittää virrankulutusprofiilimalli RISC-V-ytimelle. TheSyDeKick-tarkistusympäristöä käytettiin virrankulutusprofiilimallien kehittämiseen. Lisäksi C-ohjelmille käännetyistä A-Core-mikroprosessoria varten suoritettiin Python-ohjattuja sekoitetun tilan simulointeja tarvittavien tietojen saamiseksi digitaalisten piirien virrankulutusprofiilien määrittämiseksi. Ehdotettu menetelmä voisi käyttää aikaan sidottua virrankulutusprofiilin määrittämistä A-Core RISC-V mikroprosessoriytimelle, joka riippuu ohjelmistosta, jännitteestä ja kellotaajuudesta. Tämän projektin tulokset mahdollistavat parametroitujen virrankulutusprofiilien luomisen A-Corelle, mikä voi edistää tehokkaampaa ja kestävämpää laskentaa.

Avainsanat RISC-V, Tehon Profilointi, Virranhallinta, TheSyDeKick

Preface

First, I'm deeply grateful to Professor Jussi Ryynänen and my instructor Marko Kosunen, whose expertise, guidance, and support were instrumental in shaping this thesis. Their mentorship challenged me to think critically, to dig deeper, and to strive for excellence. Thanks Aleksi Korsman, Otto Simola, and Verneri Hirvonen from the A-Core project for their help. The days in the ECD group will be unforgettable.

I also want to express my gratitude to Dejun Zhang and Haizhi Lin, who are my parents, for their encouragement and unwavering support throughout my study at Aalto. Additionally, my thanks is extended to my friends Peiyao Xu, Mengyuan Huangfu, Qi Qi, Shuchang Zhang, Shuhe Yu, Zixuan Ning, and Xin Zhong for their companionship during the challenging winter months. They believed in me even when I doubted myself, and their love and encouragement kept me going.

Thank you to all who have played a role in this journey. I am grateful for your support.

Otaniemi, 11.04.2023

Lin Zhang

Contents

Symbols and Abbreviations

Abbreviations

AI Artificial Intelligence

ALU Arithmetic Logic Unit

CISC Complex Instruction Set Computer

CMOS Complementary Metal Oxide Semiconductor

CPU Central Processing Unit

CSV Comma Separated Values

CU Control Unit

DRC Design Rule Checking

DSPF Detailed Standard Parasitic Format

DVFS Dynamic Voltage and Frequency Scaling

EDA Electronic Design Automation

EVCD Enhanced Value Change Dump

FPU Floating Point Unit

FSDB Fast Signal Database

GPIO General Purpose Input/Output

GPU Graphics Processing Unit

GUI Graphical User Interface

HDL Hardware Description Language

I/O Input/Output

IC Integrated Circuit

IoT Internet of Things

ISA Instruction Set Architecture

JTAG Joint Test Action Group

LED Light-emitting Diode

LVS Layout versus Schematic

$MOSFET$ Metal Oxide Semiconductor Field Effect Transistor

$NMOS$ N-channel Metalo Oxide Semiconductor

$PMOS$ P-channel Metal Oxide Semiconductor

PMU Power Management Unit

RAM Random Access Memory

$RISC$ Reduced Instruction Set Computer

$RSPF$ Reduced Standard Parasitic Format

RTL Register Transfer Level

SoC Systems-on-a-Chip

$SPEF$ Standard Parasitic Exchange Format

SPF Standard Parasitic Format

UDP User Datagram Protocols

VCD Value Change Dump

Symbols

A Total area of the transistors

C_L Capacitance of the load

E_C Energy stored on the capacitor

E_{VDD} Energy taken from the supply

$f_{0\to1}$ Frequency of energy-consuming transitions

i Current

I_{DD} DC current from supply

I_{leak} Leakage current

I_{off} Off-state leakage current

I_{sub} Substrate leakage current

p Power

$P_{dynamic}$ Dynamic power

P_{dyn} Dynamic power

P_{static}	Static power
P_{total}	Total power
t	Time
v	Voltage
V_{DD}	Supply voltage
V_{in}	Input voltage
V_{out}	Output voltage
w	Energy

1 Introduction

Over the past two decades, microprocessor technology has made significant advancements, resulting in a performance improvement of three orders in magnitude [1]. However, the energy efficiency of microprocessors has emerged as a crucial limiting factor that extends beyond the number of processors [2, 3]. Modern systems-on-chips experience thermal limitations and require enhanced energy efficiency to improve performance while maintaining chip longevity [4]. To address these constraints, effective power management systems are necessary to enhance energy efficiency while avoiding performance degradation. Such systems should adapt to the demands and limitations of specific workloads to optimize power consumption without affecting performance.

Achieving power management for modern microprocessor cores involves a complex closed-loop software-hardware optimization challenge aimed at regulating power supply and clock resources in accordance with computational requirements [5, 6]. Latest research of RISC-V processors has been demonstrated by research groups involved in RISC-V implementations [7–9]. To enable the advancement of power management systems for the A-Core RISC-V microprocessor core developed in Aalto University, the development of a dynamic power profile model is essential, which is a software, voltage and clock-frequency dependent time-varying power consumption profiling model. Additionally, the model has the potential to evaluate the power consumption and efficiency of other microprocessor cores, providing a comprehensive assessment of their dynamic power consumption. The structure of this thesis is as follows:

Chapter 2 presents the background information on the CPU (Central Processing Unit), ISA (Instruction Set Architecture), RISC-V, and A-Core processor implementation. The function of the CPU and how ISA impacts microprocessor design are explained. The chapter then delves into the details of the RISC-V architecture, which offers the benefits of open-source design and customizable microprocessors. The implementation of A-Core processor is also discussed, which is based on the RISC-V architecture.

Chapter 3 explores power consumption and management techniques. It examines power consumption in CMOS technology, including static and dynamic power consumption. Related literature review of power management techniques and power management systems are reviewed. The chapter also covers parasitic information and waveform information, which are used to analyze the power consumption and develop the profile model of digital circuits. The power profiling algorithms in this thesis utilizes various information extracted from the core during its operation, such as signal change information from the VCD file and capacitance information of nets from the SPEF file. These data will be employed to compute the power consumption, and the time-based power consumption curve will be generated for power profiling.

Chapter 4 provides an overview of the tools and methodology used in this project. This chapter introduces the workflow and the structure of the power profiling model, also the design methodology employed in this project is comprehensively presented. The power profiling algorithms developed in this thesis leverages the capabilities of

TheSyDeKick and Python programming language. A user-friendly TheSyDeKick interface for controlling and running simulations is established, which simplifies the power profiling process. The interface also enables the use of Python-based simplified power profiles for hardware and power management system development. Additionally, the chapter explores the ModelSim, IC design flow and the file generation which can be used in the profiling model.

Chapter 5 focuses on the design verification results, which is crucial to ensure that the design meets the specifications correctly. To assess the feasibility of the power profiling model, different test targets are evaluated and simulated to assess their efficiency, which include the trap test and the blinky test. Furthermore, the model's correctness and generality will be evaluated. The power profiling is carried out on the A-Core RISC-V microprocessor, and the results of different simulations are presented and summarized.

The objective of the work is using the post-layout data from the synthesized A-Core microprocessor to create a software, voltage and clock-frequency dependent time-varying power consumption profiles. During the work, the theory of the designed model is thoroughly studied and applied to the practical realm. The operation of the developed model is verified with simulations to show their good performance. Overall, the thesis aims to contribute to the development of efficiency computing by proposing a methodology to create a power profiling model for the RISC-V microprocessor core that can optimize power consumption while maintaining performance.

2 Background

2.1 Central Processing Unit

The CPU is a vital constituent of a computer system, responsible for executing program instructions [10, 11]. The CPU carries out instructions involving basic arithmetic operations, logic operations, control functions, and input/output (I/O) operations. The general structure of CPU is depicted in Fig. 1. It is comprised of several key components, including the ALU (Arithmetic Logic Unit), which conducts arithmetic and logic operations; the memory unit encompassing processor that provide operands to the ALU and preserve the results of ALU operations; and the control unit (CU), which supervises instruction fetching, decoding, and execution by synchronizing the operations of the ALU and other components [10, 12].

Modern CPU design trends involve incorporating CPUs into IC (Integrated Circuit) microprocessors that can accommodate multiple CPUs on a single chip, leading to the development of multicore processors [13]. Additionally, each physical CPU, or processor core, can be configured to support multi-threading, allowing for the creation of additional virtual or logical CPUs [14, 15]. In addition to the CPU, an IC microprocessor may incorporate memory, GPU (Graphics Processing Unit), peripheral interfaces, and other components. This integration of multiple components into a single chip is known as a microcontroller or System-on-a-Chip (SoC). The amalgamation of components within a singular integrated circuit contributes to enhanced efficiency and a reduced physical footprint, making it a desirable solution for various computing applications [16].

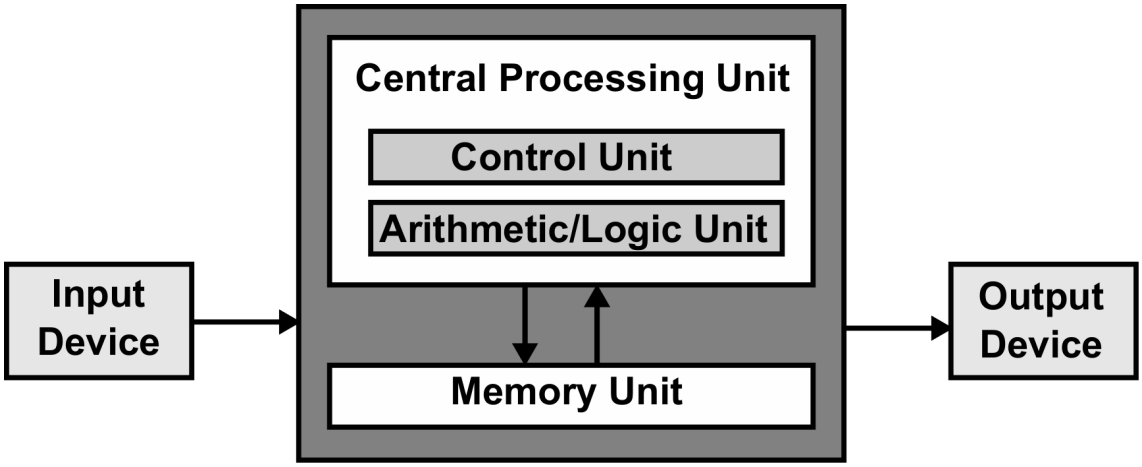

Fig. 1. General Structure of CPU [10]

Typically, the CPU executes an instruction by retrieving it from memory, employing the ALU to conduct the required operation, and subsequently storing the obtained result back into memory [17]. The control unit is a constituent of the CPU, responsible for guiding its operation. It delivers timing and control signals to the

computer's memory, ALU, and I/O devices. The management of most computer resources is undertaken by the control unit, which is responsible for coordinating the flow of data between the CPU and other peripheral devices. [17, 18].

A CPU's circuitry is embedded with a predetermined set of fundamental operations, termed an instruction set, which includes functions such as addition, subtraction, and comparison [18, 19]. Each instruction is identified by a unique bit combination, denoted as the machine language opcode. When processing an instruction, the CPU deciphers the opcode into control signals, which subsequently govern the CPU's behavior. The actual arithmetic or logic operation for each instruction is executed by the ALU, which carries out integer arithmetic and logic operations (bitwise).

The ALU's inputs comprise operands for operation, status data from prior operations, and a code from the CU indicating the operation to perform. In addition to integer math and logic operations, the CPU's instruction set may include various machine instructions, such as loading data, storing it, and performing mathematical operations on floating point numbers, which are executed by the FPU (Floating Point Unit) [19]. Based on the instruction, operands may be obtained from internal CPU registers, external memory, or produced by the ALU. Upon the stabilization of all input signals and their passage through the ALU circuitry, the resulting outcome of the concluded operation is manifested at the ALU's outputs [18, 20].

2.2 Instruction Set Architecture

The ISA serves as an abstract model of a computer system that outlines how the software manages the CPU, acting as the interface between the software and hardware, outlining the CPU's capabilities and how it executes tasks. The ISA is an essential aspect that distinguishes CPUs. Different processor hardware implementations are achievable with ISA, resulting in various performance levels [21].

CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer) are two primary categories of ISA. CISC contains many unique instructions, including specialized ones that are rarely used. As a result, the number of instructions in CISC is extensive, which is more complex. RISC, on the other hand, only includes the most commonly used instructions, with infrequently used operations executed through multiple commonly used instructions, resulting in a compact instruction set. Fig. 2 outlines the primary differences between CISC and RISC [22, 23].

Initially, CISC was preferred over RISC as it offered more functionality with fewer instructions. However, as the CISC instruction set developed and more specialized instructions were added, it became evident that 80% of these instructions were rarely used, leading to increased CPU design complexity, increased time cost, and hardware overhead. Consequently, since the advent of RISC, modern instruction set architectures have gradually shifted towards RISC architecture [22].

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	More registers
More addressing modes	Fewer addressing modes
Extensive use of micro-programming	Complexity in compiler
Instructions vary in cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

Fig. 2. CISC and RISC

2.3 RISC-V

The CPU architecture was mainly controlled by Intel (x86 architecture) and ARM (ARM architecture) for a long time [24, 25]. Besides the x86 and ARM architectures, the RISC-V ISA has gained significant popularity in recent years in both academia and industry [26, 27]. RISC-V is built on the concept of reduced instruction set computers and originated from research at Berkeley in 2010 [28].

The complexity of the mature x86 and ARM architectures, along with expensive patents and architecture licensing issues, led the Berkeley developers to invent a free, simple, and open-source ISA, which is RISC-V. One of the key advantages of RISC-V is its open-source nature, enabling collaboration and innovation within the industry [27, 28]. This has resulted in a thriving ecosystem of hardware and software developers actively contributing to the advancement of the ISA.

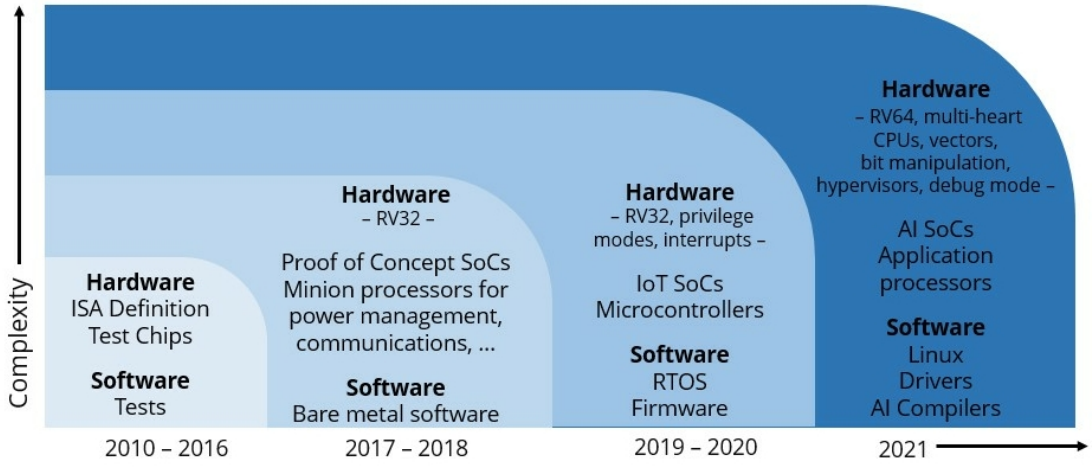

Fig. 3. Industry Innovation on RISC-V

As shown in Fig. 3, the versatility of the RISC-V ISA has led to a surge in the development of RISC-V based processors for various applications, such as microcontrollers, IoT SoCs, and AI SoCs [29, 30]. Companies can optimize RISC-V based processors for specific use cases and applications, improving performance and energy efficiency [31]. Furthermore, the open-source nature of RISC-V has facilitated the development of new hardware accelerators and custom instructions, enhancing the performance and efficiency of RISC-V based processors [32].

The encoding of the RISC-V instruction set is highly consistent, arranging the indices of the general-purpose register necessitated by the instruction at a predetermined location within the instruction code (Fig. 4). This organization enables the instruction decoder in seamlessly decoding the index of the register and subsequently accessing the general register file. Consequently, the decoder can rapidly decode the register index and retrieve the contents of the general register file [33, 34].

In a typical RISC-V instruction, the first few bits might indicate the operation to be performed, while a specific position within the instruction code might indicate the index of the register that should be used as the source operand. Another specific position might indicate the index of the register that should be used as the destination operand. By following this consistent structure, RISC-V processors can more easily decode and execute instructions quickly and efficiently [33].

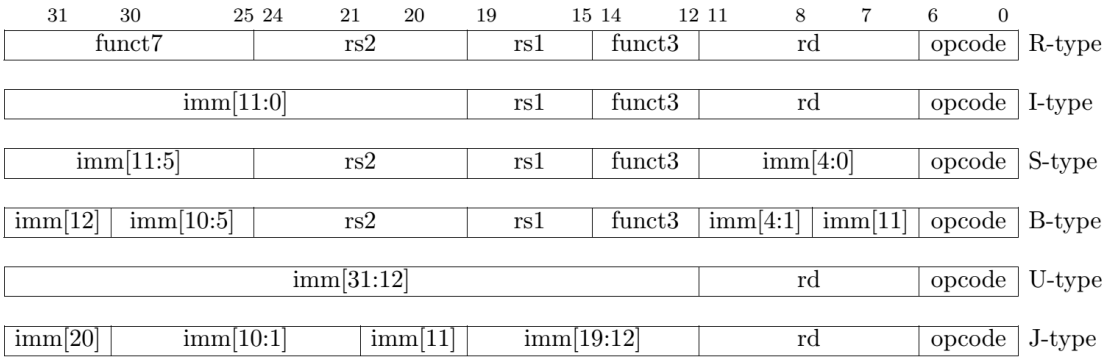

Fig. 4. Instruction Set Encoding of RISC-V [33]

In the RISC-V architecture, each instruction is encoded as a fixed-length binary code, typically 16 or 32 bits in length, depending on whether the instruction is part of the compressed or uncompressed instruction set. Like other RISC architectures, RISC-V employs "load" and "write" instructions for memory access, while other standard instructions are unable to access memory [35]. The architecture supports 1-byte, half-word, and single-word read/write operations, with the basic unit of memory access being 1 byte.

The RISC-V architecture is modular, with each module represented by an English letter [33]. The fundamental subset of integer instructions is denoted by the alphabetic character "I", constitutes the most essential and obligatory part of the instruction set. Utilizing this subset, a comprehensive software compiler can be implemented (Fig. 5). Additional instruction subsets, such as M/A/F/D/C, function as optional modules (Fig. 6). A specific combination of the modules "IMAFD" represented by the English letter "G", wherein RV32I[M][A][F][D] corresponds to RV32G.

To enhance code density, RISC-V also offers a "compressed" subset of instructions, symbolized by the alphabetic character "C" (Fig. 6). Compressed instructions have a length of 16 bits, while normal uncompressed instructions are 32 bits long. Additionally, the RISC-V encompasses an "embedded" architecture, denoted by "E" (Fig. 5). This architecture is predominantly employed in deeply embedded systems that prioritize minimal area and power consumption, necessitating support for only 16 general purpose integer registers, as opposed to 32 in non-embedded standard architectures.

Basic instruction set	Number of instructions	Description
RV32I	47	32-bit address space and integer instructions, supporting 32 general-purpose integer registers
RV32E	47	Subset of RV32I, supports only 16 general purpose integer registers
RV64I	59	64-bit address space with integer instructions and some 32-bit integer instructions
RV128I	71	128-bit address space with integer instructions and some 64-bit and 32-bit instructions

Fig. 5. Basic Instruction Sets of RISC-V

Extended instruction set	Number of instructions	Description
M	8	Integer Multiplication and Division Instructions
A	11	Memory Atomic operation instructions and Load-Reserved/Store-Conditional instructions
F	26	Single precision (32-bit) floating point instructions
D	26	Double-precision (64-bit) floating-point instructions, must support F-extended instructions
C	46	Compressed instructions, the instruction length is 16 bits

Fig. 6. Optional Modules of RISC-V

2.4 A-Core Processor Implementation

A-Core is a modular and easily expandable RISC-V core, which can be used as a controller in the mixed-mode System-on-Chip designs, implemented by ECD group from the Department of ELE at Aalto University [36]. A-Core aims to address several areas of development, including the RV32I core, a set of standard and custom extensions, programming interface, hardware verification with RTL (Register-transfer Level) simulations, and a firmware development environment that includes assembly and C programming. By focusing on these areas, A-Core developers created a flexible and robust RISC-V core that can support a wide range of applications and use cases. Additionally, the use of open-source tools and accessible programming languages allows for greater collaboration and transparency in the development process, which led to more innovation and progress.

The development of the A-Core RISC-V core involved building an open source implementation from scratch with accessible tools, which required the Chisel hardware description language. Chisel is incorporated within the Scala programming language, elevating the abstraction level for circuit design and facilitating a more intuitive and efficient design process while also supporting rich parameterization [37]. The compiler generated Verilog description that can be used to synthesize the circuit. Chisel's ability to operate at a higher abstraction level diminishes the time and effort necessary for circuit design and simultaneously enhances the quality of the produced circuits [37].

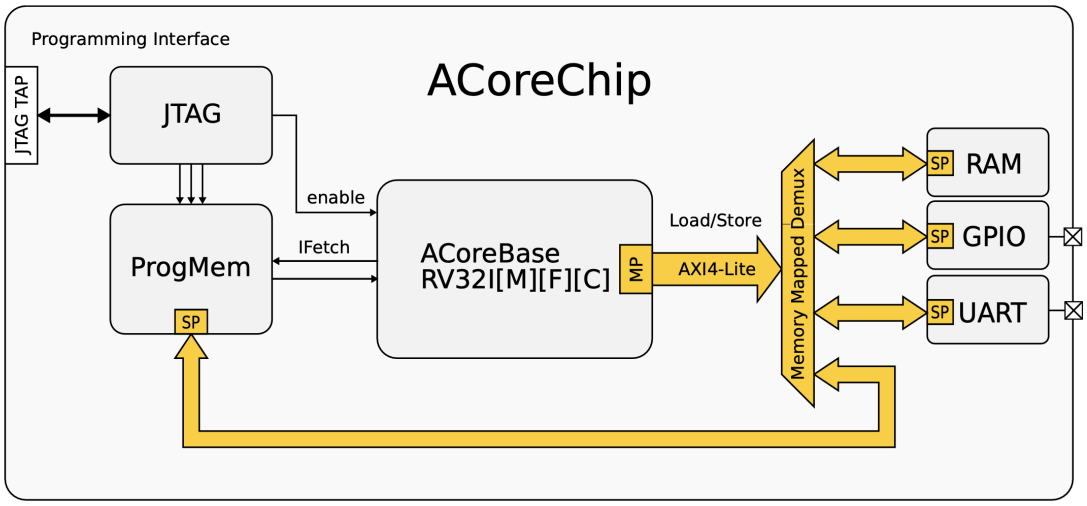

Fig. 7. Illustration of ACoreChip [38]

As Fig. 7 shows, ACoreChip is a SoC that includes a RISC-V processor core (ACoreBase), memory and peripherals. Memory-mapped peripherals encompass RAM, ProgMem (read-only access to instruction memory), GPIO (General Purpose Input/Output), and custom accelerator IP. ACoreBase communicates with memory

through an AXI4-Lite data bus, which provides a low-latency, high-throughput interface for data memory operations. In addition, ACoreChip has a separate instruction fetch bus to improve instruction throughput and reduce the impact of memory access latency. JTAG (Joint Test Action Group) programming interface is used to program and debug the processor core, allowing testing and refining.

ACoreBase is the RISC-V CPU core utilized in ACoreChip, which supports a parameterized RV32I[M][F][C] extension configurations [33]. It enables multi-cycle instruction fetch and execution, resulting in efficient performance. The RV32I base as a default includes instructions for basic integer arithmetic and logic, program flow modification, and memory access (load/store). Additionally, the "M" extension specifies instructions for the multiplication and division operations of integers. The "F" extension is also supported, which allows for single-precision floating-point operations. To manage the control and status registers required for the "F" extension, the "Zicsr" extension is required, which can be utilized for implementing timers and counters. The "C" extension permits the intermixing of 16-bit instructions with 32-bit instructions freely.

A-Core is a 7-stage pipelined, which allows for efficient instruction fetch, decode, and execution, shown in Fig. 8. Because A-Core is an individual core, pipelining becomes the first major push to improve single-core performance. Pipelining is found in virtually all modern processors. The data path was divided into separate stages, and registers were added between the stages. Additionally, the core supports a machine mode execution environment, and the parameterized RV32I[M][F][C][Zicsr] configuration enables the core to support a wide range of applications. A-Core is also designed to handle most synchronous exceptions, enabling it to recover from errors and continue processing without significant disruption.

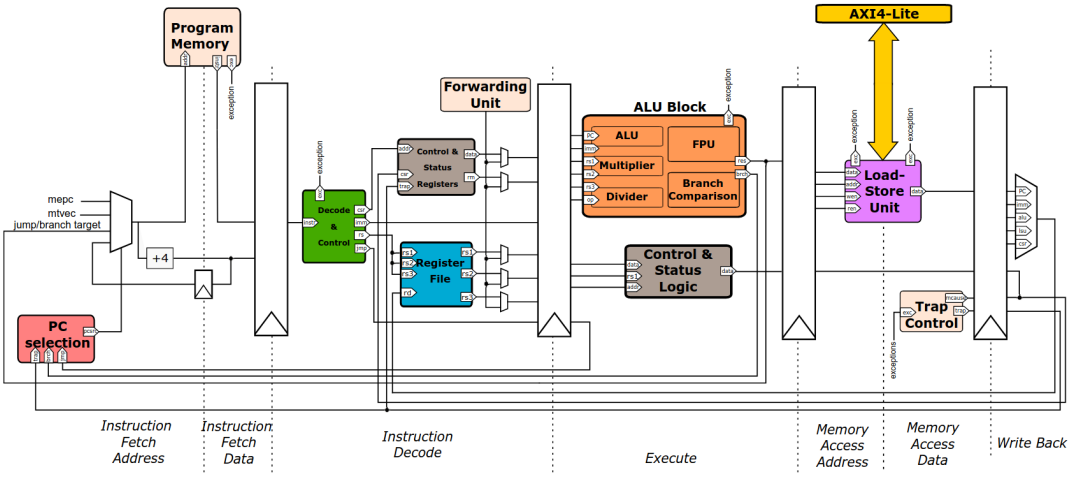

Fig. 8. A-Core Pipeline [38]

3 Power Consumption and Management

Power is characterized as the rate of energy expenditure or absorption over time and is quantified in watts (W). It can be expressed as the product of energy w (joules) and time t (second), as shown in (1).

$$p = \frac{dw}{dt} = vi \tag{1}$$

In (1), the term p denotes instantaneous power, which is a time-dependent variable. The power consumed or provided by an element is determined through the multiplication of the voltage v across it and the current i traversing it. A positive (+) sign signifies that the element is absorbing power, while a negative (-) sign indicates that the element is supplying power.

Power is an essential concept in electronics, representing the quantity of energy dissipated per unit of time in a device. It is also used to quantify the rate at which electrical energy is transferred or transformed. Electronic devices and systems rely on power to operate, which is usually supplied by batteries or power adapters. These power sources convert electrical energy from a power source into the required voltage and current levels for the device. Furthermore, power is a critical factor in determining the performance of electronic devices and systems. Generally, devices and systems with a higher power rating can perform more work within a given period than those with a lower power rating.

The total power dissipated within a device comprises two components: static power and dynamic power, as depicted in (2). Static power is dissipated when the device is in a steady state, while dynamic power is consumed during device switching.

$$P_{total} = P_{static} + P_{dynamic} \tag{2}$$

3.1 Power Consumption in CMOS Technology

In CMOS technology, power consumption is an important aspect to consider when designing digital circuits. The fundamental building block of a CMOS system is the CMOS inverter, which consists of a complementary pair of MOSFET transistors. The inverter acts as a signal negator and is used to implement logical functions. Fig. 9 shows the inverter in transistor level.

As (2) reveals that a CMOS inverter has two primary sources of power consumption: static power and dynamic power. Static power, results from the short circuit current which flows when both complementary transistors are simultaneously activated during switching. This causes a direct current flow originating between the supply and the ground. Dynamic power, conversely, is expended when the inverter changes its logical state, leading to the charging or discharging of the load capacitance [39–41].

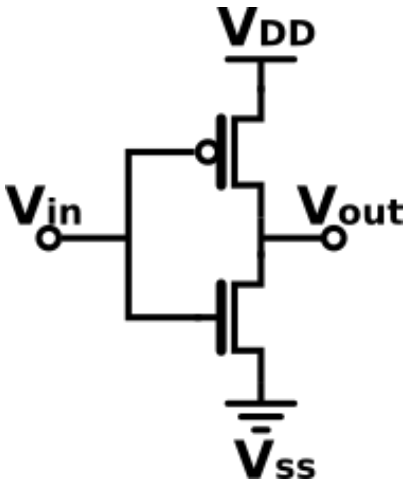

Fig. 9. CMOS Inverter in Transistor Level [39]

The transient characteristics of a CMOS inverter during switching can be observed in Fig. 10, which illustrates the voltage and current characteristics. During the high-to-low transition, the V_{in} (input voltage) drops, activating the PMOS transistor and deactivating the NMOS transistor. Consequently, the V_{out} (output voltage) starts to decrease, but initially, it decreases slowly due to the capacitance of the connected load. As the V_{out} continues to decrease, the capacitance becomes discharged, and the output voltage decreases more rapidly until it reaches a stable low state. As for the I_{DD} (DC current from supply), it remains at a low level at the beginning. Then the I_{DD} increases rapidly and decrease rapidly until it reaches the stable low state [39].

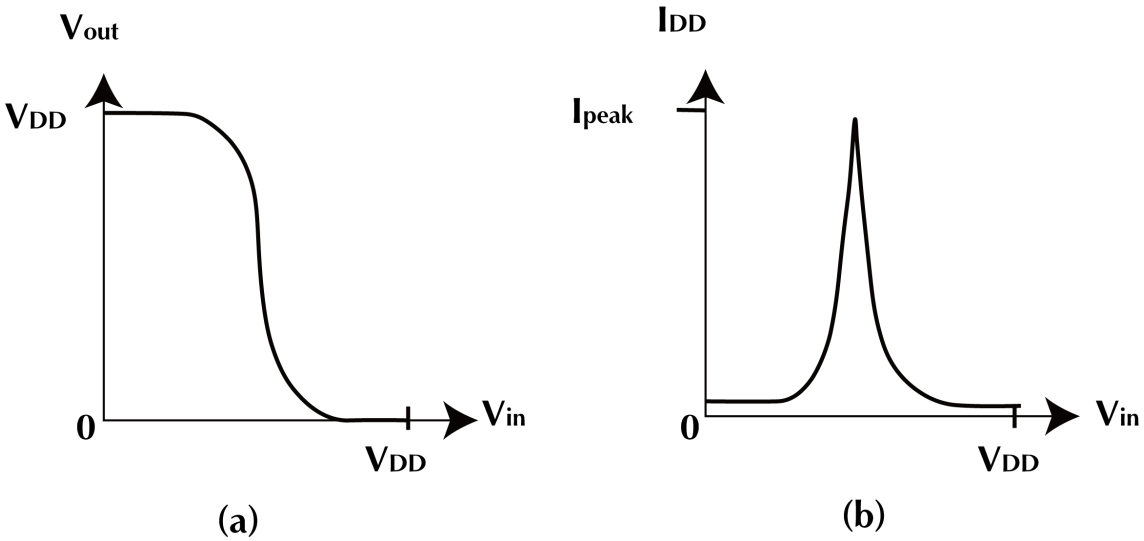

Fig. 10. Transient Characteristics of CMOS Inverter [39]

3.1.1 Static Power Consumption

Static power in CMOS circuits occurs when the circuit is in idle/standby state, even if it is not actively executing tasks. This power consumption mainly stems from leakage current flowing through the transistors and parasitic capacitances when the circuit is "OFF" [39, 41].

The static power consumed can be articulated as the multiplication of I_{leak} and the V_{DD} , as shown in (3). Here, I_{leak} represents the total leakage current flowing through the circuit's transistors, and V_{DD} denotes the supply voltage [39].

$$P_{static} = I_{leak} \cdot V_{DD} \tag{3}$$

Leakage current is affected by several factors, such as temperature, voltage, and process technology. To model the leakage current, the I_{off} (off-state leakage current of the transistors), I_{sub} (substrate leakage current), and A (total area of the transistors) must be considered. The leakage current can be modeled as [39]:

$$I_{leak} = I_{off} \cdot A + I_{sub} \tag{4}$$

Substituting the I_{leak} in (3) by (4), the static power consumption can be then modeled as:

$$P_{static} = (I_{off} \cdot A + I_{sub}) \cdot V_{DD} \tag{5}$$

This equation demonstrates that the static power consumption in the CMOS circuit exhibits proportionality to the leakage current, supply voltage, and transistor area. By minimizing leakage current and total transistor area, the static power consumption of modern electronics can be reduced. As the demand for energy-efficient devices continues to grow, minimizing static power consumption will remain a critical aspect of circuit design [39, 41].

3.1.2 Dynamic Power Consumption

Since static power consumption comprises a relatively minor segment of the total power consumed compared to dynamic power consumption, it is often disregarded. Dynamic power, serving as the principal origin of power dissipation, is consumed during switching due to the charging/discharging of the load capacitance [39, 42]. During the high-to-low transition, the load capacitor discharges, while during the low-to-high transition, the capacitor charges. Both transitions require energy from the supply voltage, and a portion of this energy is dissipated in the transistors [42].

Considering the low-to-high transition situation, assuming the input signal of the inverter have zero rise/fall times, which implies that the NMOS and PMOS devices are never simultaneously active [42]. Consequently, the equivalent circuit depicted in Fig. 9 is valid, as shown in Fig. 11.

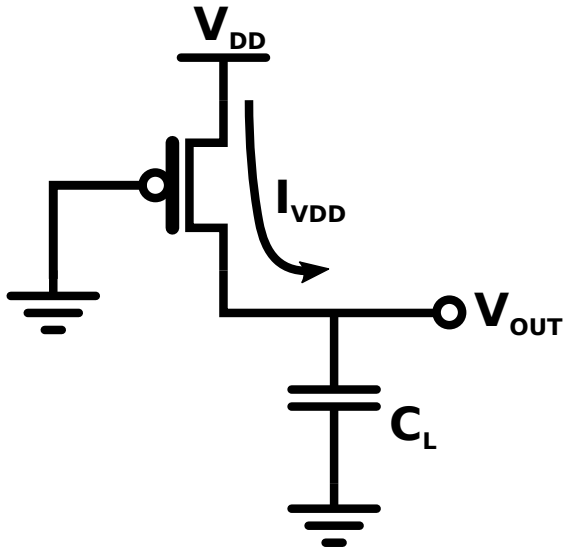

Fig. 11. Equivalent Circuit of Inverter (Low to high transition) [39]

By integrating the instantaneous power over the relevant period, the values of the E_{VDD} which is the energy drawn from the supply during the low-to-high transition, and the energy E_C retained on the capacitor at the transition end can be determined. Each switching cycle necessitates an amount of energy equal to $C_L V_{DD}^2$, where C_L signifies the load capacitance and V_{DD} represents the supply voltage. The frequency of device switching must also be considered when calculating the power consumption. If the device is switched on and off $f_{0\to1}$ times per second, the dynamic power consumption is:

$$P_{dyn} = C_L V_{DD}^2 f_{0\to1} \tag{6}$$

where $f_{0\to1}$ signifies the frequency of transitions.

3.2 Power Management

Electronic devices are known for their complex power requirements, which necessitates careful power supply control to ensure efficient and effective operation. This underscores the importance of power management in electronic devices. High power dissipation leads to increased energy consumption and temperatures. To keep energy consumption low and prevent overheating, minimizing power dissipation is essential for CPUs. Power management is also critical for optimizing the physical size of embedded systems. Effective management of the factors that impact power dissipation in CMOS circuits is necessary to minimize power consumption while still maintaining required functionality and performance, with the goal of maximizing energy efficiency [43].

In comparison to other components, CPUs possess more sophisticated capabilities for dynamically managing energy consumption. Prevalent power management techniques include DVFS (Dynamic Voltage and Frequency Scaling), which allows

for power reduction at the expense of performance, and power management through low-power states that reduce power at the cost of functionality [44, 45]. Power gating is another method that involves selectively shutting off power to parts of the circuit when not in use, resulting in substantial power savings, particularly in circuits with high leakage current.

Investigating the sources of power dissipation in modern CMOS circuits reveals that supply voltage significantly affects both dynamic and static power, while clock frequency also impacts dynamic power dissipation [46, 47]. As a result, reducing the voltage level and clock frequency of a CPU can decrease power consumption. The voltage and clock frequency of a processor are interrelated, with the capacitance charging/discharging time in a transistor determining the speed of logic state switching. Since charge and discharge time depends on the current, which is a function of voltage, the maximum frequency is contingent upon the applied voltage. Consequently, the clock frequency and supply voltage should be adjusted concurrently to accommodate the processor's requirements, maintaining the voltage as low as possible for any selected clock frequency to optimize energy efficiency. As processor workloads and usage fluctuate over time, regulating voltage and clock frequency in real-time based on the processor's needs can significantly lower energy consumption.

Power management is a critical aspect of designing and operating electronic devices. By carefully managing the power consumption of these devices, it is feasible to reduce energy consumption, minimize heat dissipation, and maintain the desired level of functionality and performance [43].

3.2.1 Power Management System

The power management system, as shown in Fig. 12, is a critical component of any modern microprocessor. It consists of several key components, including the PMU (Power Management Unit), supply unit, clock unit, performance metrics, and the CPU. The PMU controls the supply unit and the clock unit provided to the CPU based on the performance metrics [48–51].

In detail, the PMU monitors the power usage of various components of the microprocessor and adjusts the power supply accordingly. It can reduce the power supply to the processor core when the processor is idle, and increase it when the processor is under heavy load to ensure efficient power usage. The PMU also manages the sleep and wake states of the processor by reducing power supply to the processor core and other components in low-power sleep states, and quickly increasing it during wake-up transitions [51]. In addition to power management, the PMU can improve the overall performance of the processor by carefully controlling the power supply to the processor core and other components. By ensuring the processor operates at its maximum speed and efficiency, the PMU can help reduce overall power consumption. Therefore, the PMU plays a crucial role in the operation of a microprocessor by ensuring efficient power usage and effective performance, which ultimately enhances the performance of the electronic device [48, 51].

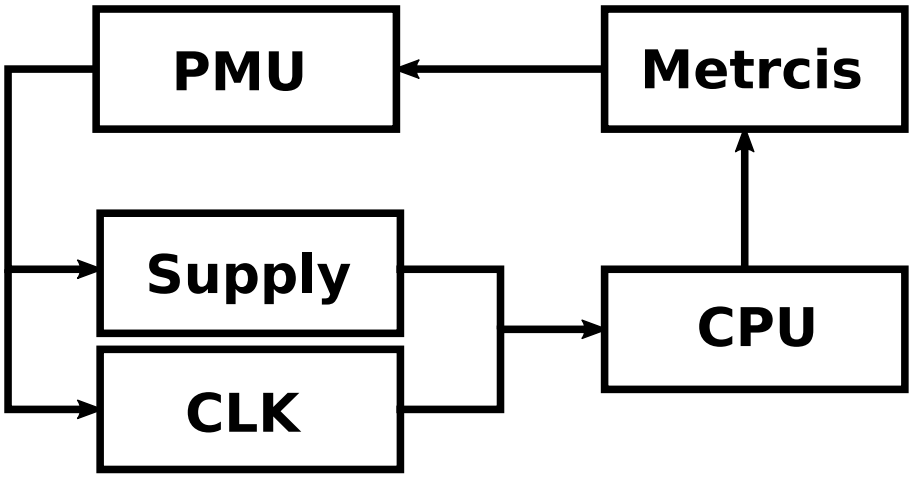

Fig. 12. Power Management System

The clock unit is a vital component responsible for generating clock signals that drive the processor's operation. It comprises a clock generator that produces clock signal and a clock distribution network that distributes the signal to the processor's components. The clock unit synchronizes the various components' activities, making it a critical element of the microprocessor's operation [52]. Without it, the processor would be unable to perform any tasks, as the components wouldn't be able to coordinate their activities. Additionally, the clock frequency determines the processor's speed, making the clock unit a key factor in its overall performance. The higher the clock rate, the faster the processor can perform its tasks. Adjusting the clock frequency can also reduce power consumption, enabling power management [53]. When the processor is idle, the clock frequency can be reduced to conserve power. When the processor is under heavy load, the clock frequency can be increased to ensure that the processor has enough power to perform its tasks [54].

The supply unit, another component of the power management system, provides power to the various components of the processor. It includes a power converter that transforms the input power from a power source, into the voltage and current levels required by the processor, as well as a power distribution network that supplies power to the various components. The supply unit ensures a stable and reliable source of power, which is necessary for the proper functioning of the processor [52]. It also optimizes performance and power efficiency by adjusting power supply according to the components' power needs, to ensure that the processor uses power only when needed. The supply unit provides the necessary power to the processor core and other components to ensure that they can operate at their maximum speed and efficiency. This can help to improve the overall performance of the processor, which is important for applications that require high-performance computing.

Performance metrics are crucial for evaluating the microprocessor's capabilities and comparing the performance of different microprocessors [54]. One common metric is the clock speed, which refers to the number of clock cycles per second that the processor can perform. Higher clock speed means a processor with higher clock speed that can perform tasks more quickly than a processor with a lower clock speed. The number of cores is another important metric, as most modern microprocessors have multiple independent processing units that can operate simultaneously. A higher core count enables the processor to perform more tasks simultaneously, improving overall performance. Cache size and memory bandwidth are also performance metrics for microprocessors [52, 54]. The cache is a high-speed memory utilized by the processor to store frequently accessed data, and a larger cache can improve performance. Memory bandwidth pertains to the volume of data that can be transferred between the CPU and the primary memory within a specified time frame. Higher memory bandwidth can improve processor performance.

3.3 Parasitic Information

3.3.1 Parasitic Formats

The SPF (Standard Parasitic Format) is a widely adopted standard to define the netlist parasitics, created by Cadence Design Systems. There are two forms of SPF: DSPF (Detailed-SPF) and RSPF (Reduced-SPF). Both DSPF and RSPF characterize parasitic information through an RC network. However, RSPF employs an RC "pi" model for each net. This model encompasses an equivalent "near" capacitance at the net driver, an equivalent "far" capacitance for the net, and an equivalent resistance that links the two capacitances. Conversely, DSPF models a detailed RC parasitic network for every net. As a result, DSPF provides greater accuracy than RSPF, but also DSPF files are generally larger than RSPF files for identical designs. Additionally, DSPF lacks a specification for coupling capacitors, and it shares structural similarities with a SPICE netlist.

The SPEF (Standard Parasitic Exchange Format) is an IEEE standard which defines netlist parasitics. While SPEF differs from SPF, they serve a similar purpose. and both of them account for resistance and capacitance parasitics. SPEF can represent parasitics in detailed or reduced (pi-model) forms, with the reduced form being more prevalent. Moreover, SPEF offers a syntax for modeling capacitance between different nets. In comparison to DSPF and RSPF, SPEF is more compact due to its use of name-to-integer mappings.

3.3.2 Standard Parasitic Exchange Format

The SPEF is widely employed for representing the parasitic data of wires within a chip [55]. It accurately captures the parasitic elements of wires (non-ideal), such as resistances and capacitances, but does not consider wire inductance. SPEF is a crucial format for exchanging parasitic information among various EDA (Electronic Design Automation) tools during the design process [55]. After the place and route stage, SPEF is extracted, enabling precise calculations for IR-drop analysis and other

analyses. The SPEF file contains resistance and capacitance parameters that depend on the block placement and the routing among the placed cells.

Typical SPEF files comprise four main sections, including header, name map, top level port, and the parasitic section. The header section provides information regarding design name, extraction tool, and units. To diminish the SPEF file size, the name map section permits the mapping of long names to abbreviated numbers preceded by "*" [55]. The top level port section lists the top level ports within the design, designating them as "I" (input), "O" (output), or "B" (bidirectional).

In the parasitic section, each extracted net encompasses a D_NET section. This section commences with a "D_NET" line that conveys the net name and its total capacitance. The CONN subsection delineates the connectivity of the nets by enumerating the pins associated with the net. The CAP subsection provides a detailed description of the capacitance data for the net, which encompasses the parasitic capacitance between a node and either the ground or another node within the net. Finally, the RES subsection presents the resistance network pertinent to the net, representing the parasitic resistances situated between pairs of nodes or points within the net.

```
1  *D_NET *105 1.94482
2
3  *CONN
4  *I JHV/U64885:E I *C 643.845 9827.11 *L 3.30000
5  *I JHV/U65821:Z O *C 641.216 9324.88 *D OR2N1P1
6  *I JHV/U56325:A I *C 690.356 9176.02 *L 5.50000
7
8  *CAP
9  1 JHV/U64885:E 0.936057
10 2 JHV/U56325:A  JHV/U10716:Z 0.622675
11 3 JHV/U65821:Z 0.386093
12
13 *RES
14 1 JHV/U64885:E  JHV/U65821:Z 10.7916
15 2 JHV/U64885:E  JHV/U56325:A 8.07710
16 3 JHV/U56325:A  JHV/U65821:Z 11.9156
17 *END
```

Listing 1. Example Parasitics Section of the SPEF File

3.4 Waveform Information

3.4.1 Signal Change Formats

There are various file formats available to save waveform data used in EDA and digital verification. Commonly used formats include VCD (Value Change Dump), EVCD (Enhanced Value Change Dump), and FSDB (Fast Signal Database) . Each of these formats has specific features and advantages, which make them suitable for different types of electronic design and verification tools [56, 57].

VCD, a text file format, represents the simulation waveforms generated during digital logic simulation [58]. It contains information about signal changes and their timestamps. EVCD is an extension of the VCD format that includes additional information such as signal values and signal attributes [58]. FSDB, a binary file format, is used to store simulation data, including waveform information and debug information [57]. FSDB files are preferred over VCD files in the semiconductor industry because of their faster speed and compactness. The use of these formats has become critical for electronic designers and verification engineers, as they enable accurate and efficient analysis and simulation of digital circuits. The choice of format depends on the specific requirements of the simulation and analysis tools being used.

3.4.2 Value Change Dump

VCD is a file format utilized to store and represent digital simulation waveforms in the field of EDA [58]. A VCD file comprises data about changes in the values of selected variables in the design, as recorded by value change dump system tasks. There are two types of VCD files: the four state type and the extended type. The extended type represents variable changes in all states and includes strength information. The four state type, on the other hand, represents variable changes in "0", "1", "x", and "z", without strength information [58]. In this project, the four state type value change dump file was employed, the format will be discussed in detail.

The standard four state VCD file typically consists of four primary sections, each with its own purpose: header, variable definition, dumpvars, and value change section. The header section contains meta-information about the VCD file, such as the date, version, and timescale. It starts with a keyword, and ends with "$end" [58]. The timescale defines the time unit used for the simulation data.

```
$date
    Tue Oct 18 14:41:32 2022
$end
$version
    QuestaSim Version 10.6_1
$end
$timescale
    1fs
$end
```

Listing 2. Example Header Section

The variable definition section contains scope information and enumerates signals instantiated. The keyword "$scope" indicates the hierarchical scope or module in the design where the signals are defined. The scope is specified as a module, task, or function, followed by the name of the scope. The keyword "$var" defines a signal/variable within the current scope. Each signal/variable is assigned a concise identifier used in the following value change section [58]. Multiple variables can share the same identifier if the simulator determines that they will always possess the same value. The following example variable definitions section defines three signals (clk, reset, and data) within the "top" module scope.

```
$scope module top $end
$var wire 1 ! clk $end
$var wire 1 " reset $end
$var wire 8 # data $end
$upscope $end
```

Listing 3. Example Variable Definition Section

Staring with the keyword $dumpvars, the dumpvars section lists the initial values of the variables that are to be dumped. In the following example dumpvars section, all the variables defined in the variable definition sections are listed with initial value.

```
$dumpvars
0!
0"
b00000000 #
$end
```

Listing 4. Example Dumpvars Change Section

The value change section enumerates time-ordered changes in signal values. The data is written in a compact form to reduce the file size. The "#" symbol followed by a time value (integer) specifies the simulation time at which the value changes occur. The value change is represented by the new value of the signal ("0", "1", "x", or "z") followed by the unique identifier of the signal. Multiple value changes can occur at the same simulation time and can be listed on the same line. The example value change section shows the changes in signal values at different time points (5fs, 10fs, and 15fs) in the simulation.

```
$dumpon
#5
0!
1"
x#
#10
1!
0"
#15
0!
1"
b00000001 #
$end
```

Listing 5. Example Value Change Section

4 Tools and methodology

4.1 Design Workflow

The power consumed by a system comprises both dynamic and static power, with the former being the dominant factor. Due to the relatively small size of static power, it is often ignored in power profiling. The general idea of the power profiling is based on the activities of key signals to evaluate the power consumption. Therefore, the information of the time-based signal activity and traced net capacitance is needed. To achieve this, detailed capacitance information of the net is saved in SPEF files extracted through the digital flow, and the switching activities are saved in VCD files simulated by running post-layout simulations through TheSyDeKick using post-layout Verilog.

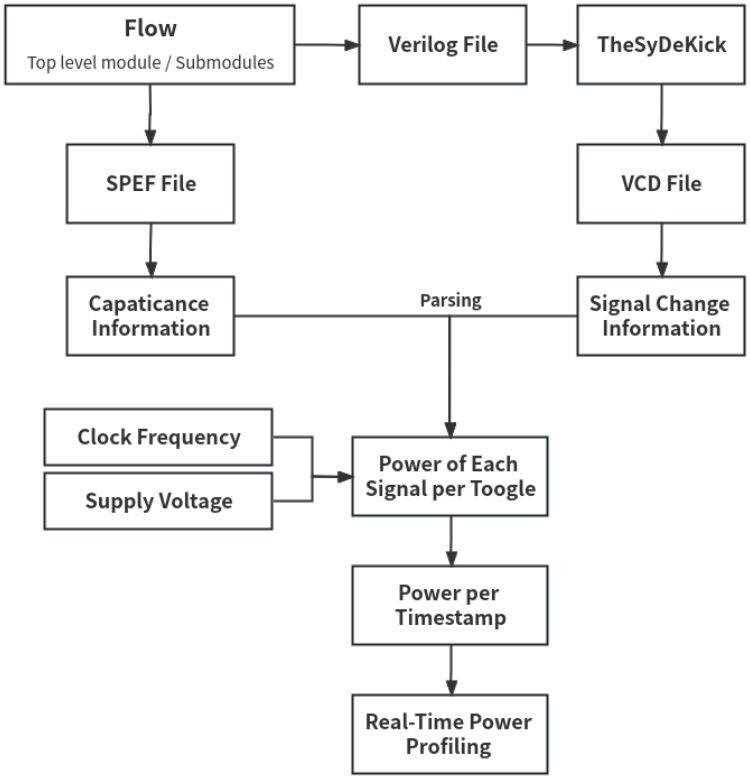

Fig. 13. Design Workflow

VCD and SPEF files are widely used file formats in the electronics design industry because of their compatibility across different tools and platforms [59]. VCD files are commonly used for digital simulations and contain a record of changes to the value of each signal in a design over time, making them ideal for post-simulation analysis and debugging [60]. This feature also makes them useful for tracking and analyzing power consumption in digital circuits, as changes in signal values often correspond to changes in power usage. SPEF files, on the other hand, are used

for parasitic extraction and timing analysis. They contain parasitic information, such as capacitance and resistance, which affect the behavior of signals in a design. SPEF files enable the extraction of parasitic information from a layout and can be used to generate accurate timing models. Together, VCD and SPEF files provide a comprehensive and accurate representation of digital circuits, allowing for precise power profiling and optimization [60]. By tracking changes in signal values and accurately modeling parasitic elements, power consumption can be addressed in the design process, leading to more efficient and effective electronic designs.

Fig. 13 illustrates the workflow of the power profiling process. The digital flow provides the SPEF file and the post-layout Verilog. The post-layout Verilog is sent to TheSyDeKick to extract the required value change information of the signals for a specific test, which is stored in a VCD file. The SPEF and VCD files are then parsed, together with the necessary information, the power consumption of each signal per toggle can be calculated. The necessary information, such as core voltage and clock frequency, is obtained from the report file from the flow. Finally, the power per timestamp is calculated and saved in CSV files, and the time-based power consumption curve can be plotted, which will be used for power profiling.

4.2 Power Profiling Model Structure

The power profiling model works as a TheSyDeKick entity, consisting of the "Simulation File", "Result Files", and "LoPoMan" folder. The "Simulation Files" contains the SPEF and VCD files used for simulation. The "Result Files" includes CSV files with time-based power information, TXT files with block names of this design, and YAML files detailing the relationship between net name and its capacitance, also signal name and its power consumption of each toggle. YAML file allows representing complex data structures in a human-readable format, which is suitable in this design. CSV files are used to plot the power profiling curve, the curve will be saved in EPS format. The "LoPoMan" contains the scripts of the power profiling model. As a TheSyDeKick entity, the model also employs "./configure && make" to simplify the user's power profiling process.

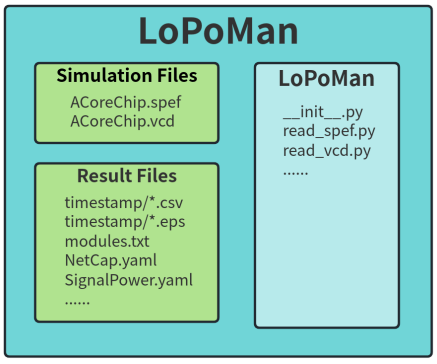

Fig. 14. Structure of the Power Profiling Model

The current power profiling model includes various features, including the generation of the basic structure of A-Core. Before performing actual power consumption analysis, the basic structure of A-Core is extracted for further module power analysis. The current general structure of A-Core is depicted in Fig. 15. The model also supports top-level module power consumption analysis and submodule power consumption analysis of the specified module under the top-level module. Additionally, it supports power consumption analysis of a specified time interval and analyzing the average power consumption.

The power profiling model in TheSyDeKick platform provides a comprehensive tool for analyzing power consumption in digital circuits. With its features, users can accurately profile power consumption and optimize their designs for better efficiency. The model simplifies the process for users, allowing for more efficient power analysis and optimization.

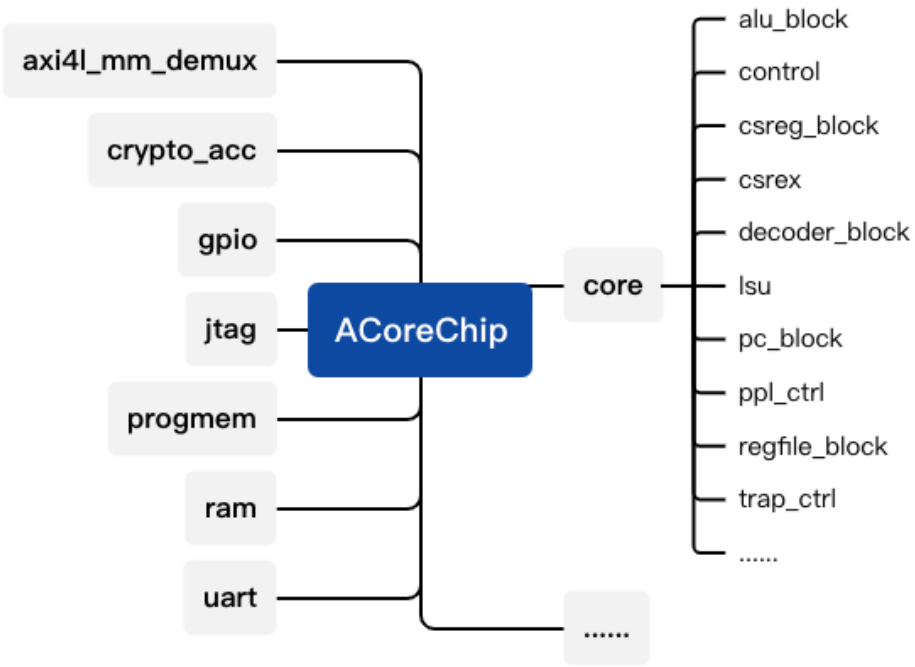

Fig. 15. Structure of A-Core

4.3 TheSyDeKick

TheSyDeKick is a multi-tool simulation and development environment designed to facilitate the creation of complex systems [38]. Its primary objective is to provide users a unified control environment that enables users to simulate, design, and measure various components of the system using a range of tools. This objective is realized through the employment of a unified control environment responsible for the managing, analyzing, and graphical representation of the consequent outcomes [61].

Python language was chooses to implement TheSyDeKick's control environment for its robust support for computing and signal processing, as well as its compatibility with interfaces to measurement equipment. Regardless of the language or tool used, all component descriptions are stored in the "Entities" directory. To ensure a standardized method for initializing git submodules, a script called "./init_submodules.sh" is utilized. Configuration is accomplished via the "./configure" script, which generates the Makefile, while execution is conducted using "make". The "./configure && make" structure is employed to eliminate the need for continually documenting configuration and execution procedures [38]. The main feature of TheSyDeKick pertains to the manner in which it interconnects various objects, or entities. By combining these features, TheSyDeKick enables users to seamlessly simulate, design, and measure intricate systems using a single control environment.

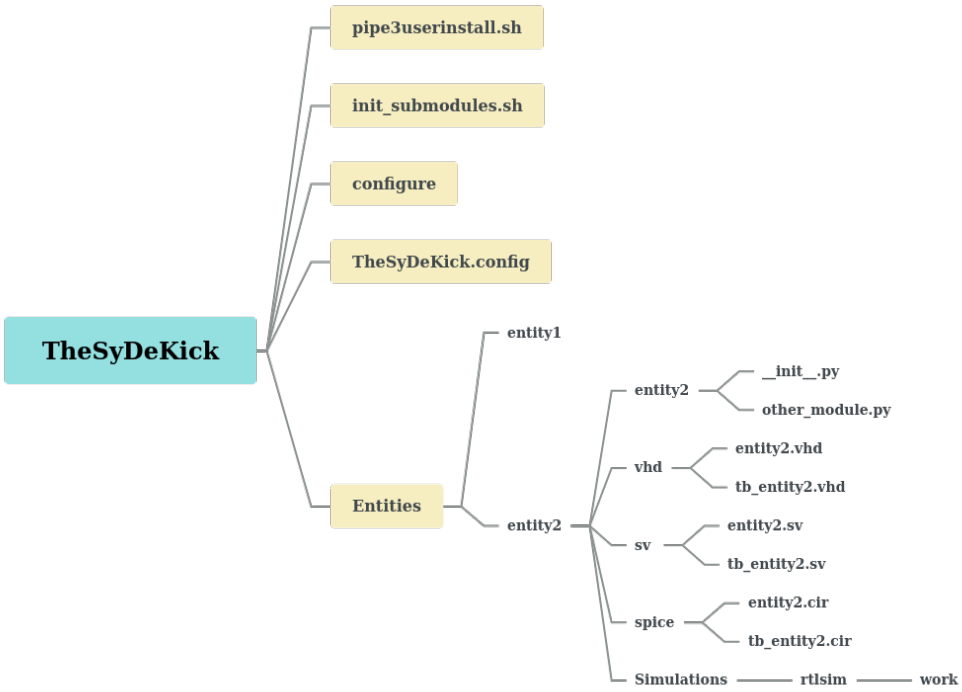

Fig. 16. Structure of theSDK

4.4 ModelSim

The ModelSim software supports the VCD file format for simulating circuit designs [62]. To simulate a design using ModelSim, several files need to be collected, including the design files with stimulus, working and resource libraries, and the configuration file automatically generated by the library mapping directive. Next, this design must be compiled using specific ModelSim commands, contingent on the language employed in its development [63]. Then, the "vsim" command is run on the top level module, and upon its successful loading, ModelSim progressively loads the instantiated modules and UDP (User Datagram Protocols) situated within the design hierarchy, facilitating the connections of ports and the resolution of hierarchical references.The initiation of the simulation is accomplished by setting the simulation time to 0 and employing the "run" command. Debugging can be performed using the ModelSim GUI and various commands, operations, and windows.

The transcript file generated can serve as a .do file, which functions as a rudimentary scripting language for ModelSim [62]. The .do files encompass a series of commands executable within the ModelSim command prompt, including those used during simulation. During simulation, waves can be added to observe the behavior of the design using the "add wave" command, which creates a wave in the simulation environment and assigns it to a specific input/output. Waves can also be assigned colors using the "-color" flag, such as "add wave -color yellow x" to create a yellow wave for the input x [63].

4.4.1 VCD File Generation

ModelSim could provide two types of VCD file generation: the extended VCD File and the four state VCD File. As mentioned before, the four state type was chosen in this project due to its widespread utilization and extensive support as a standard format and it did not require the additional information that the extended VCD File provides. Therefore, the four state VCD File was a more efficient and suitable choice for this project. The .do file can be modified to create a VCD file for signals under the desired module, such as the module ACoreChip.

```
1 #Specify VCD filename
2 vcd file ACoreChip.vcd
3
4 #Enable VCD to dump signals under a desired instance
5 vcd add -r sim/:tb_ACoreChip:ACoreChip:*
```

Listing 6. Commands to Generate the VCD File

4.4.2 VCD File Size Reduction

Typically, VCD files generated by ModelSim can be quite large. ModelSim provides the option to compress VCD files using the "gzip" algorithm. By specifying the [-compress] argument or specifying a .gz extension, the VCD will be stored in this compressed format [63]. Python library "gzip" allows to work with this compressed

files end with .gz.

```
vcd dumpports [-compress] [-direction]
              [-file <filename>] [-force_direction]
              [-in] [-out] [-inout]
```

Listing 7. Reducing the VCD File Size Using gzip Algorithm[63]

Users can also employ the "vcd on" and "vcd off" commands to regulate the VCD dumping process[63]. The "vcd on" command initiates VCD dumping to a designated file, simultaneously capturing the present values of all VCD variables. The "vcd on" command is executed automatically by default upon reaching the simulation time at which the "vcd add" command was carried out. In contrast, the "vcd off" command ceases the dumping to a specified file, consequently recording all the dumped variable values as "x". Additionally, the "when" command enables users to direct ModelSim to perform actions based on specified conditions, such as ceasing the simulation when a particular signal value is reached or at a specific simulation time. By utilizing these commands, users can obtain a VCD file for a desired interval. For instance, if a VCD file when the core is enabled is needed, i.e., when the signal "core_en" changes from 0 to 1, they can add the following commands to the .do file [63].

```
vcd file ACoreChip.vcd
vcd add -r sim/:tb_ACoreChip:ACoreChip:*
vcd off

when {sim/:tb_ACoreChip:ACoreChip:core:io_core_en == 1} {
    vcd on
}

run -all
```

Listing 8. Generating VCD File, when the core is enabled

4.5 IC Design Flow

The procedure of creating an integrated circuit is referred to as the IC design flow, encompassing a series of steps from establishing design specifications to manufacturing the final physical IC product. These stages are depicted in Fig. 17. Although the design process is often portrayed linearly, numerous iterations occur between adjacent stages, and occasionally between stages further apart.

The IC design process is initiated with a predetermined set of requirements, which serve as the benchmark for evaluating the initial design. If the design fail to fulfill the requirements, enhancements are necessitated. The transistor-level design flow originates from a set of design specifications that typically outline the anticipated functionality of the designed block, along with constraints on delay times, power dissipation, etc.

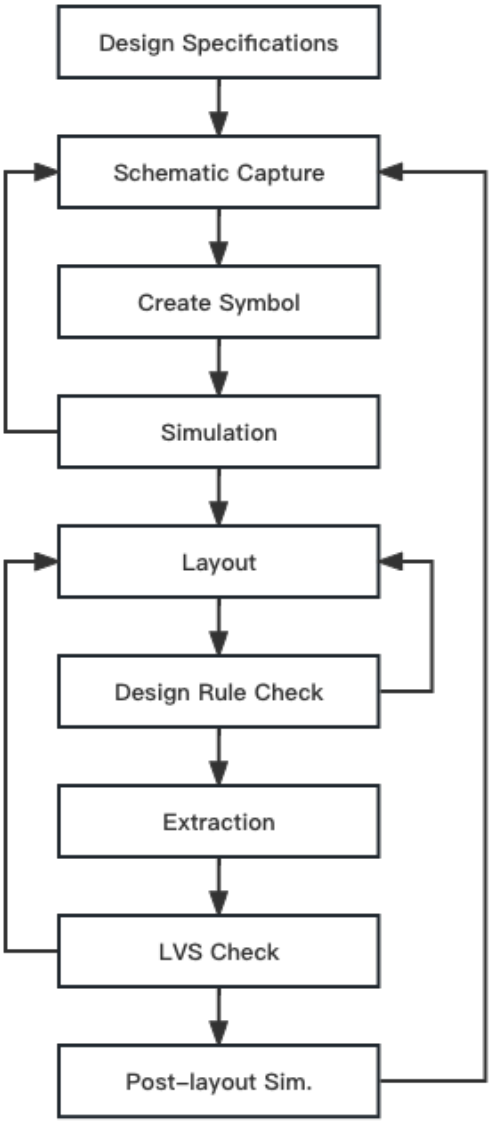

Fig. 17. Stages of IC Design Flow

A schematic editor functions as the traditional approach to capture a transistor/gate level design, affording an intuitive means of drawing, positioning, and interconnecting individual components that constitute this design. The resultant schematic is required to accurately represent the electrical properties of all elements, including their connections and links to the power supply and ground, and input/output interface pins, which are essential to generate the corresponding netlist. This netlist will be utilized in the subsequent design phases. Consequently, the formulation of a comprehensive circuit schematic constitutes the initial, critical step in the IC design flow.

For a circuit design encompassing smaller hierarchical components, it proves advantageous to recognize the modules early in the design process and allocate each module an appropriate symbol. This approach condenses the schematic representation of the overall system, generating symbols to represent the circuit is essential for subsequent simulation steps.

Upon completing the transistor-level representation of a circuit using a schematic editor, the circuit's electrical performance and functionality must be verified with a simulation tool. Detailed transistor-level simulation constitutes the first thorough validation of a circuit's operation and is essential to accomplish before proceeding to subsequent design optimization steps. Based on simulation results, designers typically modify device properties to enhance performance.

Developing a mask layout is a critical stage in the design flow of integrated circuits. Designers must execute DRC (Design Rule Checking) rectify all errors to finalize the mask layout. After completing the mask layout design, the circuit extraction process is conducted to produce a detailed netlist for simulation. The circuit extractor identifies transistors and their connections, in addition to the parasitics information existing between layers. The extracted netlist offers an precise estimation of device dimensions and parasitics. Subsequently, the extracted netlist file and parameters are employed for LVS (Layout versus Schematic) comparison and post-layout simulations.

The most effective method for analyzing the performance of the designentails conducting a post-layout simulation on the extracted circuit netlist. A comprehensive mask layout of the intended circuit must be processed and must successfully pass DRC and LVS checks without any violations. If post-layout simulation results are unsatisfactory, modifications to transistor dimensions is mandated to attain the desired circuit performance under realistic conditions. This procedure may encompass multiple design iterations until the post-layout simulation results align with the initial design specifications. However, the post-layout simulation outcome which is satisfactory does not assure a wholly triumphant product. The actual performance can only be determined through the examination of the fabricated chip prototype.

4.5.1 SPEF File Generation

Generating a SPEF file through digital flow involves a series of steps. Start by setting up the design in the digital flow environment, including the design files and any necessary libraries and constraints. Then the extraction process was on, commands "extrac_rc" will create the extracted netlist. This netlist will contain all of the information about the parasitic characteristics of the design. Command "write_parasitics" will then write the parasitic information into SPEF file. Following scripts can be used to generate the SPEF file in digital flow, showed in Code 9.

```
1 create_flow_step -name write_parasitics -owner flow -
    exclude_time_metric {
2   extract_rc
3   write_parasitics -rc_corner rcworst_CCworst -spef_file
     ACoreChip_worst.spef
4   write_parasitics -rc_corner rcbest_CCbest -spef_file
     AaltoProc_wio_rcbest_CCbest.spef}
```

Listing 9. Generating the SPEF File

As the SPEF file contains the resistance information that won't be utilized in the design, to reduce the file size and analyzing time, modifications can be done to the aforementioned scripts, by adding "-no_resistances" argument to generate the SPEF file without the resistance network for the net which is showed in Code 10.

```
1 create_flow_step -name write_parasitics -owner flow -
    exclude_time_metric {
2   extract_rc
3   write_parasitics -no_resistances -rc_corner cworst_CCworst
      -spef_file ACoreChip_best.spef
4   write_parasitics -no_resistances -rc_corner cbest_CCbest -
     spef_file ACoreChip_best.spef}
```

Listing 10. Generating the SPEF File without resistance information

5 Design verification

Different programs were executed on the A-Core processor to verify the power profile model. Milliwatt (mW) was selected as the unit of power, and millisecond (ms) was chosen as the time unit. The power results presented have a precision of 4 decimal places in general, but in some cases the power results were showed in 7 decimal to identify the differences. The time is reported with a precision of four decimal places. The selected level of precision has been chosen to ensure accurate reporting and facilitate the identification of differences. However, it should be noted that the precision should be adjusted based on the specific requirements of the situation.

5.1 Trap Test

The test program "trap" is designed as a configuration of a RV32I[M][Zicsr] C-language program. The source code for this test can be accessed in Appendix A. The test begins by defining functions required for the simulation, such as the recursive function for computing the factorial of an integer and the function of delay. Once the core is enabled, the test proceeds by computing the factorial of an integer and testing memory write/read operations. To ensure the correctness of the calculation and A-Core function, 4 integers were used for testing one by one to show the pattern.

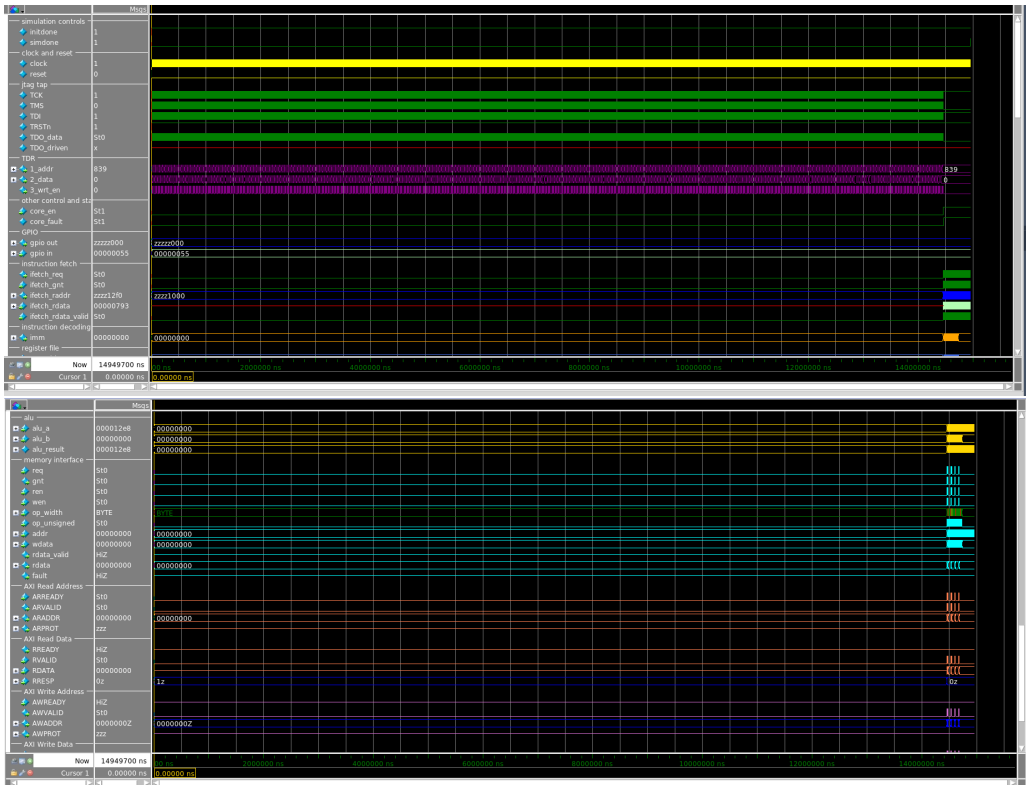

Fig. 18. ModelSim Results of the Trap Test

5.1.1 ModelSim Results

The results of the trap test during ModelSim simulation are presented in the Fig. 18. The simulation lasted for a total of 14.9497 ms, and the signal "simdone" indicated its completion. Prior to the simulation duration, the A-Core underwent preparation, while the JTAG and TDR remained active. At 14.4597 ms, the signal "core_en" was triggered, indicating the activation of the A-Core for calculation purposes. As a result, the instruction-related blocks, ALU, memory blocks, and AXI4-related blocks became active and executed the simulation.

5.1.2 Profile Results

Fig. 19 displays the power consumption behavior during the trap test. Prior to approximately 14.4500 ms, the power consumption remained at a low level because the core was inactive. The minor peak observed at the beginning can be attributed to the activation of all signals and their assignment to new values. Afterward, the power consumption increased and remained high for about 3 ms before suddenly decreasing and remaining at an exceedingly low level until the end of the simulation. The observed trend of power consumption is consistent with the ModelSim results. The average power consumed of trap tests was 0.0593 mW.

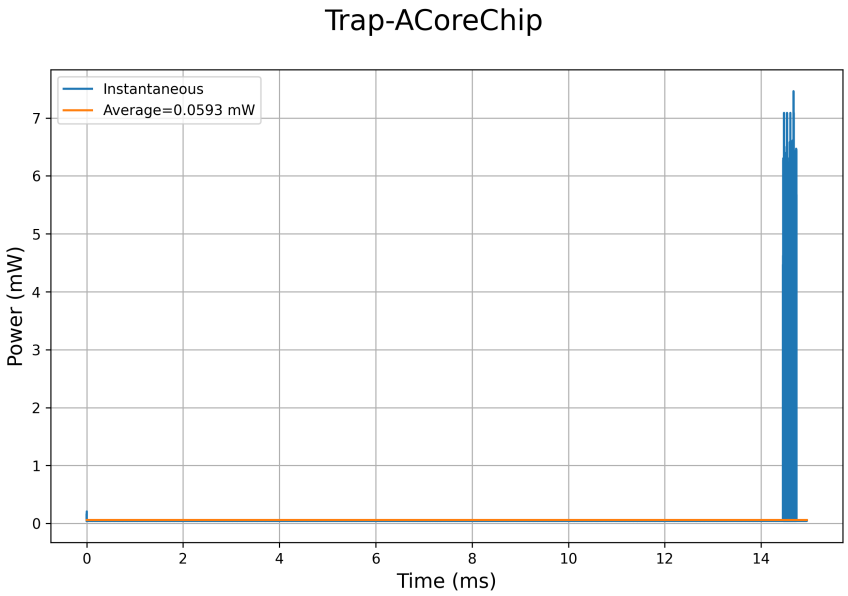

Fig. 19. Power Consumption of the Trap Test

For the purpose of attaining a more comprehensive comprehension of power consumption patterns, a zoom-in analysis was performed on the data starting from 14.4597 ms, when the core was enabled. From the Fig. 20, it can be found four similar peaks representing the four integers used for testing the chip's function with delay.

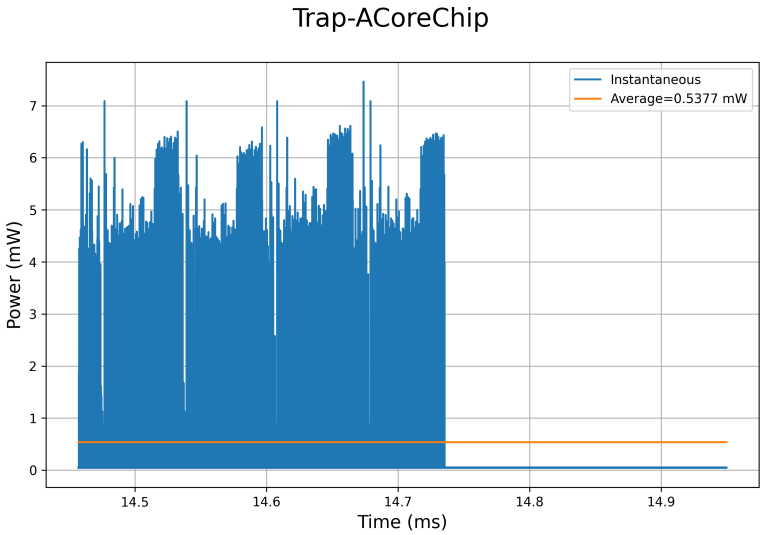

Fig. 20. Power Consumption of the Trap Test (Zoomed)

As shown in Fig. 15, the top-level module ACoreChip comprises several submodules, including axi4l_mm_demux, core, cypto_acc, gpio, jtag, progmem, ram, and uart. Fig. 21 presents the power consumption of the module core. As the central part of A-Core, the module core consumes the majority of power and exhibits the same changing pattern as ACoreChip. After the core was enabled, the average power consumption was 0.4978 mW, which accounts for 92.5795% of the power consumed by ACoreChip.

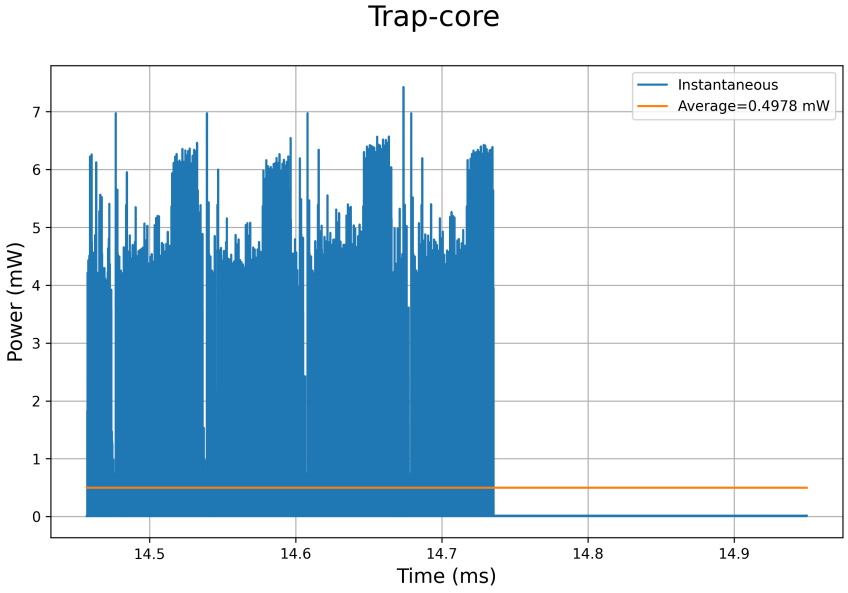

Fig. 21. Power Consumption of the Trap Test: core

Taking the submodule axi4l_mm_demux as an example, which represents the AXI4-Lite data bus that ACoreBase communicates with memory through. The power consumption of axi4l_mm_demux is showed in Fig. 23. As the ModelSim results (Fig. 22), the power consumption is corresponding to the signal change.

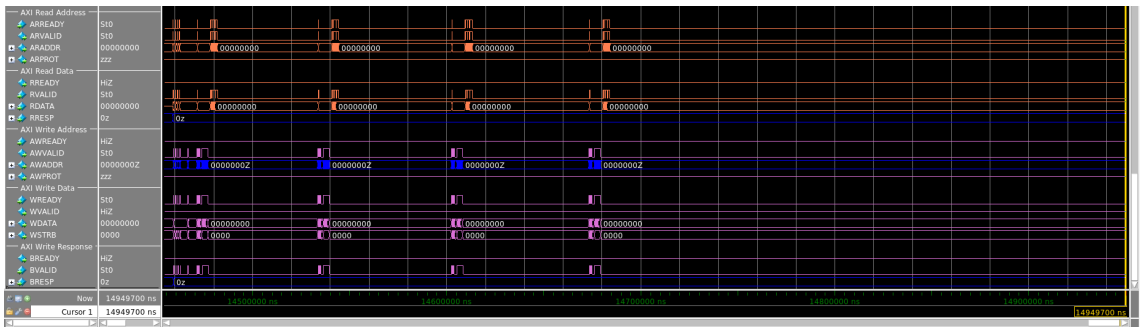

Fig. 22. ModelSim Results of the Trap Test: AXI4

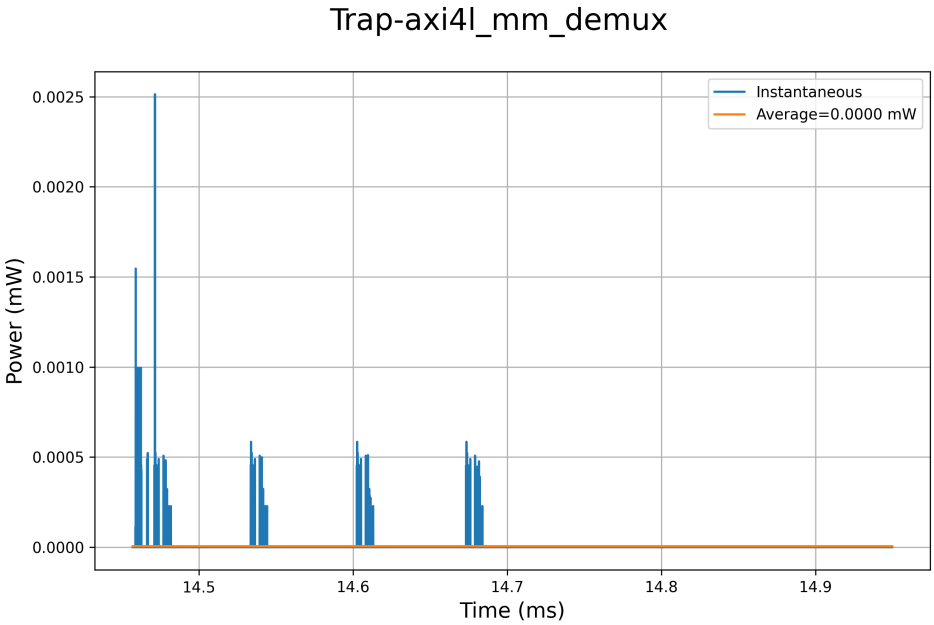

Fig. 23. Power Consumption of the Trap Test: axi4l_mm_demux

In Fig. 24, the power consumption of the remaining submodules under ACoreChip are presented. The jtag submodule is not active during the simulation. The submodules ram, gpio, and uart exhibit peaks in power consumption when the calculation of factorial is running and when memory read/write operations are being tested. On the other hand, the progmem submodule has peaks at the same position but stays active at a high level of 0.0095 mW during the calculation and at a low level of 0.0040 mW when the core exists in a dormant state.

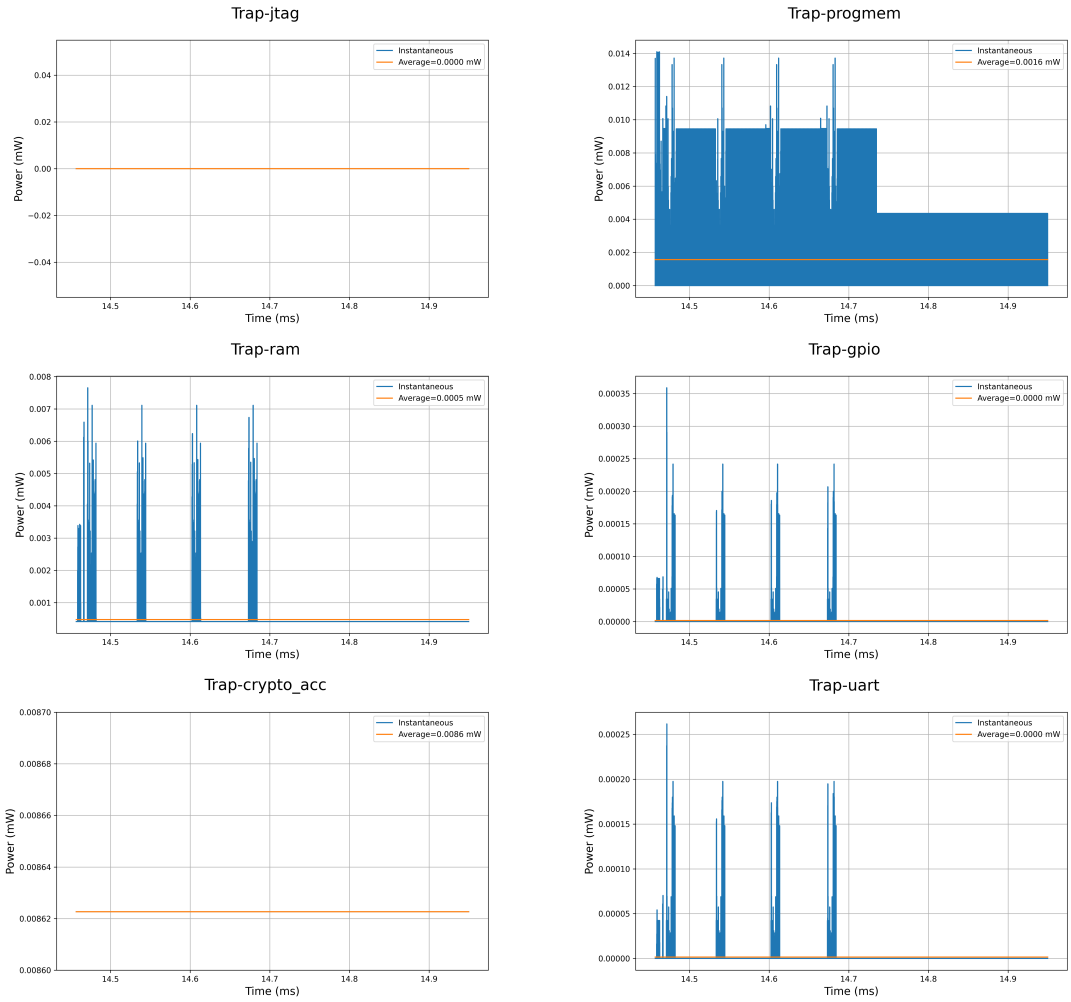

Fig. 24. Power Consumption of the Trap Test: other submodules of ACoreChip

5.1.3 Summary of Results

The simulation time for the trap test was approximately 14.9497 ms, during which the average power consumption was 0.0593 mW. Given that the enabled-core part consumes the majority of power, attention was paid on analyzing the performance of A-Core. Specifically, the average power consumption of ACoreChip was measured in the enabled-core part, which was found to be 0.5377 mW. The power profile analysis indicated that the power-intensive happens when the number is under factorial calculation.

To provide a detailed breakdown of power consumption among different submodules of ACoreChip in the enabled-core part, the results were present in Table I. This table reports the average power consumption in mW with 7 decimals to show the differences between modules. As the central module of the chip, the core had the highest average power consumption of 0.4977745 mW, which accounted for 92.5779% of ACoreChip. The crypto_acc had the second-highest average power consumption of 0.0086227 mW, accounting for 1.6037% of ACoreChip. The progmem and ram

consumed less power, and accoundts for 0.2914% and 0.0879% of the whole chip power consumption separately. The modules axi4l_mm_demux, gpio, and uart had very low average power consumption, each accounting for less than 0.0010% of ACoreChip, with values of 0.0000053 mW, 0.0000016 mW, and 0.0000015 mW, respectively. The jtag module had an average power consumption of 0 mW.

Table I. Power Consumption of Trap Test: submodues of ACoreChip

Module	Average power [mW]	Percentage of ACoreChip
core	0.4977745	92.5779%
crypto_acc	0.0086227	1.6037%
progmem	0.0015668	0.2914%
ram	0.0004724	0.0879%
axi4l_mm_demux	0.0000053	0.0010%
gpio	0.0000016	0.0003%
uart	0.0000015	0.0003%
jtag	0.0000000	0.0000%

5.2 Blinky Test

The "Blinky" test program is a configuration of a RV32I C-language program that is executed on the A-Core. The source code corresponding to the blinky test program can be found in Appendix B. The test involves defining functions that are used during the simulation. Once the core is enabled, it turns off all outputs (gpo) and then cyclically shifts the asserted output by setting gpo to "1" and then to "0" through a for loop. Then, it blinks 4 LEDs with a delay.

5.2.1 ModelSim Results

As illustrated in Fig. 25, the blinky test takes a total of 7.4506 ms. Prior to 6.9579 ms, the core was undergoing preparation for the simulation. At 6.9579 ms, the signal core_en transitioned from 0 to 1, indicating the activation of the core for simulation purposes. The detailed transactions of the gpio are depicted in Fig. 26, where it can be observed that the A-Core initially turned off all outputs and then cyclically shifted the asserted outputs. Subsequently, all outputs were set to 1 and then reset to 0 one by one through the loop. Finally, the A-Core blinked four LEDs with a delay between each blink, which starts at a time stamp of 7.4026 ms.

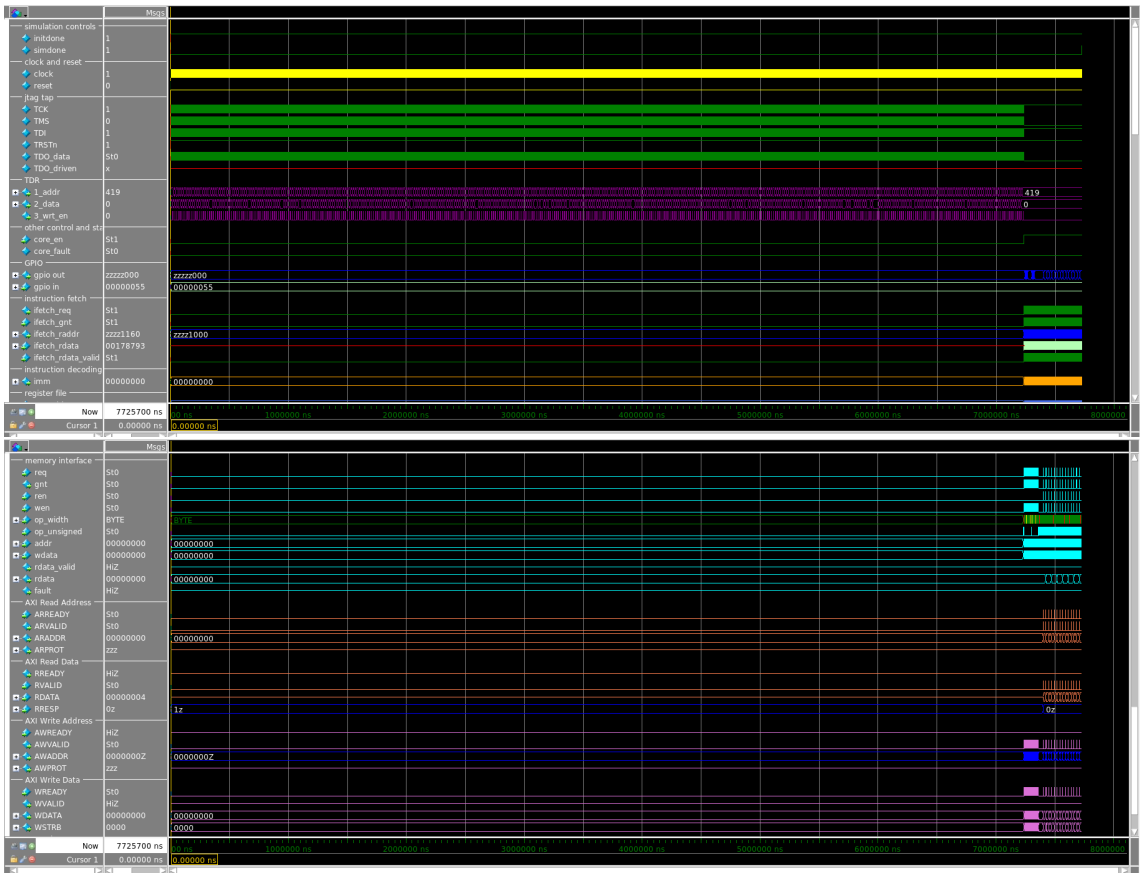

Fig. 25. ModelSim Results of Blinky Test

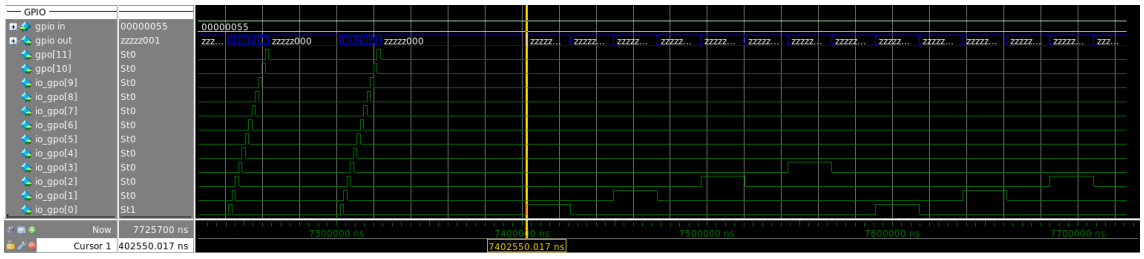

Fig. 26. ModelSim Results of the Blinky Test: GPIO

5.2.2 Profile Results: Enable F-extension

The power consumption analysis of the blinky test program is presented. Fig. 27 illustrates the power consumption of the top-level module ACoreChip, with an average power consumption of 0.0978 mW during this phase. The power remained low until approximately 6.9500 ms, during which the core was inactive. The enabled-core phase is also analyzed, which began at 6.9579 ms, exhibited a distinct pattern of power consumption. The A-Core started blinking at around 7.4026 ms.

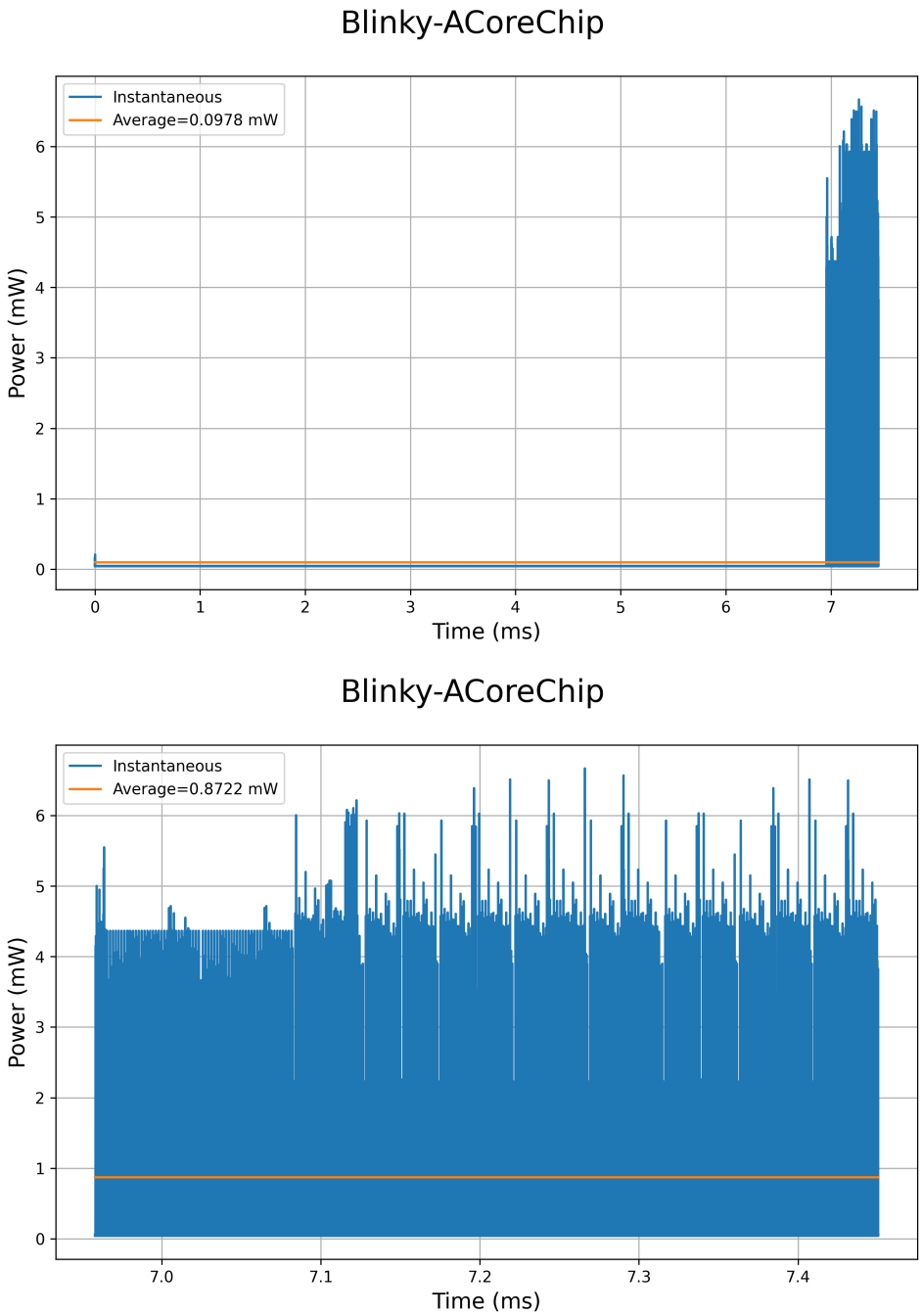

Fig. 27. Power Consumption of Blinky Test

Fig. 28 shows the contribution of each submodule to the whole power consumption. It was observed that the submodule core was the primary contributor, while the jtag submodule remained active before 6.9579 ms and consumed negligible power during the core-enable phase.

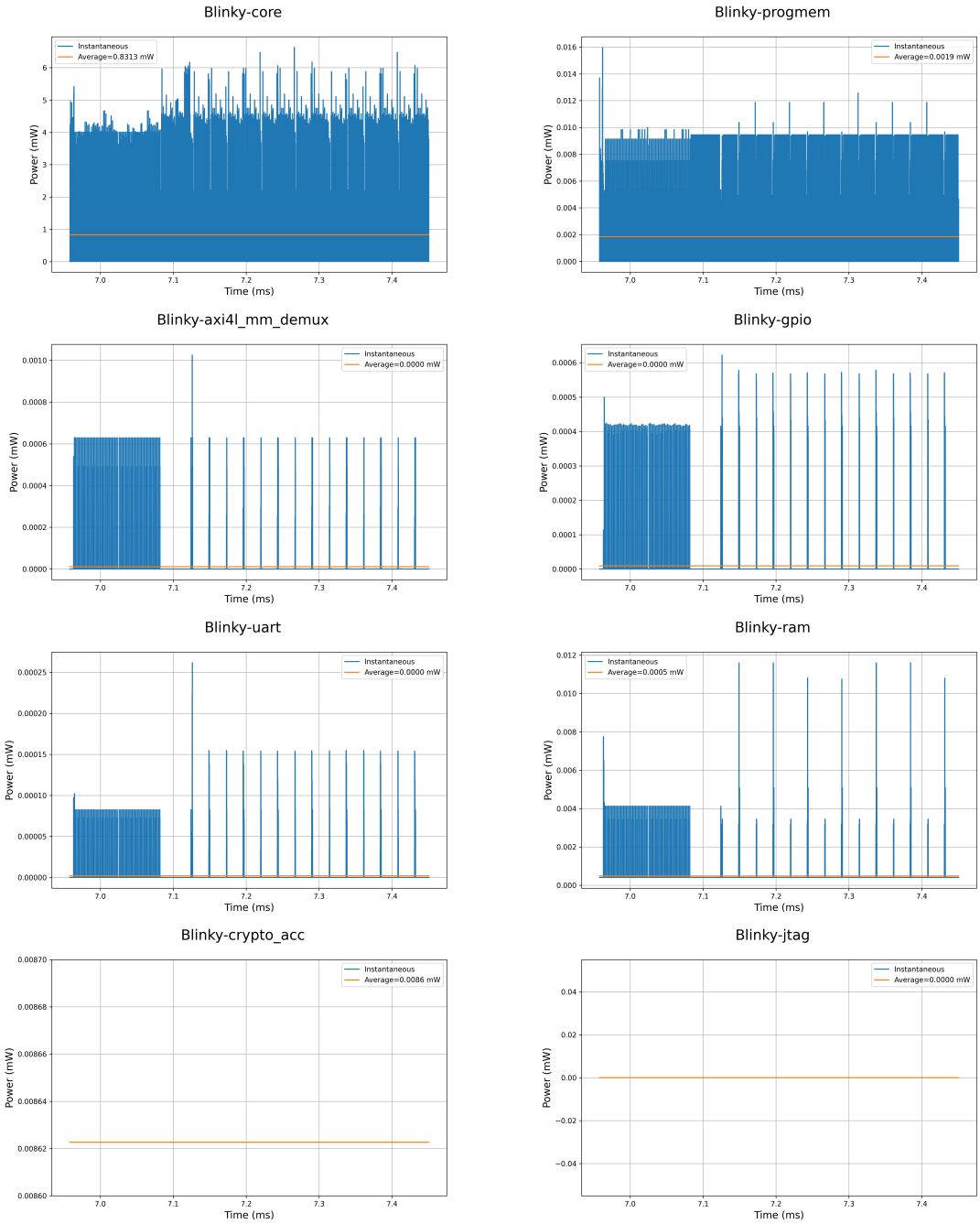

Fig. 28. Power Consumption of Blinky Test: other submodules of ACoreChip

The submodules within the core were analyzed, such as alu_block, control, decoder_block, and ppl_ctrl. The results are presented in Fig. 29. Among the submodules under the alu_block, namely alu, fpu_block, and mult_block, the fpu_block consumed the highest power, showed in Fig. 30.

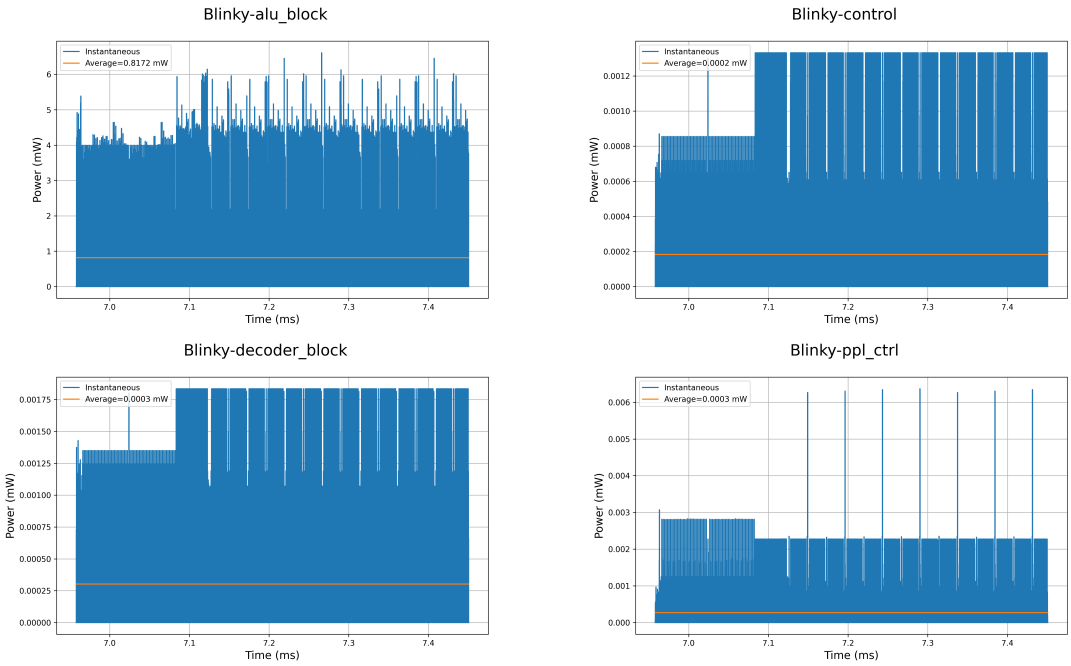

Fig. 29. Power Consumption of Blinky Test: submodules of core

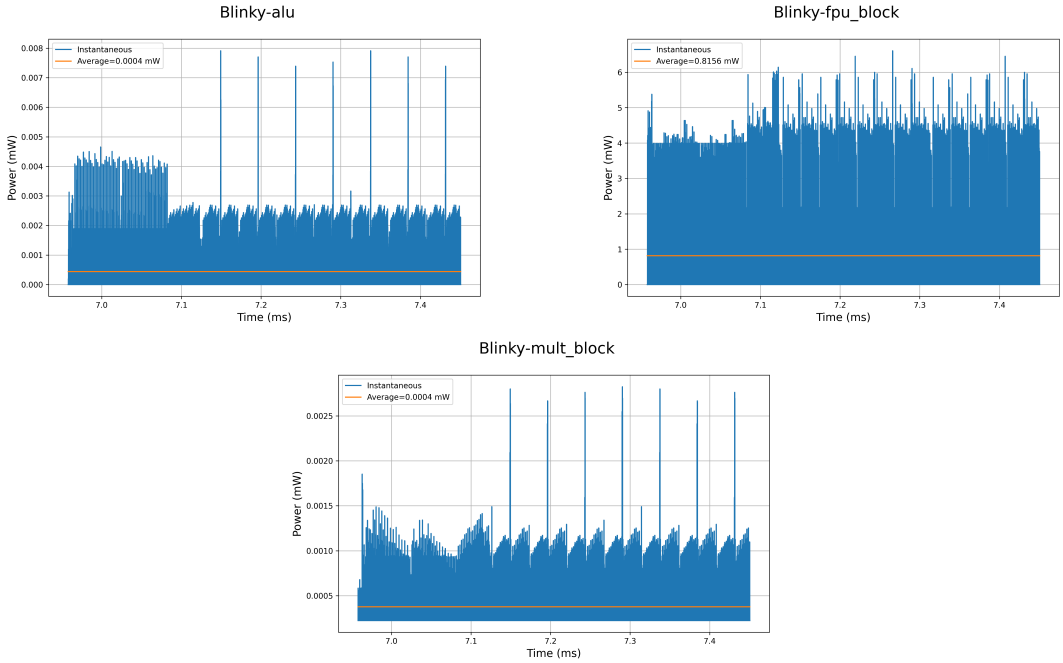

Fig. 30. Power Consumption of Blinky Test: submodules of alu_block

5.2.3 Profile Results: Disable F-extension

Based on the aforementioned results, it is clear that the submodule fpu_block within the ACoreChip/core/alu_block constitutes the primary factor influencing power consumption within the ACoreChip. However, according to the source code of the blinky test (Appendix B), the floating-point operation is not required for this simulation. Therefore, the F-extension (supports single precision floating point instructions) can be disabled for this simulation using a function called "disable_f".

The power consumption of the ACoreChip before and after disabling the F-extension is depicted in Fig. 31. It is evident that the power consumption has significantly decreased from 0.8722 mW to 0.0648 mW, a reduction of approximately 92.5705%. The time at which the core is enabled has changed from 6.9589 ms to 7.2331 ms, indicating that the A-Core requires more time to prepare because of the increased complexity of the source code. However, the simulation time of the actual blinky does not show a significant difference. The detailed power consumption of the submodules under ACoreChip with the disabled F-extension can be found in Fig. 32.

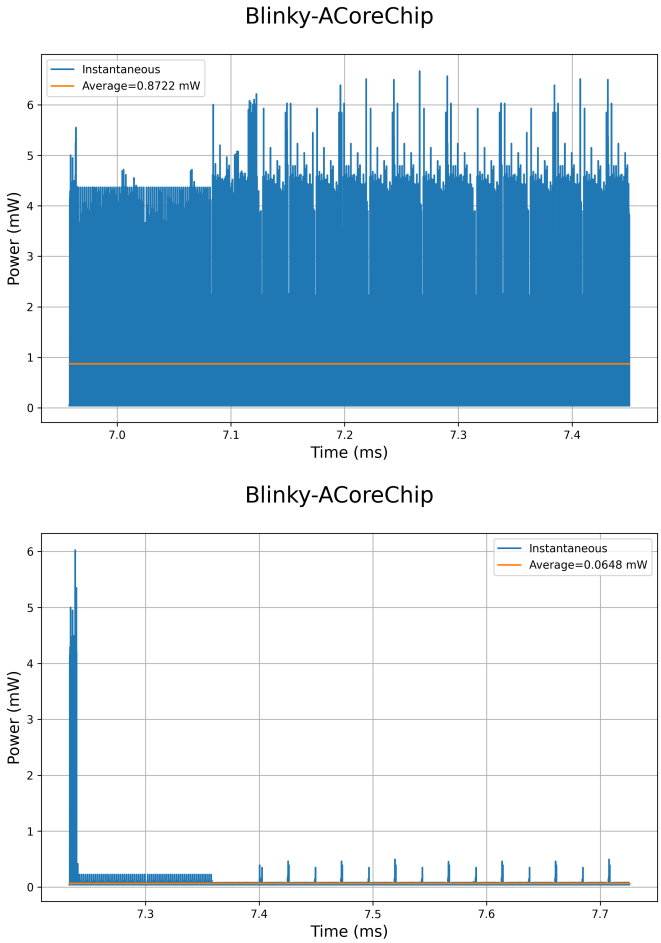

Fig. 31. Power Consumption of Blinky Test: ACoreChip, with F-extension (top) and without F-extension (bottom)

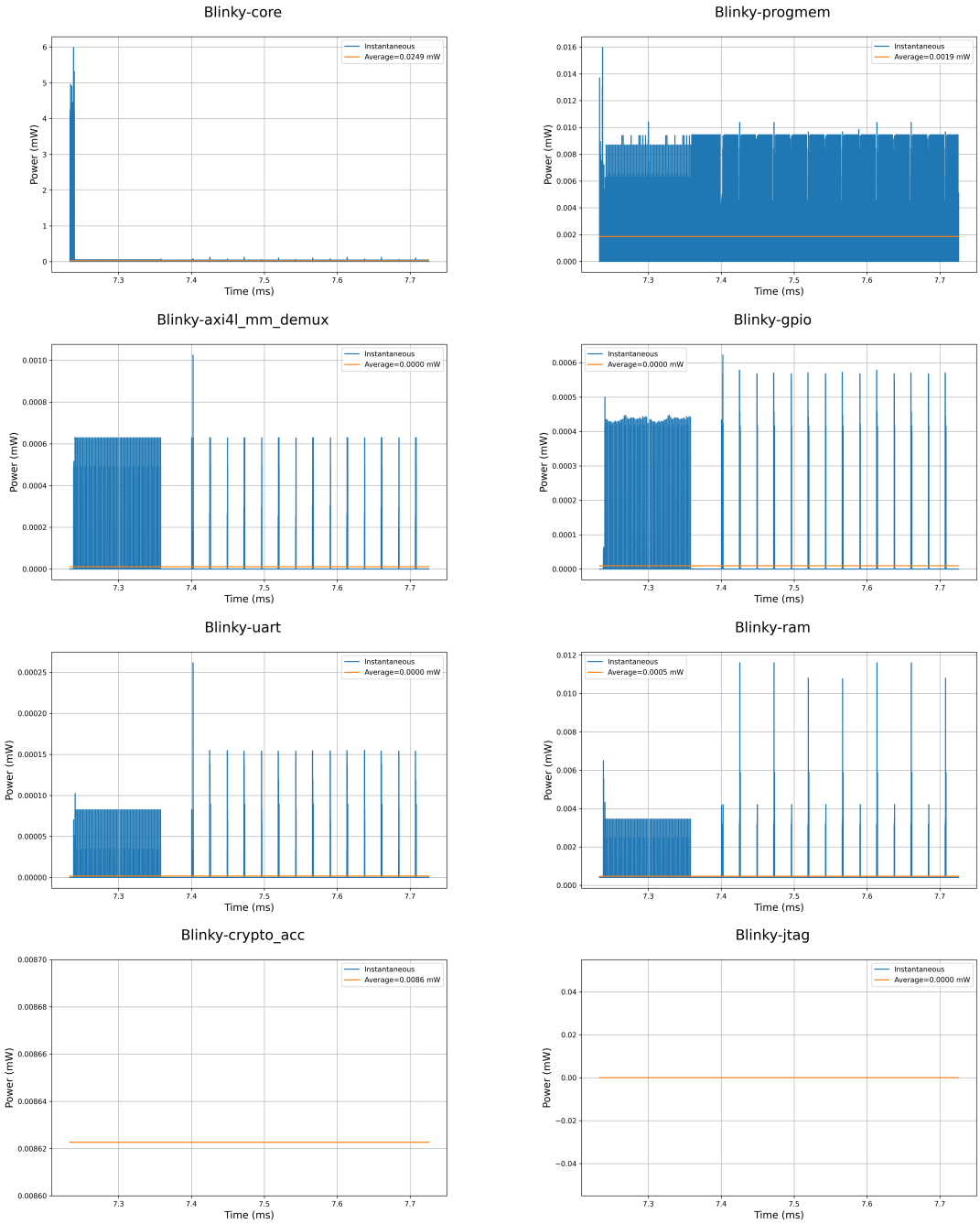

Fig. 32. Power Consumption of Blinky Test: submodules of ACoreChip, without F-extension

5.2.4 Summary of Results

The power consumption analysis of ACoreChip in the blinky test with the F-extension is summarized in Tables II, III, and IV. In this section, the power results also used 7 decimals to identify the differences between modules. As indicated, the dominant contributor to ACoreChip's power consumption is the core module, which accounts for 95.3168% of the total power consumption. Among the submodules of the module core, the submodule alu_block is the most power-hungry, consuming 98.2986% of the core's total power consumption. Furthermore, within the alu_block submodule, the fpu_block submodule is responsible for 99.8042% of alu_block's total power consumption.

Table II. Power Consumption of Blinky Test: submodues of ACoreChip

Module	Average power [mW]	Percentage of ACoreChip
core	0.8313303	95.3168%
crypto_acc	0.0086227	0.9886%
progmem	0.0018519	0.2123%
ram	0.0004838	0.0555%
axi4l_mm_demux	0.0000111	0.0013%
gpio	0.0000093	0.0011%
uart	0.0000019	0.0002%
jtag	0.0000000	0.0000%

Table III. Power Consumption of Blinky Test: submodues of core

Module	Average power [mW]	Percentage of core
alu_block	0.8171859	98.2986%
decoder_block	0.0003025	0.0364%
ppl_ctrl	0.0002704	0.0325%
control	0.0001828	0.0220%

Table IV. Power Consumption of Blinky Test: submodues of alu_block

Module	Average power [mW]	Percentage of alu_block
fpu_block	0.8155859	99.80421%
alu	0.0004395	0.05378%
mult_block	0.0003759	0.04600%

To investigate the impact of the F-extension on the power consumption, Table V presents the average power consumption of different modules of A-Core with and without the F-extension. The results indicate that disabling the F-extension can significantly reduce the power consumption of both ACoreChip and the module core. Specifically, when the F-extension is enabled, ACoreChip consumes 0.8721761 mW, while the core module consumes 0.8313303 mW. However, with the F-extension disabled, the power consumption of the core module drops to 0.0249320 mW, and ACoreChip's power consumption reduces to 0.0648478 mW. Modules such as crypto_acc, progmem, ram, axi4l_mm_demux, gpio, uart, and jtag exhibit minimal or no difference in power consumption with and without the F-extension. Notably, the power consumption of jtag remains constant at 0 mW in both cases.

Table V. Power Consumption of Blinky Test: ACoreChip, with and without F-extension

Module	P_{avg} with F-extension [mW]	P_{avg} without F-extension [mW]
ACoreChip	0.8721761	0.0648478
core	0.8313303	0.0249320
crypto_acc	0.0086227	0.0086227
progmem	0.0018519	0.0018590
ram	0.0004838	0.0004730
axi4l_mm_demux	0.0000111	0.0000110
gpio	0.0000093	0.0000095
uart	0.0000019	0.0000017
jtag	0.0000000	0.0000000

6 Conclusions

The objective of the work is using the post-layout data from the synthesized A-Core microprocessor to create a power profiling model which can do software, voltage and clock-frequency dependent time-varying power consumption profiles. By analyzing the power consumption patterns, designers can strike an optimal balance between performance and power consumption, catering to the requirements of specific applications. Furthermore, such a model facilitates the development of dynamic power management strategies, enabling systems to adapt their power consumption based on workload demands. Additionally, power profiling models serve as a valuable tool for benchmarking and comparing core implementations, ultimately guiding designers in selecting the most suitable implementations.

The developed power profiling model available in TheSyDeKick platform is an effective tool for analyzing the power consumption. The model simplifies the process for users and provides accurate power consumption profiling to optimize designs for better efficiency. The model is equipped with various features such as the generation of the basic structure of the core, top-level and submodule power consumption analysis of the specified module, analysis of power consumption within a specified time interval, and average power consumption analysis. By utilizing this model, designers can efficiently optimize their digital circuit designs for better power efficiency.

Literature review of related background knowledge was conducted and presented. Due to the open-source, customizable, and growing ecosystem, the RISC-V is becoming increasingly popular in the industry, education and research. ECD group from Aalto University developed their own modular and expandable RISC-V core A-Core, which can be used as a controller in the mixed-mode System-on-Chip designs. Power consumption in CMOS technology was reviewed. Various techniques for power management, including dynamic voltage scaling, clock gating, and power gating were discussed. Information and formats of parastics and waveform utilized in the power consumption calculation were introduced.

The development of power profiling algorithms primarily utilized TheSyDeKick and Python, where the former provided a simulation and verification environment. An interface was established for controlling and running simulations, which simplified the generation of test benches and execution, as well as post-processing for power profiling. This interface enabled the use of Python-based simplified power profiles for further hardware and power management system development. The power profiling algorithms operate based on various information extracted from the core during operation, such as signal change information and capacitance information of nets, which is then utilized to compute the power consumption of the core. Capacitance information was generated by digital flow and stored in SPEF file, while signal change information was generated through the post-route simulation in TheSyDeKick and stored as VCD file. To evaluate the feasibility of the power profiling model, different test targets were reviewed and simulated to evaluate their efficiency, as well as the model's correctness and generality. The dynamic power consumption of the A-Core was analyzed, and the results of different simulations were presented and summarized.

Future efforts could be directed towards the use of the existing power profiling model in power management system development. To avoid time-consuming RTL simulations in mixed-mode systems, a model that generates estimates for metrics as a function of supply voltage and clock-frequency with sufficient accuracy can be created to support the development of power control algorithms/software. The ultimate goal is the physical implementation of a full power management system.

In conclusion, this thesis has presented the development and evaluation of a dynamic power profile model for microprocessor cores, which is essential for supporting power management system development. The power profiling algorithms were implemented using TheSyDeKick and Python, and the workflow and design methodology were presented. Through simulations, the efficiency, correctness, and generality of the model were evaluated. The results demonstrated that the model can be used to evaluate the power consumption and efficiency of microprocessor cores. Future work could focus on integrating the power profiling model into power management system development and creating a model that generates estimates for metrics as a function of supply voltage and clock-frequency. The objectives of the thesis were successfully achieved, and the performance of the developed model was verified through simulations.

References

[1] S. Borkar and A. A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, no. 5, pp. 67–77, 2011.

[2] J. Rabaey, *Low power design essentials*. Springer Science & Business Media, 2009.

[3] D. M. Brooks *et al.*, "Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors," *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000.

[4] A. Jerraya and W. Wolf, *Multiprocessor systems-on-chips*. Elsevier, 2004.

[5] J. Kong, S. W. Chung, and K. Skadron, "Recent thermal management techniques for microprocessors," *ACM Computing Surveys (CSUR)*, vol. 44, no. 3, pp. 1–42, 2012.

[6] J. Pouwelse, K. Langendoen, and H. Sips, "Dynamic voltage scaling on a low-power microprocessor," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, 2001, pp. 251–259.

[7] B. Keller *et al.*, "A risc-v processor soc with integrated power management at submicrosecond timescales in 28 nm fd-soi," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 7, pp. 1863–1875, 2017.

[8] J. C. Wright *et al.*, "A dual-core risc-v vector processor with on-chip fine-grain power management in 28-nm fd-soi," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2721–2725, 2020.

[9] B. Zimmer *et al.*, "A risc-v vector processor with simultaneous-switching switched-capacitor dc–dc converters in 28 nm fdsoi," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 930–942, 2016.

[10] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[11] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: Scheduling and synchronization*. CRC press, 2018.

[12] D. A. Patterson and J. L. Hennessy, *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann, 2016.

[13] J. Parkhurst, J. Darringer, and B. Grundmann, "From single core to multi-core: Preparing for a new exponential," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006, pp. 67–72.

[14] S. Akhter and J. Roberts, *Multi-core programming*. Intel press Hillsboro, Oregon, 2006, vol. 33.

[15] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (mpsoc) technology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 10, pp. 1701–1713, 2008.

[16] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-chip Verification: Methodology and Techniques*. Springer Science & Business Media, 2007.

[17] J. Šilc, J. Silc, B. Robic, and T. Ungerer, *Processor Architecture: From Dataflow to Superscalar and Beyond; with 34 Tables*. Springer Science & Business Media, 1999.

[18] D. L. Kuck, *Structure of Computers and Computations*. John Wiley & Sons, Inc., 1978.

[19] D. TMS320C55x, "Cpu reference guide," *Texas Instruments Inc*, 2004.

[20] L. Gopal, N. S. M. Mahayadin, A. K. Chowdhury, A. A. Gopalai, and A. K. Singh, "Design and synthesis of reversible arithmetic and logic unit (alu)," in *2014 International Conference on Computer, Communications, and Control Technology (I4CT)*, IEEE, 2014, pp. 289–293.

[21] H.-W. Tseng, "Instruction set architecture," 2006.

[22] T. Jamil, "Risc versus cisc," *Ieee Potentials*, vol. 14, no. 3, pp. 13–16, 1995.

[23] D. Bhandarkar and D. W. Clark, "Performance from architecture: Comparing a risc and a cisc with similar hardware organization," in *Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, 1991, pp. 310–319.

[24] U. Degenbaev, "Formal specification of the x86 instruction set architecture," 2012.

[25] J. Goodacre and A. N. Sloss, "Parallelism and the arm instruction set architecture," *Computer*, vol. 38, no. 7, pp. 42–50, 2005.

[26] D. Kanter, "Risc-v offers simple, modular isa," *Microprocessor Report*, pp. 4–5, 2016.

[27] E. Cui, T. Li, and Q. Wei, "Risc-v instruction set architecture extensions: A survey," *IEEE Access*, vol. 11, pp. 24 696–24 711, 2023.

[28] A. S. Waterman, *Design of the RISC-V instruction set architecture*. University of California, Berkeley, 2016.

[29] E. Torres-Sánchez, J. Alastruey-Benedé, and E. Torres-Moreno, "Developing an ai iot application with open software on a risc-v soc," in *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, IEEE, 2020, pp. 1–6.

[30] M. Sharma, E. Bhatnagar, K. Puri, A. Mitra, and J. Agarwal, "A survey of risc-v cpu for iot applications," *Available at SSRN 4033491*, 2022.

[31] P. D. Schiavone *et al.*, "Arnold: An efpga-augmented risc-v soc for flexible and low-power iot end nodes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 4, pp. 677–690, 2021.

[32] Y. Lee *et al.*, "An agile approach to building risc-v microprocessors," *ieee Micro*, vol. 36, no. 2, pp. 8–20, 2016.

[33] A. Waterman *et al.*, "The risc-v instruction set manual," *Volume I: User-Level ISA', version*, vol. 2, 2014.

[34] A. S. Waterman, *Design of the RISC-V instruction set architecture*. University of California, Berkeley, 2016.

[35] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanovic, "The risc-v instruction set manual volume 2: Privileged architecture version 1.7," University of California at Berkeley Berkeley United States, Tech. Rep., 2015.

[36] M. Kosunen, V. Hirvonen, A. Korsman, and O. Simola, *A-Core*, https://gitlab.com/a-core, [Accessed: 2023-4-10].

[37] J. Bachrach *et al.*, "Chisel: Constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1216–1225.

[38] M. Kosunen, *TheSyDeKick-complete kit for system-on-chip development*, https://github.com/TheSystemDevelopmentKit, [Accessed: 2023-4-10].

[39] B. Razavi, *Fundamentals of microelectronics*. John Wiley & Sons, 2021.

[40] T. Simunic, "Dynamic management of power consumption," *Power aware computing*, pp. 101–125, 2002.

[41] N. S. Kim *et al.*, "Leakage current: Moore's law meets static power," *computer*, vol. 36, no. 12, pp. 68–75, 2003.

[42] M. N. Horenstein, *Microelectronic circuits and devices*. Prentice-Hall, Inc., 1990.

[43] F. C. Lee and X. Zhou, "Power management issues for future generation microprocessors," in *11th International Symposium on Power Semiconductor Devices and ICs. ISPSD'99 Proceedings (Cat. No. 99CH36312)*, IEEE, 1999, pp. 27–33.

[44] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, 2010, pp. 1–8.

[45] B. Keller *et al.*, "Sub-microsecond adaptive voltage scaling in a 28nm fd-soi processor soc," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, 2016, pp. 269–272.

[46] L. Benini and G. DeMicheli, *Dynamic power management: design techniques and CAD tools*. Springer Science & Business Media, 1997.

[47] V. Tiwari, R. Donnelly, S. Malik, and R. Gonzalez, "Dynamic power management for microprocessors: A case study," in *Proceedings Tenth International Conference on VLSI Design*, IEEE, 1997, pp. 185–192.

[48] J. Haj-Yahya *et al.*, "Power management of modern processors," *Energy Efficient High Performance Processors: Recent Approaches for Designing Green High Performance Computing*, pp. 1–55, 2018.

[49] J. C. Wright *et al.*, *A dual-core risc-v vector processor with on-chip fine-grain power management in 28-nm fd-soi*, 2020.

[50] B. Keller *et al.*, *A risc-v processor soc with integrated power management at submicrosecond timescales in 28 nm fd-soi*, 2017.

[51] C. Schmidt *et al.*, *Programmable fine-grained power management and system analysis of risc-v vector processors in 28-nm fd-soi*, 2020.

[52] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 8, no. 3, pp. 299–316, 2000.

[53] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," *Mobile Computing*, pp. 449–471, 1996.

[54] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 813–833, 1999.

[55] *Ieee standard for integrated circuit (ic) open library architecture (ola)*, 2020.

[56] S. Ahuja, D. A. Mathaikutty, G. Singh, J. Stetzer, S. K. Shukla, and A. Dingankar, "Power estimation methodology for a high-level synthesis framework," in *2009 10th International Symposium on Quality Electronic Design*, IEEE, 2009, pp. 541–546.

[57] D. E. Galbi, K. Kannan, and M. Hudson, "Measuring active power using pt px a user perspective," *SNUG Bostone*, pp. 1–13, 2010.

[58] *Ieee standard verilog hardware description language*, 2001.

[59] C. Papameletis, "Development of design methodologies and cad tools for system-level evaluation of interconnect reliability issues in soc designs," 2010.

[60] J. Becker, M. Huebner, and M. Ullmann, *Power estimation and power measurement of xilinx virtex fpgas: Trade-offs and limitations*, 2003.

[61] S. Porrasmaa, "Programmatic integrated circuit design in context of Analog-to-Digital converters," M.S. thesis, Aalto University, Finland, 2021.

[62] *ModelSim ® SE user's manual*, https://www.lirmm.fr/~bosio/TPVHDL/docs/modelsim_se_user.pdf, [Accessed: 2023-4-10].

[63] *Modelsim command reference manual*, https://cseweb.ucsd.edu/~hadi/teaching/cs3220/01-2014fa/doc/modelsim/ModelSim_Reference_Manual_v10.1c.pdf, [Accessed: 2023-4-10].

A Trap Test

```
#include <stdint.h>
#include "a-core-utils.h"
#include "a-core.h"
#include "acore-gpio.h"

// statically initialize some data in .data section
int result = 3;

// factorial of an integer
int factorial(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n-1);
}

void delay(int cycles) {
    for (int i = 0; i < cycles; i++) asm("nop");
}

void main() {
    // test function calls
    int a = 4;
    result = factorial(a);
    int thresh = 50; //delay

    // test memory write
    *(volatile int*)(0x20000000) = result;

    // test memory read
    volatile int r = *(volatile int*)(0x20000000);

    asm("li t1, 0x20000000");
    asm("li t2, 0");
    asm("sb t2, 2(t1)");
    asm("addi t2, t2, 1");
    asm("sb t2, 3(t1)");
    asm("addi t2, t2, 1");
    asm("sh t2, 0(t1)");
    asm("addi t2, t2, 1");
    asm("sh t2, 1(t1)");
    asm("addi t2, t2, 1");
    asm("addi t2, t2, 1");
```

```
delay(thresh);

a = 4;
result = factorial(a);
*(volatile int*)(0x20000000) = result;
r = *(volatile int*)(0x20000000);
asm("li t1, 0x20000000");
asm("li t2, 0");
asm("sb t2, 2(t1)");
asm("addi t2, t2, 1");
asm("sb t2, 3(t1)");
asm("addi t2, t2, 1");
asm("sh t2, 0(t1)");
asm("addi t2, t2, 1");
asm("sh t2, 1(t1)");
asm("addi t2, t2, 1");
asm("addi t2, t2, 1");

delay(thresh);

a = 7;
result = factorial(a);
*(volatile int*)(0x20000000) = result;
r = *(volatile int*)(0x20000000);
asm("li t1, 0x20000000");
asm("li t2, 0");
asm("sb t2, 2(t1)");
asm("addi t2, t2, 1");
asm("sb t2, 3(t1)");
asm("addi t2, t2, 1");
asm("sh t2, 0(t1)");
asm("addi t2, t2, 1");
asm("sh t2, 1(t1)");
asm("addi t2, t2, 1");
asm("addi t2, t2, 1");

delay(thresh);

a = 10;
result = factorial(a);
*(volatile int*)(0x20000000) = result;
r = *(volatile int*)(0x20000000);
asm("li t1, 0x20000000");
asm("li t2, 0");
```

```
    asm("sb t2, 2(t1)");
    asm("addi t2, t2, 1");
    asm("sb t2, 3(t1)");
    asm("addi t2, t2, 1");
    asm("sh t2, 0(t1)");
    asm("addi t2, t2, 1");
    asm("sh t2, 1(t1)");
    asm("addi t2, t2, 1");
    asm("addi t2, t2, 1");

    delay(thresh);

    // infinite loop
    for(;;);
}
```

B Blinky Test

```
#include <stdint.h>
#include "a-core-utils.h"
#include "a-core.h"
#include "acore-gpio.h"

#define GPIO_OUT (volatile uint32_t*)0x30000010
#define GPIO_IN  (volatile uint32_t*)0x30000000

// init data registers to zero
void gpo_init(volatile uint32_t* base_addr) {
    *base_addr = 0;
}

// index 0 is the least significant bit
// index 31 is the most significant bit
void gpo_set_bit(volatile uint32_t* addr, int value, int index) {
    if (value) {
        *addr |= 1 << index;
    } else {
        *addr &= ~(1 << index);
    }
}

void gpo_write(volatile uint32_t* base_addr, uint32_t value) {
    *base_addr = value;
```

```
}

void delay(int cycles) {
    for (int i = 0; i < cycles; i++) asm("nop");
}

void main() {

    // disable the F-extension
    // disable_f();

    int thresh = 20; //delay

    // turn all outputs off
    gpo_init((volatile uint32_t*)GPIO_OUT);

    // shift asserted output around in a loop for a while
    uint32_t bits = 1;
    for (int i = 0; i < 64+1; i++) {
        gpo_write((volatile uint32_t*)GPIO_OUT, bits);
        if (bits == 0)
            bits = 1;
        else
            bits <<= 1;
    }

    delay(40);

    // reset the outputs to be 0
    gpo_write((volatile uint32_t*)GPIO_OUT, 0);

    int i = 0;  //bit setting gpo[0]
    for (;;) {
        for (int i = 0; i < 4; i++) {
            gpo_set_bit(GPIO_OUT, 1, i);
            delay(thresh);
            gpo_set_bit(GPIO_OUT, 0, i);
            delay(thresh);

        }
    }
}
```