



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



Faculty of Science and Technology  
Department of Science and Technology

Dissertation submitted in partial fulfilment of the requirement for the Master's  
Degree  
Field :Electronics  
Specialty : Instrumentation

Presented by:

➤ Yehekour Badreddine.

## Title

---

*Design and implementation of mixed signal chip using ic  
design flow*

---

Publicly defended on : 16/06/2026

In front of the Jury:

|                     |               |        |                          |
|---------------------|---------------|--------|--------------------------|
| Belkassa Missoum    | President     | M.A.A. | University of Tissemsilt |
| Sahnoune Ahmed      | Examiner      | M.C.B. | University of Tissemsilt |
| HEDJAR Hakima       | Supervisor    | M.C.B. | University of Tissemsilt |
| Halimaoui Alaaddine | Co-Supervisor |        | University of Tissemsilt |

Academic year: 2025-2026

# Dedication

In the Name of Allah, the Most Gracious, the Most Merciful

With the help, guidance, and blessings of Almighty God, I have completed this thesis. I dedicate this work to all those who have supported and encouraged me throughout this academic journey.

First and foremost, I dedicate this achievement to my beloved parents. Your unconditional love, endless sacrifices, unwavering support, and constant prayers have been the foundation of my success. You have always been my source of strength, inspiration, and motivation. Every accomplishment I achieve reflects the values, principles, and determination that you instilled in me. May Allah bless you with good health, happiness, and a long life filled with peace and prosperity.

I also dedicate this work to my dear family especially my sisters, whose affection, encouragement, and confidence in me have provided comfort and strength during every stage of my studies. Your presence has been a constant source of motivation throughout this journey.

To my friends and colleagues, To my classmates and graduation cohort (promo Electronique Instrumentation 2025/ 2026 ), with whom I shared unforgettable academic moments filled with cooperation and friendship. thank you for your friendship, support, and encouragement. The moments we shared, the challenges we overcame together, and the memories we created have made this journey meaningful and unforgettable.

I would also like to express my gratitude to all my teachers and professors who generously shared their knowledge, guidance, and experience. Your dedication and support have played a significant role in my academic and personal development.

Finally, I dedicate this work to everyone who contributed, directly or indirectly, to the completion of this thesis. Your encouragement, advice, and belief in my abilities have helped me reach this important milestone.

# Acknowledgements

In the Name of Allah, the Most Gracious, the Most Merciful.

All praise and thanks are due to Almighty Allah, who granted me the strength, patience, and perseverance to complete this work successfully.

I would like to express my deepest gratitude to my supervisor, **Dr. HEDJAR Hakima**, for her valuable guidance, continuous support, constructive feedback, and encouragement throughout the development of this thesis.

We are also sincerely thankful to our co-supervisor, **Mr. Halimaoui Alaaddine**, for his guidance and helpful suggestions, which greatly contributed to the development of our work. His support and input were instrumental at critical stages of our research.

I would also like to sincerely thank the members of the examination committee, **Dr. Sahnoune Ahmed**, and **Dr. Belkassa Missoum**, for accepting to evaluate this thesis and for their time, effort, and valuable remarks.

My heartfelt thanks go to all the professors and staff of the **Department of Electrical and Electronic Engineering**, who provided me with knowledge, support, and guidance throughout my academic journey.

I am deeply grateful to my family for their unconditional love, patience, encouragement, and constant support.

I would also like to thank my friends and colleagues for their friendship and encouragement.

Finally, I extend my gratitude to everyone who contributed directly or indirectly to the completion of this work.

*Thank you all for your support and encouragement.*

## Abstract

The continuous evolution of semiconductor technology has increased the demand for embedded systems that offer higher performance, lower power consumption, and reduced development cost. In this context, this thesis presents the design and implementation of a RISC-V based microcontroller using reusable IP cores and open-source Electronic Design Automation (EDA) tools.

The work begins with an overview of integrated circuit design fundamentals, semiconductor technologies, and digital system design methodologies. It then introduces microcontroller architectures and justifies the selection of a RISC architecture based on the PicoRV32 processor core. The proposed microcontroller integrates ROM, RAM, GPIO, UART, and I<sup>2</sup>C peripherals through a memory-mapped bus architecture, providing a flexible and scalable embedded platform.

The implementation phase focuses on IP integration, hardware description in Verilog, system verification, synthesis, and physical design using open-source tools. The final design was successfully implemented in 130 nm technology and verified through simulation and physical validation.

The obtained results demonstrate that the proposed microcontroller achieves low power consumption, compact silicon area, and reliable timing performance while maintaining a low-cost development flow. The project confirms the effectiveness of combining reusable IP cores with open-source EDA tools for the development of modern embedded systems and future ASIC designs.

## Résumé

L'évolution continue des technologies des semi-conducteurs a entraîné une demande croissante pour des systèmes embarqués offrant de meilleures performances, une faible consommation d'énergie et un coût de développement réduit. Dans ce contexte, ce mémoire présente la conception et l'implémentation d'un microcontrôleur basé sur l'architecture RISC-V en utilisant des cœurs IP réutilisables et des outils EDA open-source.

Ce travail commence par une étude des fondements de la conception des circuits intégrés, des technologies des semi-conducteurs et des méthodologies de conception numérique. Il présente ensuite les architectures de microcontrôleurs et justifie le choix d'une architecture RISC basée sur le cœur PicoRV32. Le microcontrôleur proposé intègre des mémoires ROM et RAM ainsi que des périphériques GPIO, UART et I<sup>2</sup>C à travers une architecture de bus à mémoire mappée, offrant ainsi une plateforme embarquée flexible et évolutive.

La phase d'implémentation porte sur l'intégration des cœurs IP, la description matérielle en Verilog, la vérification du système, la synthèse et la conception physique à l'aide d'outils open-source. La conception finale a été implémentée avec succès en technologie 130 nm et validée par simulation et vérification physique.

Les résultats obtenus montrent que le microcontrôleur proposé présente une faible consommation énergétique, une surface silicium réduite et des performances temporelles satisfaisantes tout en conservant un coût de développement minimal. Ce projet démontre l'efficacité de l'utilisation conjointe de cœurs IP réutilisables et d'outils EDA open-source pour le développement de systèmes embarqués modernes et de futures conceptions ASIC.

## ملخص

أدى التطور المستمر في تكنولوجيا أشباه الموصلات إلى زيادة الحاجة إلى الأنظمة المدمجة التي توفر أداءً أعلى واستهلاكاً أقل للطاقة وتكلفة تطوير منخفضة. في هذا الإطار، يقدم هذا العمل تصميم وتنفيذ متحكم دقيق (Microcontroller) يعتمد على معمارية RISC-V باستخدام نوى IP قابلة لإعادة الاستخدام وأدوات تصميم إلكتروني مفتوحة المصدر (EDA).

يبدأ هذا العمل بدراسة أساسيات تصميم الدوائر المتكاملة وتقنيات أشباه الموصلات ومنهجيات التصميم الرقمي، ثم يستعرض معماريات المتحكمات الدقيقة ويقرر اختيار معمارية RISC المعتمدة على نواة PicoRV32. يدمج المتحكم المقترح ذاكرات ROM و RAM بالإضافة إلى الوحدات الطرفية GPIO و UART و I<sup>2</sup>C من خلال بنية ناقل ذات عنونة معتمدة على الذاكرة (Memory-Mapped Bus)، مما يوفر منصة مدمجة مرنة وقابلة للتطوير.

تركز مرحلة التنفيذ على دمج نوى IP، والوصف العتادي باستخدام لغة Verilog، والتحقق من النظام، والتوليف المنطقي، والتصميم الفيزيائي باستخدام أدوات مفتوحة المصدر. وقد تم تنفيذ التصميم النهائي بنجاح باستخدام تقنية 130 نانومتر والتحقق منه من خلال المحاكاة والتحقق الفيزيائي.

أظهرت النتائج أن المتحكم المقترح يحقق استهلاكاً منخفضاً للطاقة ومساحة سيليكون مدمجة وأداءً زمنياً موثقاً مع المحافظة على تكلفة تطوير منخفضة. كما يؤكد هذا المشروع فعالية الجمع بين نوى IP القابلة لإعادة الاستخدام وأدوات EDA مفتوحة المصدر في تطوير الأنظمة المدمجة الحديثة وتصميمات الدوائر المتكاملة المستقبلية.

# Contents

|                                                         |             |
|---------------------------------------------------------|-------------|
| <b>Dedication</b>                                       | <b>i</b>    |
| <b>Acknowledgements</b>                                 | <b>ii</b>   |
| <b>Table of contents</b>                                | <b>ix</b>   |
| <b>list of figures</b>                                  | <b>xi</b>   |
| <b>list of tables</b>                                   | <b>xii</b>  |
| <b>Abbreviations</b>                                    | <b>xiii</b> |
| <b>General introduction</b>                             | <b>1</b>    |
| <b>1 Integrated Circuit Design Fundamentals</b>         | <b>3</b>    |
| 1.1 Introduction: . . . . .                             | 3           |
| 1.2 Definition of integrated circuits: . . . . .        | 3           |
| 1.3 Classification of integrated circuits: . . . . .    | 4           |
| 1.3.1 Classification by function: . . . . .             | 4           |
| 1.3.2 Classification by technology: . . . . .           | 6           |
| 1.3.3 Classification by level of integration: . . . . . | 6           |

|        |                                                          |    |
|--------|----------------------------------------------------------|----|
| 1.4    | Comparison between ICs types: . . . . .                  | 7  |
| 1.5    | Integrated circuits design: . . . . .                    | 7  |
| 1.6    | Importance of ICs design: . . . . .                      | 7  |
| 1.7    | Semiconductor technology overview: . . . . .             | 8  |
| 1.7.1  | Introduction to semiconductor: . . . . .                 | 8  |
| 1.8    | Semiconductor fabrication technology: . . . . .          | 8  |
| 1.8.1  | Microtechnology: . . . . .                               | 9  |
| 1.8.2  | Nanotechnology: . . . . .                                | 9  |
| 1.9    | Semiconductor components: . . . . .                      | 9  |
| 1.9.1  | PN Junction: . . . . .                                   | 9  |
| 1.9.2  | Transistors: . . . . .                                   | 10 |
| 1.10   | CMOS Technology: . . . . .                               | 11 |
| 1.11   | Moore's law and technology scaling: . . . . .            | 11 |
| 1.12   | General ICs design methodology: . . . . .                | 12 |
| 1.12.1 | System specification and architectural design: . . . . . | 13 |
| 1.12.2 | Logic design: . . . . .                                  | 13 |
| 1.12.3 | Physical design: . . . . .                               | 13 |
| 1.12.4 | Verification and testing: . . . . .                      | 13 |
| 1.12.5 | The signoff: . . . . .                                   | 14 |
| 1.13   | Digital logic fundamentals: . . . . .                    | 14 |
| 1.13.1 | Combinational logic: . . . . .                           | 15 |
| 1.13.2 | Sequential logic: . . . . .                              | 15 |
| 1.14   | Intellectual property core : . . . . .                   | 15 |
| 1.15   | From logic gates to processor systems: . . . . .         | 16 |
| 1.16   | Conclusion: . . . . .                                    | 17 |

|          |                                                                  |           |
|----------|------------------------------------------------------------------|-----------|
| <b>2</b> | <b>Design and architecture of a RISC-V-based microcontroller</b> | <b>18</b> |
| 2.1      | Introduction: . . . . .                                          | 18        |
| 2.2      | Microcontroller: . . . . .                                       | 18        |
| 2.3      | Microprocessor vs Microcontroller: . . . . .                     | 19        |
| 2.4      | Justification for choosing the microcontroller: . . . . .        | 20        |
| 2.5      | CPU architecture: . . . . .                                      | 20        |
| 2.5.1    | RISC architecture: . . . . .                                     | 21        |
| 2.5.2    | CISC architecture: . . . . .                                     | 21        |
| 2.6      | Justification for choosing the RISC architecture: . . . . .      | 22        |
| 2.7      | Datapath: . . . . .                                              | 22        |
| 2.8      | Types of datapath designs: . . . . .                             | 23        |
| 2.8.1    | Single-cycle datapath: . . . . .                                 | 23        |
| 2.8.2    | Multi-Cycle Datapath: . . . . .                                  | 23        |
| 2.8.3    | Pipelined datapath: . . . . .                                    | 24        |
| 2.9      | Microcontroller components: . . . . .                            | 24        |
| 2.9.1    | Control unit(CU): . . . . .                                      | 24        |
| 2.9.2    | Arithmetic and logic unit (ALU): . . . . .                       | 25        |
| 2.9.3    | Registers: . . . . .                                             | 25        |
| 2.9.4    | Memory: . . . . .                                                | 26        |
| 2.10     | The Architecture of the proposed microcontroller: . . . . .      | 26        |
| 2.10.1   | Clock system: . . . . .                                          | 27        |
| 2.10.2   | PicoRV32: . . . . .                                              | 27        |
| 2.10.3   | System bus (Memory-mapped bus): . . . . .                        | 28        |
| 2.10.4   | Memory: . . . . .                                                | 28        |
| 2.10.5   | GPIO (General Purpose Input/Output): . . . . .                   | 29        |

|          |                                                                                        |           |
|----------|----------------------------------------------------------------------------------------|-----------|
| 2.10.6   | UART (Universal Asynchronous Receiver/Transmitter): . . . . .                          | 29        |
| 2.10.7   | I2C (Inter-Integrated Circuit): . . . . .                                              | 29        |
| 2.11     | Datapath architecture of the proposed microcontroller: . . . . .                       | 30        |
| 2.11.1   | Global Datapath Overview: . . . . .                                                    | 30        |
| 2.11.2   | Justification for choosing a single-cycle datapath: . . . . .                          | 31        |
| 2.11.3   | Internal datapath of the CPU: . . . . .                                                | 31        |
| 2.11.4   | System bus architecture: . . . . .                                                     | 32        |
| 2.11.5   | Data flow description: . . . . .                                                       | 33        |
| 2.11.6   | Design considerations: . . . . .                                                       | 33        |
| 2.12     | The microcontroller design flow: . . . . .                                             | 33        |
| 2.12.1   | Selectiong the system specification: . . . . .                                         | 33        |
| 2.12.2   | Selection of IP cores: . . . . .                                                       | 34        |
| 2.12.3   | Integration of ready IPs: . . . . .                                                    | 34        |
| 2.12.4   | Top-Level integration: . . . . .                                                       | 34        |
| 2.12.5   | Simulation and verification: . . . . .                                                 | 34        |
| 2.12.6   | Synthesis and implementation: . . . . .                                                | 35        |
| 2.12.7   | Testing and validation: . . . . .                                                      | 35        |
| 2.12.8   | Design strategy summary: . . . . .                                                     | 35        |
| 2.13     | Conclusion: . . . . .                                                                  | 36        |
| <b>3</b> | <b>IP integration for microcontroller and Implementation using Open-and free tools</b> | <b>37</b> |
| 3.1      | Introduction: . . . . .                                                                | 37        |
| 3.2      | Design methodology: . . . . .                                                          | 37        |
| 3.3      | Open-source tools: . . . . .                                                           | 38        |
| 3.3.1    | Vivado: . . . . .                                                                      | 38        |
| 3.3.2    | Icarus Verilog: . . . . .                                                              | 39        |

|        |                                                                           |    |
|--------|---------------------------------------------------------------------------|----|
| 3.3.3  | GTKWave: . . . . .                                                        | 39 |
| 3.3.4  | Laberlane: . . . . .                                                      | 40 |
| 3.4    | Simulation and verification: . . . . .                                    | 40 |
| 3.5    | Peripheral integration: . . . . .                                         | 43 |
| 3.6    | System verification: . . . . .                                            | 44 |
| 3.7    | Synthesis flow: . . . . .                                                 | 44 |
| 3.7.1  | The synthesis design: . . . . .                                           | 45 |
| 3.8    | Top level design: . . . . .                                               | 46 |
| 3.9    | The finale Soc: . . . . .                                                 | 50 |
| 3.10   | Result: . . . . .                                                         | 50 |
| 3.10.1 | Performance Efficiency: . . . . .                                         | 51 |
| 3.10.2 | Physical area utilization: . . . . .                                      | 51 |
| 3.10.3 | Power consumption: . . . . .                                              | 52 |
| 3.10.4 | Timing closure and clock characteristics: . . . . .                       | 52 |
| 3.11   | Comparison of the proposed microcontroller with existing chips: . . . . . | 53 |
| 3.12   | Architecture impact: . . . . .                                            | 53 |
| 3.13   | Implementation challenges: . . . . .                                      | 54 |
| 3.14   | Conclusion: . . . . .                                                     | 55 |

# List of Figures

|     |                                                                                                                                                                                 |    |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1 | Old version of an integrated circuit developed in 1961 on the left and the 14 nm-based Intel Xeon E5-2600 V3 18-core processor with 5.69 billion transistors on the right. [13] | 4  |
| 1.2 | Portion of the periodic table related to semiconductors[21]                                                                                                                     | 8  |
| 1.3 | P-N Junction Diode[24]                                                                                                                                                          | 10 |
| 1.4 | categorized of transistors based on element structure and operation principles.[27]                                                                                             | 11 |
| 1.5 | Transistors in Intel microprocessors.[29]                                                                                                                                       | 12 |
| 1.6 | IC design flow[30]                                                                                                                                                              | 12 |
| 1.7 | Logic gates.[31]                                                                                                                                                                | 15 |
| 1.8 | The steps to made the microcontroller.                                                                                                                                          | 17 |
| 2.1 | Difference Between CISC and RISC [41]                                                                                                                                           | 19 |
| 2.2 | Single-Cycle Datapath[46]                                                                                                                                                       | 23 |
| 2.3 | Multi-Cycle Datapath[46]                                                                                                                                                        | 23 |
| 2.4 | Pipelined-cycle datapath[46]                                                                                                                                                    | 24 |
| 2.5 | The Architecture used for the Microcontroller                                                                                                                                   | 27 |
| 2.6 | Global Datapath Overview                                                                                                                                                        | 30 |
| 2.7 | Instruction Datapath                                                                                                                                                            | 32 |

|      |                                                     |    |
|------|-----------------------------------------------------|----|
| 3.1  | Vivado interface . . . . .                          | 38 |
| 3.2  | Laberlane interface . . . . .                       | 40 |
| 3.3  | Add source in vivado. . . . .                       | 41 |
| 3.4  | Simulation of the IP . . . . .                      | 41 |
| 3.5  | Simulation and the wave form of the I2C . . . . .   | 42 |
| 3.6  | part from the configuration file . . . . .          | 44 |
| 3.7  | part from the configuration file . . . . .          | 45 |
| 3.8  | picture that shows the simulation filed . . . . .   | 49 |
| 3.9  | the top level of the microcontroller chip . . . . . | 49 |
| 3.10 | The finale chip . . . . .                           | 50 |

# List of Tables

|     |                                                               |    |
|-----|---------------------------------------------------------------|----|
| 1.1 | Comparison between analog and digital integrated circuits[17] | 7  |
| 2.1 | Comparison Between Microprocessor and Microcontroller[42]     | 20 |
| 2.2 | Comparison between CISC and RISC Architectures[43]            | 22 |
| 3.1 | Microcontroller Feature Comparison                            | 53 |

# Abbreviations

|         |                                             |
|---------|---------------------------------------------|
| IC      | Integrated Circuit                          |
| CMOS    | Complementary Metal-Oxide-Semiconductor     |
| NMOS    | N-type Metal-Oxide-Semiconductor            |
| PMOS    | P-type Metal-Oxide-Semiconductor            |
| HDL     | Hardware Description Language               |
| VHDL    | VHSIC Hardware Description Language         |
| VHSIC   | Very High-Speed Integrated Circuit          |
| Verilog | Verilog Hardware Description Language       |
| IP      | Intellectual Property (Core)                |
| SoC     | System-on-Chip                              |
| FPGA    | Field-Programmable Gate Array               |
| ASIC    | Application-Specific Integrated Circuit     |
| ALU     | Arithmetic Logic Unit                       |
| CU      | Control Unit                                |
| RAM     | Random Access Memory                        |
| ROM     | Read-Only Memory                            |
| RISC    | Reduced Instruction Set Computer            |
| CISC    | Complex Instruction Set Computer            |
| GPIO    | General-Purpose Input/Output                |
| UART    | Universal Asynchronous Receiver-Transmitter |
| I2C     | Inter-Integrated Circuit                    |
| RTL     | Register Transfer Level                     |
| PDK     | Process Design Kit                          |
| CPU     | Central Processing Unit                     |
| EDA     | Electronic Design Automation                |
| ISA     | Instruction Set Architecture                |

# General introduction

The rapid advancement of semiconductor technology has profoundly reshaped modern electronic systems, enabling the development of integrated circuits (ICs)[1] that are increasingly powerful, compact, and energy-efficient. These circuits, spanning from basic digital logic elements to sophisticated System-on-Chip (SoC) architectures[2], form the cornerstone of contemporary electronic devices. They drive innovation across diverse domains such as computing, telecommunications, industrial automation, and the Internet of Things (IoT)[3][4].

A key driver of this evolution has been Moore's Law[5], which has guided the continuous scaling of semiconductor processes. This scaling has facilitated the integration of billions of transistors onto a single chip, while simultaneously reducing power consumption and manufacturing costs. As a result, digital IC design has emerged as a multidisciplinary field, encompassing computer architecture, digital logic design, hardware description languages, and advanced physical implementation techniques[6].

In this context, microcontrollers play a pivotal role in digital systems by integrating a processor core, memory, and peripheral interfaces onto a single chip. Their adaptability, cost-effectiveness, and low-power characteristics make them indispensable in embedded applications ranging from consumer electronics to industrial automation. The recent emergence of the open-source RISC-V instruction set architecture (ISA)[7] has further expanded opportunities for innovation, offering a flexible and accessible alternative to proprietary processor designs and stimulating progress in both academic and industrial research.

This project focuses on the design and implementation of a RISC-V (Reduced Instruction Set Computer)-based microcontroller using open-source tools[8]. The architecture adopts a modular approach centered on the PicoRV32 processor core, integrating essential peripherals such as GPIO(General-Purpose Input/Output), UART(Universal Asynchronous Receiver-Transmitter), and I<sup>2</sup>C(Inter-Integrated Circuit) communication interfaces, along with memory modules, all interconnected via a memory-mapped bus architecture[9]. The design methodology emphasizes the reuse and integration of Intellectual Property (IP) cores, thereby accelerat-

ing development while ensuring reliability and scalability.

To validate the proposed architecture, a complete digital design flow is employed, encompassing hardware description, simulation, verification, and top-level implementation. The system is implemented using open-source Electronic Design Automation (EDA) tools, demonstrating the practicality of modern IC development in accessible, cost-effective environments. The final implementation is evaluated in terms of performance, silicon area, timing characteristics, and power consumption, using a 130 nm CMOS technology as the fabrication reference.

This manuscript is organized into three comprehensive chapters, each addressing a critical stage in the design and realization of the proposed system. Chapter One establishes the theoretical foundation by presenting the essential principles of integrated circuit design, semiconductor technologies, CMOS circuit operation, and digital logic fundamentals, thereby providing the conceptual framework necessary for understanding modern IC development. Chapter Two transitions to the practical aspects of the project, introducing the architecture and design methodology of the RISC-V microcontroller. It elaborates on the modular organization centered around the PicoRV32 core, the datapath structure, and the integration of peripherals through a memory-mapped bus, while also highlighting the role of open-source tools in enabling efficient and scalable design practices. Chapter Three focuses on the implementation phase, emphasizing the reuse and integration of Intellectual Property (IP) cores, the application of open-source toolchains for synthesis and verification, and the rigorous simulation procedures employed to validate functionality. The chapter culminates with a detailed analysis of experimental results, examining metrics such as area utilization, power consumption, performance efficiency, and timing closure, thereby demonstrating the reliability and scalability of the proposed microcontroller design.

-

# Chapter 1

## Integrated Circuit Design Fundamentals

### 1.1 Introduction:

integrated circuit (IC) design is the foundation of modern electronic systems. It facilitates the integration of a substantial number of transistors onto a single chip, leading to enhancements in performance, reductions in power consumption, and optimization of physical area. This discipline is central to the advancement of digital systems, including microprocessors, microcontrollers, and System-on-Chip (SoC) architectures[2].

This chapter introduces the fundamental concepts of integrated circuit design. It covers the definition and classification of ICs, provides an overview of semiconductor technology and fabrication processes, and details the fundamental electronic components employed within ICs. Furthermore, it discusses Complementary Metal-Oxide-Semiconductor (CMOS) technology, Moore's Law, and the generalized IC design workflow, spanning from initial specification through final verification.

Additionally, the chapter examines digital logic fundamentals and the function of intellectual property (IP) cores. This content establishes the requisite background for comprehending advanced microcontroller and system design concepts presented in subsequent chapters.

### 1.2 Definition of integrated circuits:

An integrated circuit (IC), or more frequently called a microchip or chip, is one of the most important inventions today. The IC is a single piece of semiconductor material with tiny electronic components interconnected (from several million to several billion), such as transistors, resistors, and capacitors. These components are connected to perform specific functions for the

required application. They are connected by a complex network of metallic junctions formed on the chip's surface. The components are manufactured using micro-fabrication techniques on a single-crystal silicon semiconductor chip, usually called silicon wafer, it can be produced with other semiconductors like germanium (Ge) and gallium (Ga)[10].

The integrated circuit (IC) are used today in most areas of life such as smartphones, computers, TV sets[11]. In recent years, the performance of ICs has increased enormously with less power consumption and low cost compared to the number of components in the chip, contributing to unprecedented development.[12] **Figure:1.1**.show the difference between the new generation of ICs and the first developed in 1961.

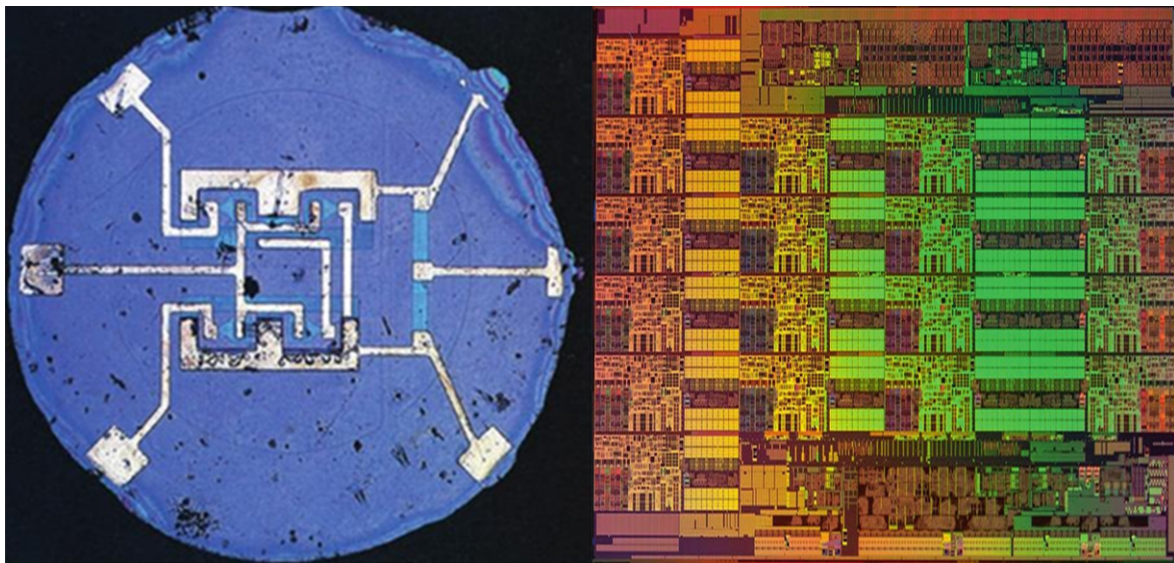


**Figure 1.1:** Old version of an integrated circuit developed in 1961 on the left and the 14 nm-based Intel Xeon E5-2600 V3 18-core processor with 5.69 billion transistors on the right. [13]

## 1.3 Classification of integrated circuits:

Integrated circuits can be classified according to their function, technology, and level of integration.

### 1.3.1 Classification by function:

Integrated circuits can be classified according to the type of signals they can process (analog, digital, and mixed)[12]

### **Analog integrated circuits:**

Analog integrated circuits are electronic circuits that are designed to process continuous signals originating from real-world physical phenomena such as sound, light, temperature, or pressure. The term "analog" reflects the fact that these signals are continuously proportional to the physical quantities they represent. The amplitude of an analog signal may assume any value within a specified range, showing continuous variation over its operating region. That type can amplify or modify the signals for further processing, voltage regulators, analog-to-digital converters (ADCs), and audio amplifiers.[14] Analog circuits are designed as single modules, with a verified layout produced for each such module. The design process itself is only minimally automated; an engineer typically drafts the layout for an analog circuit (and analog parts in mixed-signal designs) almost completely manually.[15]

### **Digital integrated circuits:**

Digital integrated circuits are electronic circuits that are designed to process information represented by discrete signal levels, typically corresponding to binary logic values 0 and 1, also referred to as switching circuits because their input and output voltages are restricted to two distinct levels, commonly denoted as logic high and logic low, and their operation is implemented using logic gates.[14]

Digital values are discrete with respect to time and value; they are less susceptible to disturbances. Furthermore, fewer constraints are needed in the design to ensure its proper functioning, which are easily automated. Thus, digital integrated circuits can be designed using well-understood synthesis algorithms; this can be done with logic synthesis.[15]

Unlike analogue signals, digital signals assume values from a finite set of discrete levels, which enables robust computation, reliable data storage, and efficient information processing. Digital integrated circuits constitute the fundamental building blocks of modern electronic systems, forming the basis of logic gates, microprocessors, digital signal processing units, and memory circuits.[14]

### **Mixed-signal integrated circuits:**

A mixed-signal integrated circuit (ICs) combines analog and digital components on a single semiconductor chip. These circuits provide an interface between the real analog world and digital processing systems by integrating functions such as analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). A mixed-signal integrated circuit leverages the best of both worlds to optimize chip performance, particularly relevant with the proliferation of

smartphones and portable electronic devices. Mixed-signal integrated circuits are becoming increasingly popular.[16]

### **1.3.2 Classification by technology:**

Integrated circuits are primarily classified based on their fabrication technology, which determines their performance characteristics and power efficiency. ICs can be classified into three main technological families[15]:

- Bipolar ICs
- CMOS ICs
- BICMOS ICs

### **1.3.3 Classification by level of integration:**

The following classification outlines the different generations of ICs, from SSI to ULSI, based on the scale of integration and component count per chip:[12]

- SSI (small-scale integration): up to 100 electronic components per chip.
- MSI (medium-scale integration): from 100 to 3,000 electronic components per chip.
- LSI (large-scale integration): from 3,000 to 100,000 electronic components per chip.
- VLSI (very large-scale integration): from 100,000 to 1,000,000 electronic components per chip.
- ULSI (ultra large-scale integration): more than 1 million electronic components per chip.

Beyond integration scale, it is also essential to distinguish between IC types, as shown in the following comparison.

## 1.4 Comparison between ICs types:

The following table **Table1.1** provides a comprehensive comparison between analog and digital integrated circuits (ICs) across various technical parameters.

**Table 1.1:** Comparison between analog and digital integrated circuits[17]

| Aspect            | Analog IC                                       | Digital IC                                           |
|-------------------|-------------------------------------------------|------------------------------------------------------|
| Signal Type       | Continuous signals                              | Discrete signals (binary)                            |
| Processing        | Processes real world, variable data             | Processes binary data                                |
| Complexity        | Simpler design                                  | Higher complexity in circuits                        |
| Noise Sensitivity | More vulnerable to interference and distortions | More resistant to interference and noise disruptions |
| Applications      | Audio, sensors, signal conditioning             | Computing, storage, digital communication            |
| Power Consumption | Generally higher for amplifiers                 | Lower for logic circuits                             |
| Accuracy          | Depends on precision of components              | High for binary operations                           |
| Speed             | Limited by signal amplitude and frequency       | Faster due to digital switching                      |

## 1.5 Integrated circuits design:

Integrated circuits (ICs) design is a process of planning, simulating, designing and interconnecting circuit elements to perform a specific desired function. The ICs design process typically involves multiple stages, including system specification and requirements, architectural design, logic design, physical layout design, and verification, ensuring that the final chip meets performance, reliability, and manufacturability requirements.[11]

## 1.6 Importance of ICs design:

Integrated circuits design enables:[18]

- Reduction in the size of electronic systems.
- High-speed data processing.
- Low power consumption.
- High reliability.

- Improved performance.
- Reduced manufacturing cost.

## 1.7 Semiconductor technology overview:

### 1.7.1 Introduction to semiconductor:

Semiconductors are materials that present unique electronic properties that can work as a conductor or insulator of electricity. We can use an operation called doping (N-type and P-type) with electric fields and temperature variations to control the conductivity.[19] There are two types of semiconductors: elemental and compound semiconductors. The first one can be found in group IV of the periodic table as show in **figure1.2**, such as silicon (Si) and germanium (Ge), while the second one is formed from combinations of materials from groups III and V (III-V semiconductors) or II and VI (II-VI semiconductors) of the periodic table. Semiconductors is an important substance for the new technology as building block of computers, electronic devices such as diodes, transistors, and integrated circuits and integrated.[20]

| Period | Column II       | III            | IV              | V               | VI              |
|--------|-----------------|----------------|-----------------|-----------------|-----------------|
| 2      |                 | B<br>Boron     | C<br>Carbon     | N<br>Nitrogen   |                 |
| 3      | Mg<br>Magnesium | Al<br>Aluminum | Si<br>Silicon   | P<br>Phosphorus | S<br>Sulfur     |
| 4      | Zn<br>Zinc      | Ga<br>Gallium  | Ge<br>Germanium | As<br>Arsenic   | Se<br>Selenium  |
| 5      | Cd<br>Cadmium   | In<br>Indium   | Sn<br>Tin       | Sb<br>Antimony  | Te<br>Tellurium |
| 6      | Hg<br>Mercury   |                | Pb<br>Lead      |                 |                 |

**Figure 1.2:** Portion of the periodic table related to semiconductors[21]

Building on the elemental classification shown above, the following section details semiconductor fabrication technology, starting with the principles of microtechnology, nanotechnology and miniaturization.

## 1.8 Semiconductor fabrication technology:

Modern semiconductor manufacturing technology lies in mastering small-scale engineering, particularly microtechnology and nanotechnology. The following paragraph presents manufacturing technology at the microscale.

### **1.8.1 Microtechnology:**

'Microtechnology' refers to the application of technological processes to create miniaturised objects or systems at the microscale. It involves engineering structures at scales close to 1 micron ( $\mu\text{m}$ ), which is equivalent to  $10^{-6}$  m or  $10^3$  nm, and even 1000 nm in the nanoscale equivalent. At this scale, microtechnology enables the precise control and development of small-scale devices, structures, and components. Microtechnology is essential for advancing areas like microelectronics, sensors, and microelectromechanical systems (MEMS), where miniaturization allows for more compact, efficient, and innovative technological solutions.[22] While microtechnology focuses on the micrometer scale, further miniaturization leads us into the realm of nanotechnology, where matter behaves at an even smaller level

### **1.8.2 Nanotechnology:**

Nanotechnology refers to the science and engineering of materials, devices, and systems on the nanometer scale between 1 and 100 nm, where unique properties enable innovative applications across diverse fields, including chemistry, physics, biology, medicine, engineering, and electronics. Nanotechnology is the application of this knowledge in the development of practical tools, devices, and systems that operate on this scale. This branch of science explores how matter behaves differently at this scale, leading to advances in the creation of new materials, devices, and systems with enhanced or fundamentally new chemical, physical, and biological functionalities. This precise control at the nanoscale enables the development of materials and systems with unique and improved functionalities.[22] These advanced fabrication scales and techniques are essential for constructing fundamental semiconductor components, the most basic of which is the PN junction.

## **1.9 Semiconductor components:**

The interaction of variously doped materials allows electronic components to carry out their specific functions.

### **1.9.1 PN Junction:**

A PN junction is formed when two regions of the same semiconductor material are doped differently, one doped with acceptor impurity atoms to form the p region, and another doped with donor atoms to form the n region. The boundary between the n and p regions forms

a metallurgical junction, there is a large difference in carrier concentration. Because of this difference, electrons from the n region diffuse into the p region, and holes from the p region diffuse into the n region. As electrons and holes move across the junction, they leave behind fixed charged ions: positive donor ions remain in the n region and negative acceptor ions remain in the p region. These fixed charges create a region near the junction called the depletion region (or space charge region). In this region, there are almost no free charge carriers. The charged ions in the depletion region generate an internal electric field, directed from the n region to the p region. This electric field is strongest at the junction and becomes zero outside the depletion region and the presence of this electric field creates a built-in potential difference across the junction, which opposes further diffusion of charge carriers. Numerous electrical components, including the diode, are based on the PN junction. As seen in **figure:1.3**, a diode is essentially a PN junction contained in a protective packaging. [23]

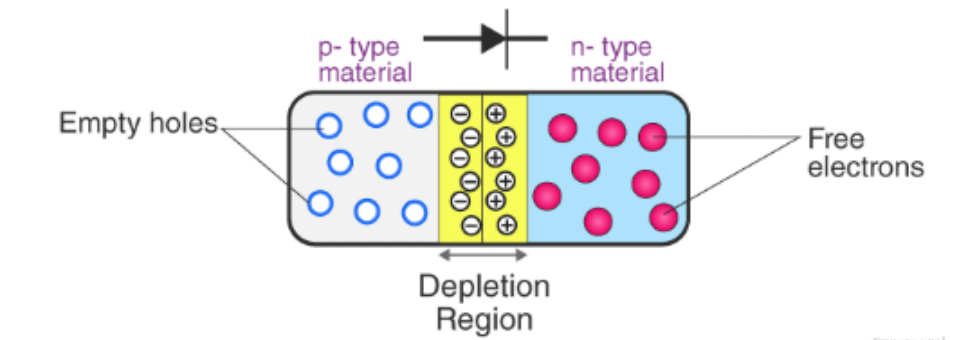


**Figure 1.3:** P-N Junction Diode[24]

Transistors, which are particularly relevant to our investigation, are among the other electronic components that rely on the PN junction.

## 1.9.2 Transistors:

A semiconductor diode is certainly useful in electronic circuits, but it is still a fairly passive device. It does not amplify the signal. The semiconductor component that really revolutionized the world of electronics is the transistor.[25] Transistors can be viewed as electrically controlled switches with a control terminal and two other terminals that are connected or disconnected depending on the voltage or current applied to the control[26] a diode has just a single PN junction and two leads, the transistor (usually) has two PN junctions and three leads (Collector, Base, Emitter). Transistors do exhibit gain and are true active devices. The name "transistor" comes from transfer resistor.[25] there too many components developed from transistors some of this types show in **figure:1.4**

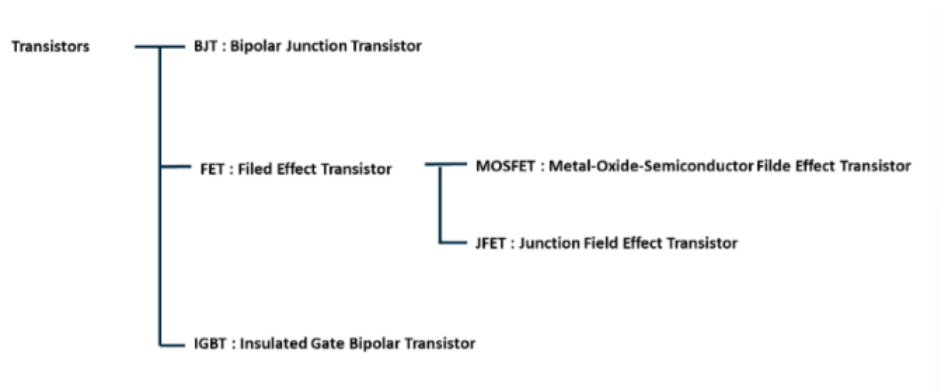


**Figure 1.4:** categorized of transistors based on element structure and operation principles.[27]

## 1.10 CMOS Technology:

Complementary metal-oxide-semiconductor (CMOS) technology is fabricated by integrating NMOS and PMOS transistors on a single silicon substrate. It is derived from PMOS and NMOS processes, combining their complementary properties. CMOS integrated circuits are widely used due to their very low power consumption, high efficiency, and strong interference resistance, resulting from a wide logic voltage range. They also generate less heat, allow for very high integration density, reduce manufacturing costs, and offer good thermal stability and radiation resistance. Furthermore, CMOS circuits offer high operating capability and reliability across a wide temperature range. However, CMOS technology has some drawbacks, most notably a more complex manufacturing process because NMOS and PMOS devices must be fabricated on the same substrate. This increases processing requirements and silicon surface area, and may result in relatively slower operation compared to some single-type MOS technologies in certain applications.[28]

## 1.11 Moore's law and technology scaling:

In 1965 Gordon E. Moore, Intel cofounder observed that the number of components on an integrated circuit doubles approximately every year as show in **figure:1.5**. He predicted that this exponential growth would continue for at least a decade, leading to increased performance twice (performance means the speed of operation in terms of clock frequency) that makes the operations faster requiring lesser time, reduced cost per component, and reducing the chip areas.[5][6]

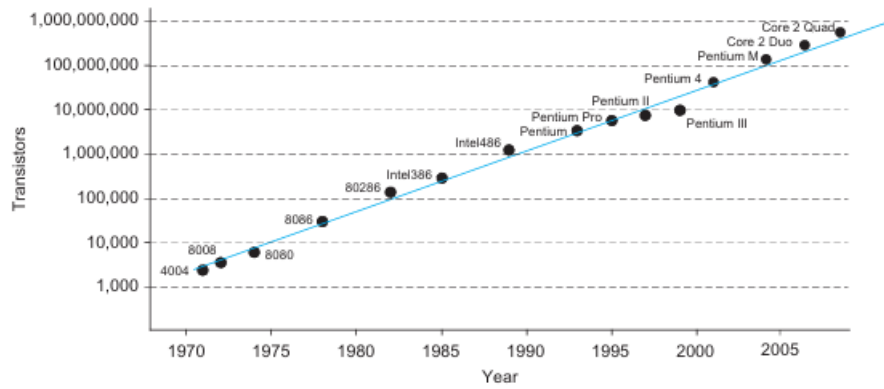


**Figure 1.5:** Transistors in Intel microprocessors.[29]

## 1.12 General ICs design methodology:

There are many steps involved in the ICs Design as shown in **figure:1.6**:

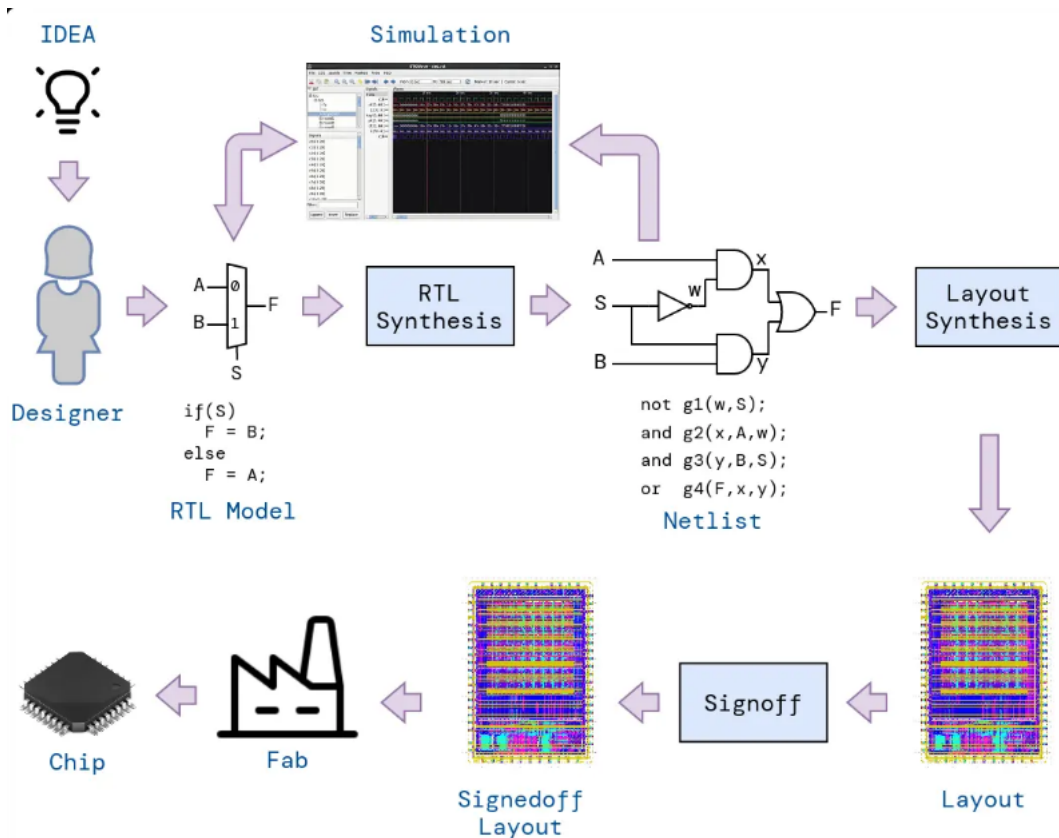


**Figure 1.6:** IC design flow[30]

### **1.12.1 System specification and architectural design:**

This step documents all the primary requirements for developing the ICs. This means that everything required for the ICs design is stated clearly in this phase:[11]

- the purpose of the ICs to be designed.
- Functional requirements.
- Performance (frequency, power, speed, area).
- Target technology node (e.g., 7nm, 28nm).
- The targeted cost of the system.

### **1.12.2 Logic design:**

The circuit design stage translates architectural behavior into concrete hardware logic at the RTL level. Once the integrated circuit concept is defined, the high-level requirements are decomposed into the lower-level building blocks. Existing blocks/macros are used or modified to achieve the desired functionality. These blocks are simulations to validate the desired functionality.

### **1.12.3 Physical design:**

At this stage, the layout of the interconnected shapes of the required circuit components is created. This step begins with a basic schematic of the floorplan to determine the location, shape, and size of modules in a chip. This schematic facilitates the analysis of available space on the chip, latency, congestion, and other factors to optimize component placement and mitigate issues that could negatively impact performance and speed. Within each floorplan region, transistors are individually positioned and then connected by wires to link them, ensuring the implementation of the required functions. The architectural design must be reviewed after each stage of design and implementation phase.[11]

### **1.12.4 Verification and testing:**

In this step, this process refines the design and allows for the modeling of physical effects (resistance, interference, etc.) likely to occur during manufacturing. Here, the design rules are

created by checking the physical effects caused due to manufacturing process. The designers take into account the additional resistance due to wiring, signal interference, and other changes that may occur during manufacturing.[11]

#### **1.12.5 The signoff:**

In the final signoff stage, the essential parameters of the task are verified before manufacturing begins. Such parameters include timing, power consumption, and signal integrity. These parameters control aims to minimize their impact on performance and the manufacturing process. Once all the necessary steps have been completed, the integrated circuit is sent for manufacturing.[11]

### **1.13 Digital logic fundamentals:**

Digital electronics is the most common application of Boolean algebra and forms the basis of all digital circuits in computers, mobile phones, and many other consumer products. The most common basic unit in digital electronics is the logic gate. By combining multiple logic gates, more complex systems can be created. All these complex digital electronic systems are called digital circuits. Binary, hexadecimal, and octal refer to different number systems that we can use, the binary system is the one we use in the digital electronics. There are three basis binary operations, AND, OR and NOT that called Boolean algebra or switching algebra. These operations can be represented by something called the logic gates expressed by symbols show in **figure:1.7**. [31]

| GATE  | SYMBOL                                                                            | NOTATION               | TRUTH TABLE                                                                                                                                                                                                                                                     |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
|-------|-----------------------------------------------------------------------------------|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|--------|---|-------|----------|---|---|---|---|---|---|---|---|---|---|---|---|
| AND   |  | $A \cdot B$            | <table><tr><th>Input</th><th>Output</th></tr><tr><th>A</th><th>B</th><th>A AND B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>  | Input | Output | A | B     | A AND B  | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| Input | Output                                                                            |                        |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| A     | B                                                                                 | A AND B                |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 0                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 1                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 0                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 1                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| OR    |  | $A + B$                | <table><tr><th>Input</th><th>Output</th></tr><tr><th>A</th><th>B</th><th>A OR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>   | Input | Output | A | B     | A OR B   | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| Input | Output                                                                            |                        |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| A     | B                                                                                 | A OR B                 |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 0                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 1                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 0                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 1                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| NOT   |  | $\overline{A}$         | <table><tr><th>Input</th><th>Output</th></tr><tr><th>A</th><th>NOT A</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>                                                                                                                | Input | Output | A | NOT A | 0        | 1 | 1 | 0 |   |   |   |   |   |   |   |   |   |
| Input | Output                                                                            |                        |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| A     | NOT A                                                                             |                        |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 1                                                                                 |                        |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 0                                                                                 |                        |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| NAND  |  | $\overline{A \cdot B}$ | <table><tr><th>Input</th><th>Output</th></tr><tr><th>A</th><th>B</th><th>A NAND B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table> | Input | Output | A | B     | A NAND B | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| Input | Output                                                                            |                        |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| A     | B                                                                                 | A NAND B               |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 0                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 1                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 0                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 1                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| NOR   |  | $\overline{A + B}$     | <table><tr><th>Input</th><th>Output</th></tr><tr><th>A</th><th>B</th><th>A NOR B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>  | Input | Output | A | B     | A NOR B  | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| Input | Output                                                                            |                        |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| A     | B                                                                                 | A NOR B                |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 0                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 1                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 0                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 1                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| XOR   |  | $A \oplus B$           | <table><tr><th>Input</th><th>Output</th></tr><tr><th>A</th><th>B</th><th>A XOR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>  | Input | Output | A | B     | A XOR B  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| Input | Output                                                                            |                        |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| A     | B                                                                                 | A XOR B                |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 0                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 0     | 1                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 0                                                                                 | 1                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |
| 1     | 1                                                                                 | 0                      |                                                                                                                                                                                                                                                                 |       |        |   |       |          |   |   |   |   |   |   |   |   |   |   |   |   |

Figure 1.7: Logic gates.[31]

### 1.13.1 Combinational logic:

Combinational logic is a kind of digital circuit. What comes out of it depends only on what's going into it right then. As soon as the inputs shift, it completely loses any information about what the previous inputs were. So, these kinds of circuits really don't have a memory. They're built from basic logic gates, like AND, OR, NOT, and XOR.[32]

### 1.13.2 Sequential logic:

Sequential logic is a type of digital circuit where the output relies on current inputs, AND previous states (memory). The main parts we use to build sequential circuits are called registers and latches[33].

## 1.14 Intellectual property core :

An IP core or Intellectual Property Core represents a form of semiconductor intellectual property, typically instantiated as a reusable hardware description language (HDL and VHDL) component. Its application spans across various hardware design paradigms, including field-programmable gate arrays (FPGAs), programmable system-on-chips (SoCs), and application-

specific integrated circuits (ASICs).

Within the context of FPGAs and programmable SoCs, IP cores serve as foundational building blocks. These components can be integrated into comprehensive system implementations through the deployment of specialized design tools, exemplified by the Vivado IP Integrator developed by AMD.[34]

## 1.15 From logic gates to processor systems:

We are looking at how digital electronics are put together, starting from the smallest bits all the way to a full working computer. The foundational elements comprise semiconductor components such as transistors and Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs), fabricated through micro and nanotechnology processes. Next, we have logic gates, built from these transistors (specifically MOSFETs in CMOS technology). Each gate performs a specific Boolean function, processing binary inputs represented as logical 0 or 1. The interconnection of multiple logic gates can create what we call combinational circuits, like Adders, Multiplexers (or MUX), Decoders, and Encoders. The integration of memory elements transforms a purely combinational system into a sequential circuit. The main parts for Sequential Circuits are Latches and Flip-Flops (D, JK, and T types). A collection of interconnected flip-flops, configured to store digital data, constitutes a register. By combining parts like registers, a Control Unit, a Memory System, and a Bus System (which are all built from logic gates and transistors), enables the realization of a Central Processing Unit (CPU). From there, we can even make a microcontroller. **figure:1.8** represents all of these details

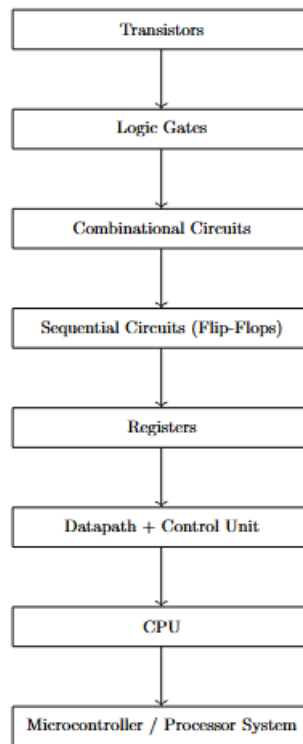


**Figure 1.8:** The steps to made the microccontroller.

## 1.16 Conclusion:

This chapter presented the fundamental concepts of integrated circuit design, establishing the requisite background for understanding modern digital systems. It defined and classified integrated circuits, detailing the key fabrication technologies, particularly semiconductor processes and CMOS technology.

The chapter also outlined the general IC design methodology, spanning from system specification to verification. It emphasized the critical role of digital logic fundamentals and IP cores in complex system design. Furthermore, the discussion addressed how advancements, such as Moore's Law, have driven continuous improvements in integration density and performance.

Collectively, this chapter establishes the theoretical foundation essential for the design and implementation of the RISC-based microcontroller detailed in subsequent chapters.

-

# **Chapter 2**

## **Design and architecture of a RISC-V-based microcontroller**

### **2.1 Introduction:**

This chapter presents the design and architecture of a RISC-based microcontroller. It introduces fundamental microcontroller concepts and compares them with microprocessors, emphasizing the advantages that lead to their selection in embedded system design. It further discusses CPU architectures, specifically RISC and CISC principles, and elucidates the rationale for adopting a RISC-based approach[35]. Furthermore, it details the main architectural blocks of a microcontroller, comprising the datapath, control unit, memory, and peripheral interfaces. Subsequently, the architecture of the proposed system, based on the PicoRV32 core, is described, alongside its key components such as the system bus, General Purpose Input/Output (GPIO), Universal Asynchronous Receiver/Transmitter (UART), and Inter-Integrated Circuit (I2C) modules. Finally, the chapter elucidates the complete design flow, spanning from system specification through implementation and testing, and addresses relevant design considerations and identified limitations.

### **2.2 Microcontroller:**

A Microcontroller is a digital electronic component with miniaturized transistors in a single semiconductor Integrated Circuit (IC).[36]In our digital age, microcontrollers play a crucial role in powering various electronic devices, doing the heart job on too many devices that we use everyday, from smartphones to supercomputers[37].

A microcontroller capable of performing ALU (Arithmetic Logical Unit) operations include adding, subtracting, comparing two numbers, fetching numbers from one area to another and communicating with the other devices connected to it.[36] [38]Determining and simulating the designs of these devices requires complex techniques. Using VHDL (VHSIC Hardware Description Language), which makes it possible to precisely model, simulate, and synthesize Microcontroller architectures. There are essential elements of a Microcontroller modeled in VHDL as Control Unit (CU), Arithmetic Logic Unit (ALU), Registers, Data Bus and Address Bus, Memory Interface as (RAM, ROM), Input and output ports to function.[39].

Microcontroller are typically constructed in the proximity of two major computer architectures: Complex Instruction Set Computing (CISC) and Reduced Instruction Set Computing (RISC)[40]. **figure:2.1**

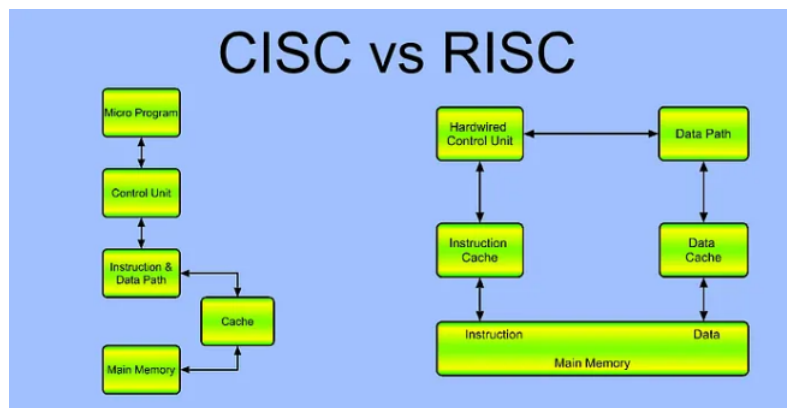


**Figure 2.1:** Difference Between CISC and RISC [41]

## 2.3 Microprocessor vs Microcontroller:

There are a number of differences between a microcontroller and a microprocessor . The main difference is that the microprocessor is required to add other peripherals such as read/write memory, read only memory (ROM), decoders, drivers, and the number of I/O devices to make a complete microcomputer system. In Addition to this, some special-purpose devices, for example, interrupt controllers, programmable timers, programmable I/O devices. The Microcontroller has integrated random access memory, Read Only Memory, parallel I/O, serial I/O, a clock circuit, and counters. More differences in the **Table:2.1**.

**Table 2.1:** Comparison Between Microprocessor and Microcontroller[42]

| Sr. No. | Microprocessor                                                                                                                                        | Microcontroller                                                                                                                                                   |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1       | Microprocessor includes Arithmetic Logic Unit, general-purpose register, program counter, interrupt circuit , stack pointer and clock timing circuit. | Microcontroller includes the circuitry of microprocessor and additionally has built in Random Access Memory, counters, I/O devices, Read Only Memory, and timers. |
| 2       | It has several instructions to transfer content between CPU and memory.                                                                               | It has few instructions to transfer content between CPU and memory.                                                                                               |
| 3       | It has 1- or 2-bit handling commands.                                                                                                                 | It has several bit handling commands.                                                                                                                             |
| 4       | Access time for I/O devices and memory is large.                                                                                                      | Access time is less for built-in memory and input and output components.                                                                                          |
| 5       | Hardware requirement is more for system based on microprocessor.                                                                                      | System based on Microcontroller needs less hardware reducing PCB size and increasing the dependability.                                                           |
| 6       | It is more flexible while consider designing.                                                                                                         | It is less flexible while considering designing.                                                                                                                  |
| 7       | It has single memory map for data and code.                                                                                                           | It has separate memory map for data and code.                                                                                                                     |
| 8       | The number of pins of multifunction are less                                                                                                          | Most of pins are multi-functioned.                                                                                                                                |

## 2.4 Justification for choosing the microcontroller:

The choice of a microcontroller for this project is driven by the need for a small-scale, efficient, self-sufficient, and easy-to-implement system that can be deployed within a limited timeframe by someone with no experience in the field of the ICs design.

Unlike a microprocessor, which requires external components, a microcontroller relies on a system fully integrated into a single chip. This allows us to build an architecture suitable for our requirements, relying on readily available components, and focusing our work on the architecture and the connections between these components, both inside and outside the chip.

## 2.5 CPU architecture:

There are two classifications of processors: complex instruction set computer (CISC) and the reduced instruction set computer (RISC). In reality, there are a wide range of architectures and

these architectures exist in the spectrum ranging from completely CISC to completely RISC. Examples of CISC include Intel x86 . Examples of RISC include MIPS, AVR (Atmel) , PowerPC (IBM), and ARM.[43]

### **2.5.1 RISC architecture:**

The Reduced Instruction Set Computer (RISC) architecture has long been used in many industrial applications. Unlike the CISC (Compound Instruction Set Computer) architecture, which has a large and complex basic instruction set, this architecture is characterized by a smaller number of basic instructions. A key feature of the RISC model is its load/store architecture; this design principle allows for reduced instruction widths and maintains a uniform instruction length within the set. An RISC processor typically achieves an average data transfer rate of about one instruction per clock cycle. Furthermore, this architecture allows any register to function as either a source or a destination. The RISC architecture is often derived from the Harvard architecture, which defines separate memory areas for program code and data. This separation allows simultaneous access to both memory areas, thus improving system performance.[44]The instruction set is designed to allow programmable control of operating conditions and storage memory. The advantage of the RISC system lies in its ability to implement the acquisition of analog and digital data.[7][45]

### **2.5.2 CISC architecture:**

The Complex Instruction Set Computer (CISC) represents a processor architecture characterized by an extensive array of instructions. Although typically incorporating complex instructions, certain CISC implementations may be optimized for low-power operation. A defining characteristic is its ability to access memory directly in a single instruction. Indeed, a single CISC instruction can often perform both memory read and write operations, utilizing a variety of addressing modes[35] [43] . **Table:2.2.** may display other differences.

**Table 2.2:** Comparison between CISC and RISC Architectures[43]

| <b>CISC</b>                                                                                       | <b>RISC</b>                                                               |
|---------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| Many instructions                                                                                 | Few instructions                                                          |
| Instructions have varying lengths                                                                 | Instructions have fixed lengths                                           |
| Instructions execute in varying times                                                             | Instructions execute in 1 or 2 bus cycles                                 |
| Many instructions can access memory                                                               | Few instructions (e.g., load and store) can access memory                 |
| In one instruction, the processor can both read and write memory                                  | No one instruction can both read and write memory in the same instruction |
| Fewer and more specialized registers. E.g., some registers contain data, others contain addresses | Many identical general-purpose registers                                  |
| Many different types of addressing modes                                                          | Limited number of addressing modes (e.g., register, immediate, indexed)   |

## 2.6 Justification for choosing the RISC architecture:

After evaluating various architectural models, we selected the RISC (Reduced Instruction Set Computer) architecture. This choice is driven by its simplified hardware design and a streamlined instruction set consisting of fewer, more efficient commands. By utilizing a Load/Store model, RISC allows for high-speed operations within a large register file. Furthermore, when implemented with a Harvard Architecture, it enables simultaneous access to both program memory and data memory. This significantly shortens execution time by minimizing the clock cycles per instruction (CPI), offering superior performance and efficiency for our project requirements compared to CISC systems[35].

## 2.7 Datapath:

In computer engineering, the data path is an essential part of the processor(CPU),which is responsible for executing instructions through data processing and transmission. This architectural element comprises several key units, including registers, arithmetic logic units (ALUs), multipliers, and internal busses, all controlled by signals from a dedicated controller[46]. The controller manages the data flow, sending commands to all components, and ensuring optimal operation. It receives information from the data flow and emits signals to control its operation, such as multiplier selection signals, register activation signals, and data write signals. [47]

## 2.8 Types of datapath designs:

### 2.8.1 Single-cycle datapath:

Each instruction is executed in a single clock cycle shown in **figure:2.2**, with all processing steps, from fetching to writing the instruction, performed simultaneously. Although this design is relatively simple to implement due to the absence of auxiliary registers or complex control mechanisms, it has a major drawback. This drawback primarily comes from the clock cycle length, which prevents interference during instruction execution.[46]

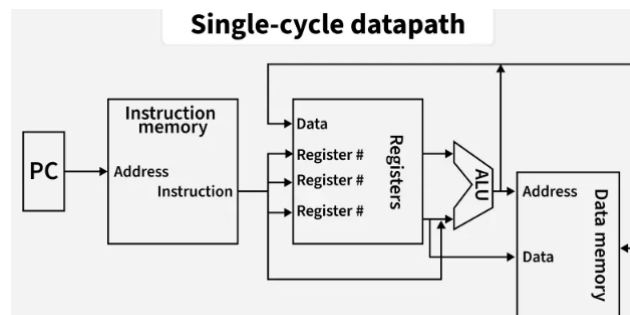


Figure 2.2: Single-Cycle Datapath[46]

### 2.8.2 Multi-Cycle Datapath:

Instructions are typically decomposed into a series of distinct operational steps, each of which is executed within a single clock cycle. This architectural design as shown in **figure:2.3**, while demanding more intricate control logic, contributes to enhanced operational efficiency. Specifically, an individual instruction progresses through multiple short cycles, utilizing additional registers positioned between these stages. In this configuration, only one instruction is processed at any given moment, ensuring that there is no temporal overlap in execution. This method offers improved efficiency over a single-cycle implementation.[46]

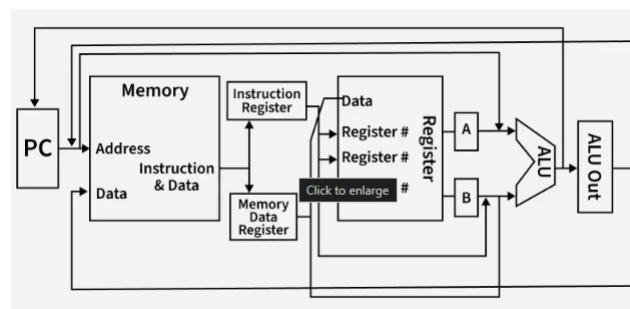


Figure 2.3: Multi-Cycle Datapath[46]

### 2.8.3 Pipelined datapath:

Instruction execution can also be divided into a series of fixed stages, as shown in **figure:2.4**, a design that facilitates the concurrent processing of multiple instructions. This architectural model significantly improves system throughput, although it inherently increases complexity. In such a segmented architecture, multiple instructions are executed in parallel, each occupying a specific stage of the pipeline. This approach relies on short clock cycles and the strategic placement of additional registers between pipeline stages. Although it achieves high performance, this method requires robust logic to detect and resolve various unexpected problems that may occur during execution.[46]

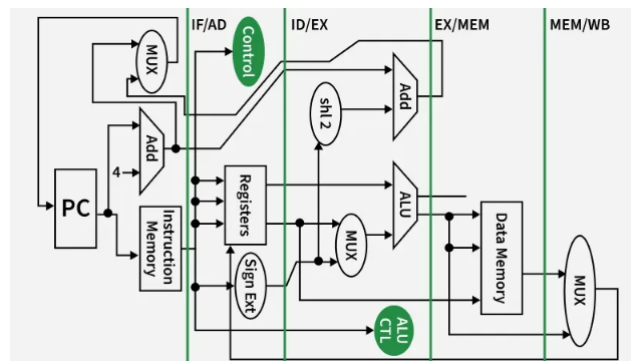


**Figure 2.4:** Pipelined-cycle datapath[46]

## 2.9 Microcontroller components:

A microcontroller is a compact integrated circuit designed to perform specific control tasks in embedded systems. It combines of some components as:

### 2.9.1 Control unit(CU):

The control unit, often called the CU, is a crucial part of a computer's central processing unit, or CPU. It primarily handles the management and coordination of how the processor operates, guiding where data and instructions go within the system. The CU works by taking program instructions and translating them into signals. These signals tell other components inside the processor and connected devices what action to take, so the CU is so important to the work of the processor and the overall performance of the computer.[48] The control unit acts as the brain of the CPU. It sends out control signals to manage all the different parts inside the core. The CU also handles tasks like reading and writing data, and these instructions work with both the registers and the memory. It's also responsible for sending out signals that help decide when

and how the ALU and other processing elements should be used. Lastly, the CU creates specific flags for program branches, which the BU then uses.[49]

### **2.9.2 Arithmetic and logic unit (ALU):**

The main function of any processor system is to process data, for example, to add two numbers together. The arithmetic logic unit (ALU) contains hardware components capable of performing arithmetic operations by taking, in general, two data words as inputs AND combining them together[50] (addition, subtraction, multiplication, binary complement), logical operations (AND, XOR, OR), and bit shifts. The functions available in the ALU determine the types of instructions a computer can execute.[51] The ALU takes results from the processor core. These operation results usually involve one or two operands. The values for these operands are pulled from a register, from the main memory, or sometimes the instruction itself gives an immediate value directly.[49]

### **2.9.3 Registers:**

Registers are fundamental storage elements embedded within a microcontroller (MCU). They function as temporary repositories within the central processing unit (CPU), enabling the retention of data and instructions essential for hardware control and monitoring.

Unlike main memory, which cannot directly support arithmetic or logical operations, registers provide the necessary interface for the processor to manipulate data—such as performing addition, subtraction, or other computations—while ensuring rapid access and temporary storage. Microcontrollers incorporate various categories of registers, typically classified according to the type of information they store or the operations they facilitate. The most common categories include:

Data Registers: Hold values actively processed by the CPU.

- Control Registers: Govern the execution of instructions and regulate system operations.
- Status Registers: Reflect the outcome of operations and indicate the current state of the processor.
- Configuration Registers: Define operational parameters and customize the behavior of the MCU..[52][53].

## 2.9.4 Memory:

Memory constitutes the fundamental storage medium in computing systems, where information is represented in binary form in modern technologies. Two principal categories of memory are distinguished: volatile and non-volatile. The defining characteristic between them lies in data persistence following a loss of power. In volatile memory, all stored information is erased once power is removed, whereas non-volatile memory retains data even after power is cut.

Compared to general-purpose computers such as desktops, laptops, tablets, or smartphones which are equipped with vast memory resources microcontrollers typically operate with significantly more limited storage capacity. This constraint necessitates careful consideration of program size and memory usage during software development for embedded systems.

Among the most common types of memory encountered are:

- RAM (Random Access Memory): provides temporary high-speed storage for data actively processed by the microcontroller.
- ROM (Read-Only Memory): Contains permanent instructions or data, often used to store firmware essential for system operation.[51]

## 2.10 The Architecture of the proposed microcontroller:

In the proposed microcontroller architecture, each component has been deliberately selected to achieve an optimal balance between simplicity, performance, and ease of implementation. While numerous ready-to-use architectures already exist, the design process revealed the opportunity to develop a new architecture tailored to specific requirements—one that is straightforward, practical to construct, and innovative.

Accordingly, the architecture was conceived and structured to meet these objectives, as illustrated in **figure2.5**.

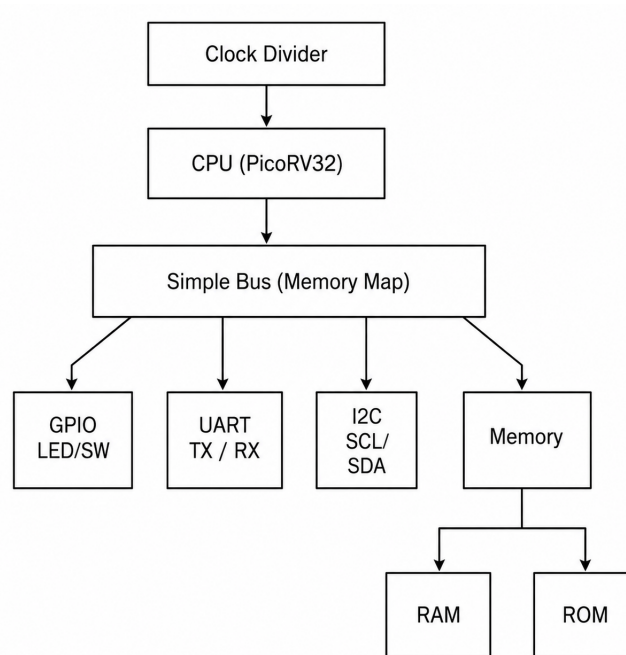


**Figure 2.5:** The Architecture used for the Microcontroller

### 2.10.1 Clock system:

A clock divider or frequency divider, as evident from the name it is a circuit that reduce the input clock signal frequency and generates a lower frequency output clock. By reducing the frequency of an input clock signal that enable the system to operate at slower, manageable frequencies.[54]

### 2.10.2 PicoRV32:

When selecting a microcontroller for use in an embedded system, it is essential to ensure that the chosen device satisfies all project requirements and integrates the necessary peripherals and accessories to support product development. Several key considerations guide this decision, including:

- Processing speed;
- Power consumption;
- Memory type and format;
- Memory capacity;
- Peripheral features; (e.g., timers/counters, ADCs, communication interfaces)

- Number of input/output (I/O) pins;
- Overall cost;

Following a thorough evaluation of the established criteria, the PICOVR32 microcontroller was determined to be the most appropriate choice. It successfully fulfills the specified requirements while maintaining an effective balance between functionality, operational efficiency, and cost-effectiveness.[51]

The PicoRV32 is an open-source, 32-bit CPU core written in Verilog, implementing the RISC-V RV32IMC instruction set. It is optimized for resource efficiency, making it highly suitable for FPGA and ASIC designs in embedded and low-power applications. Its multi-cycle architecture reduces hardware size and complexity, though with some trade-off in performance.

PicoRV32 has become widely used in research, education, and lightweight SoC design, where little hardware overhead and adaptability are crucial, because of its simplicity, flexibility, and efficiency.[55][56]

### **2.10.3 System bus (Memory-mapped bus):**

Instead of using more complex systems like AXI or Wishbone, we opted for a memory-mapped bus architecture. This decision allowed us to utilise the same fundamental load/store instructions for all of our devices, kept the design simple, and guaranteed that it operated flawlessly with the CPU.

The CPU's internal rules for managing input and output are contained in the I/O control section, commonly referred to as the bus interface unit. The primary communication channel between the processor and other parts is the data bus, which this device makes sure information flows through. The bus interface unit is responsible for responding to every signal that enters the processor and generating every signal that leaves the processor and travels throughout the system. The input/output lines that make up the control bus allow the controller to communicate with the outside world. Typically, these actions are also managed via the control bus.[57]

### **2.10.4 Memory:**

In embedded systems, both RAM (Random Access Memory) and ROM (Read-Only Memory) are used to perform distinct functions. RAM serves as a temporary storage medium during program execution. It is volatile, meaning that its contents are erased once power is removed. By contrast, ROM provides permanent storage for the program instructions executed by the

CPU. Being non-volatile- it retains data even in the absence of power, thereby preserving the essential firmware and system programs required for operation.[51]

### **2.10.5 GPIO (General Purpose Input/Output):**

GPIO pins are essential components of microcontrollers, providing a simple interface to interact with external digital devices that allow seamless interaction between the microcontroller and the external world.[58] It provides dedicated General Purpose ports that can be configured as inputs or outputs. When configured as an output, you can write to an internal register to control the state driven on the output pin. When configured as an input, you can detect the state of the input by reading the state of an internal register.[59]

### **2.10.6 UART (Universal Asynchronous Receiver/Transmitter):**

UART (Universal Asynchronous Receiver-Transmitter) is a simple and widely used hardware interface that enables asynchronous serial communication between a microcontroller and external devices. It is often integrated into embedded systems to handle data transmission and reception, using start and stop bits to frame data without requiring a shared clock. Thanks to its reliability and ease of implementation, UART is essential in industrial automation, robotics, and IoT applications where efficient device-to-device communication is needed.[60] does not require a shared clock signal between the transmitting and receiving devices. Instead, it relies on start and stop bits to synchronize the data transmission. [60]

### **2.10.7 I2C (Inter-Integrated Circuit):**

Communication between microcontrollers and different peripheral devices requires some sort of digital protocol.I2C is a way for different parts of a system to communicate using just two wires. One wire handles the data itself (it's called SDA), and the other acts as a clock to keep everything synchronized (called SCL). This setup lets you connect several devices to the same communication line. You can even have multiple main devices, or 'controllers,' sending and getting instructions and information. When data is sent, it's done in small pieces, often called byte packets, and each device on the line has its own specific address.[61]. It is a widely used protocol for short distance communication.

## 2.11 Datapath architecture of the proposed microcontroller:

### 2.11.1 Global Datapath Overview:

The main way data moves in this microcontroller we're talking about centers around a central processing unit (CPU), which is designed using the PicoRV32 RISC-V core. This CPU communicates with memory and other components, like peripherals, through a simple bus system that relies on memory binding. The datapath that we build is shown in **figure 2.6**.

At the top level, the system includes the following key components:

- Read-only memory (ROM), used for instruction storage.
- Random access memory (RAM), used for data storage.
- Peripheral components such as general-purpose input/output (GPIO) ports and a UART interface.
- A central system bus that enables communication between all components.
- The PicoRV32 CPU core.

The CPU gets its instructions from the ROM, which sends them over the system bus. Once the CPU has these instructions, it processes them and uses that same bus to communicate with RAM and other connected devices. It manages this communication through a method called memory-mapped addressing. For interacting with outside parts, like LEDs and switches, there's a GPIO module. And when it comes to sending or receiving data one bit at a time, the UART module handles that serial communication.

This whole arrangement allows data to flow simply and effectively between the processing unit, the memory, and the input/output components. That's why it's a good fit for lightweight embedded applications.

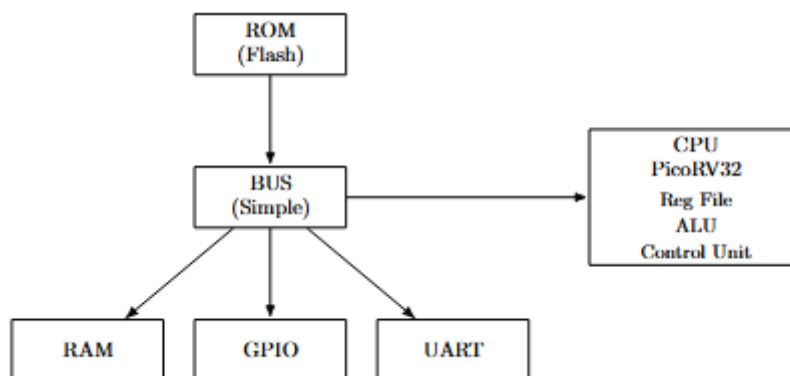


**Figure 2.6:** Global Datapath Overview

### **2.11.2 Justification for choosing a single-cycle datapath:**

We go with a single-cycle datapath mainly because it's simple and predictable. The idea is that each instruction gets done in just one clock cycle. Because of this, you don't have to worry about complex control systems, things like pipelining or handling tricky situations. That makes the whole design much simpler to put together and test. Sure, it might mean the clock runs slower and the overall performance isn't as high. But for learning environments or small, basic embedded systems, it's perfect, especially when being clear and reliable matters more than raw speed.

### **2.11.3 Internal datapath of the CPU:**

The internal datapath of the PicoRV32 processor follows the classical RISC structure as shown in **figure 2.7**.

The datapath operation can be described as follows :

The Program Counter (PC) is responsible for generating the address of the subsequent instruction, which is then retrieved from the instruction memory (ROM). Following retrieval, the instruction is conveyed to the decoder. This component is tasked with generating the necessary control signals and with reading data from the register file. Computational operations are executed by the Arithmetic Logic Unit (ALU), utilizing operands sourced from registers or immediate values. Should it be required, access to data memory (such as RAM or peripheral devices) is facilitated via the system bus. The outcomes of these operations are then committed back into the register file.[62]

This systematic progression of operations contributes to efficient instruction execution while concurrently supporting a straightforward hardware architecture.

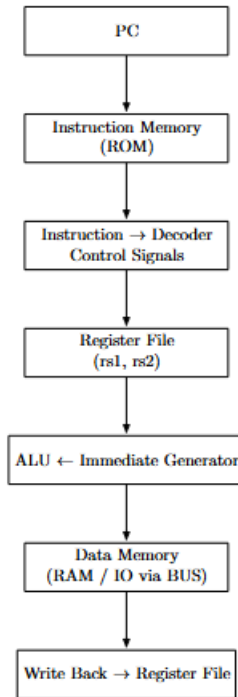


**Figure 2.7:** Instruction Datapath

#### 2.11.4 System bus architecture:

To maintain simplicity and reduce design complexity, a memory-mapped bus architecture is adopted instead of more complex protocols.

The bus enables communication between the CPU, memory, and peripherals using a shared address and data interface. All devices are accessed using specific address ranges.

The bus interface is defined by the following signals:[63]

addr[31:0]: Address bus used to select memory or peripheral registers.

wdata[31:0]: Data bus for write operations.

rdata[31:0]: Data bus for read operations.

we: Write enable signal indicating a write operation.

valid: Indicates a valid transaction request.

ready: Indicates that the selected device has completed the transaction.

This simple protocol aligns directly with the native interface of the PicoRV32 core, allowing seamless integration without the need for additional adapters or protocol converters.

### **2.11.5 Data flow description:**

So, here's how data generally moves in the system: Instructions are retrieved from read-only memory (ROM) and transmitted via the system bus. The CPU figures out what those instructions mean and acts on them using its own internal wiring. The system bus also facilitates data exchange between the processor, random-access memory (RAM), and other integrated units. Each peripheral is assigned a unique memory address and configured to respond to it. Finally, the results of the processing are stored in internal registers or sent to external devices.

### **2.11.6 Design considerations:**

The chosen datapath architecture emphasizes:

- **Simplicity:** Minimal control logic and straightforward interconnections.
- **Modularity:** Easy addition or removal of peripherals.
- **Compatibility:** Direct use of the PicoRV32 native interface.
- **Scalability:** The system can be expanded in the future by adding new modules.

This makes the design well-suited for educational purposes and rapid prototyping of embedded systems.

## **2.12 The microcontroller design flow:**

There are too many steps to build the microcontroller design as [64] :

### **2.12.1 Selection of the system specification:**

First, we start by figuring out what the microcontroller needs to do and what its main goals are. This means choosing things like the RISC-V architecture, picking out essential parts like GPIO and UART, and deciding how much ROM and RAM memory we'll need. At this point, we also think about how well it should perform, how simple the design should be, and any limits on resources. Usually, we draw a rough block diagram to get a clear picture of how the system will look and what its main pieces are. subsection Building the architectural design: Next, we look at how the microcontroller will be put together inside. We figure out the datapath "how data move around" and choose how different parts will communicate to each other. For this

project, we are going with a memory-mapped bus system(method of communication)[65]. We also create a 'memory map' to set aside specific address areas for the ROM, RAM, and other components. This stage turns our initial ideas into solid hardware design that is ready to be built.

### **2.12.2 Selection of IP cores:**

To make designing easier and quicker, we pick ready-made 'IP cores' instead of building everything ourselves from scratch, because building every component from the first steps requires a lot of time and work. We chose the PicoRV32 core for the CPU because it is simple and works well with RISC-V. Other IPs, such as the UART module and an optional I2C controller, are chosen because they are small, open-source, and easy to fit in. In This way, we can focus on getting the whole system to work together instead of worrying about the tiny details of each part.

### **2.12.3 Integration of ready IPs:**

Putting all these IPs parts together is the most important step in the whole design process. Here, we connect all the chosen modules using a shared memory-mapped bus. Every part gets its own set of memory addresses. We also build an 'address decoder' that makes sure that the right module lights up when its address is called. The CPU communicates with the ROM, RAM, and other peripherals using shared wires for things like addresses, data, and control signals[66]. All of these signals must work together at the right time. In This step, we really turn all those separate pieces into one working, complete microcontroller system.

### **2.12.4 Top-Level integration:**

Now, we gather all the system's pieces into one main'top-level' module. This module acts like the central hub, bringing in the CPU, memory, and the other components, then connecting them all using the bus. It also manages signal routing and clock distribution across the system. The result is a complete microcontroller design ready for simulation and hardware implementation.

### **2.12.5 Simulation and verification:**

Before we actually build the design on hardware, we run simulations to check if the system works correctly. We use 'testbenches' to mimic real-world situations, like writing to GPIO reg-

isters or sending data through the UART. This helps us to make sure that all the instructions run right, that memory works as it should, and that other parts react the way we expect. Simulations are great for finding and fixing design mistakes early on, saving us a lot of trouble later.

### **2.12.6 Synthesis and implementation:**

At this stage, all components of the system are combined in a single top-level module. This means turning it into something that can be put onto actual hardware, usually an FPGA. During this step, the detailed hardware description becomes tiny electrical gates, and we check the timing to make sure everything works smoothly[67]. Finally, we put the design onto the chosen hardware platform using the right design software.

### **2.12.7 Testing and validation:**

The last step is to test the microcontroller on the actual hardware. We load up some firmware programs to confirm that the system works as it should, checking things like memory access and how different parts communicate to each other. We watch outputs such as LED lights and UART messages to make sure that everything is going correctly. If we find any problems, we go back and forth, debugging and tweaking until the system is dependable and works perfectly.

### **2.12.8 Design strategy summary:**

In general, our design approach is structured and built step-by-step. We start by figuring out the system's needs, then move through planning its architecture, picking out IP cores, putting them together, and finally testing everything. Using ready-made IP cores really cuts down on the work needed, and putting things together in modules makes the design flexible and easy to expand. Of all these steps, getting those ready-made IPs to work together is the most important. It is what decides if all the system parts can actually become one unified, working microcontroller.

section Production considerations and limitations:

The proposed microcontroller design was developed and verified at the simulation level using open-source tools. However, full production and hardware implementation were not performed due to time and budget constraints.

Implementing the system in a real production environment requires additional steps, such as hardware deployment, physical testing, and optimization. In addition, the fabrication process must be outsourced to a specialized semiconductor company, which involves high costs and

long processing times.

Therefore, this work focuses on design and functional validation, providing a foundation for future production. The next chapter details the tools used and the step-by-step design process.

## **2.13 Conclusion:**

This chapter describes the architecture and design methodology of the proposed RISC-based microcontroller. It details the rationale for selecting the RISC architecture, emphasizing its benefits in terms of simplicity, performance, and power efficiency for embedded applications. The internal structure of the microcontroller is discussed, including the datapath organization, core components, and integrated peripherals such as memory and communication interfaces. This chapter also outlines the complete design flow, which encompasses the specification, implementation, verification, and testing stages. Collectively, this chapter establishes a foundational understanding of the proposed microcontroller's structure and design, thereby providing the basis for its implementation and integration, which will be addressed in the subsequent chapter.

## **Chapter 3**

# **IP integration for microcontroller and Implementation using Open-and free tools**

### **3.1 Introduction:**

This chapter details the methodology employed for the implementation of the proposed RISC-V microcontroller system, using the open-source tools for the simulation and the tests[8]. The system architecture is established by integrating a CPU core we use the Picorv32, memory modules, and various peripheral interfaces to form a complete microcontroller. Our implementation sequence involves hardware description, IP integration, simulation, subsequent verification, and ultimately, synthesis to get a ready chip.

The design flow work with architecture, which helped to facilitates the integration of the IPs. The open-source tools are consistently utilized for tasks such as simulation, waveform analysis, and the final hardware synthesis and finally get the chip design that is ready to produce.

### **3.2 Design methodology:**

The design of the ICs done by a design flow that we mention earlier, but we skipped some steps as logic and physical design, and jumped into the last 2 steps by using a ready IPs so The focus is system integration rather than low-level CPU design. In our project we are limited with the time and the budget, and the best option is to build a chip with this ready and open source components. We look for an architecture but in the end we chose to build a new one, that can meet our requirements and can finish it with time we have in our hand. we search for the IPs that we will use, after that we need to simulate and verify every IP with Icarus verilog

and GTKwaveto to make sure that the final chip can run with no problems. The next step is the integration of this IPs, we need to follow our architecture and link components with Address to make sure that the information's and the orders run with the correct way. Finally we can move to the final chip design with librlance and get the design chip that can be produce. The main IP cores used in this project are:

### 3.3 Open-source tools:

In the ICs design there are many tools can use to build your circuit, some tools need a huge budget as cadence, but there is too many open source tools with full access as skywater, Ikarus, laberlance..etc, and there is tools with Partial access as vivado from AMD, quartis from Intel. For our project we use some tools as:

#### 3.3.1 Vivado:

AMD Vivado is a tool or software that we can use for the ICs and SOC's and FPGA design. It used for Design , Synthesis, Place and Route, Verification/Simulation tools. It can help the hardware designers to reduce compile times and design iterations, while more accurately estimating power. It has to many version from 2012 to 2026 and there is a free and a paid versions can give more access so you can Choose a tier that matches your required features without overpaying.[68]the vivado interface shown in **figure:3.1**

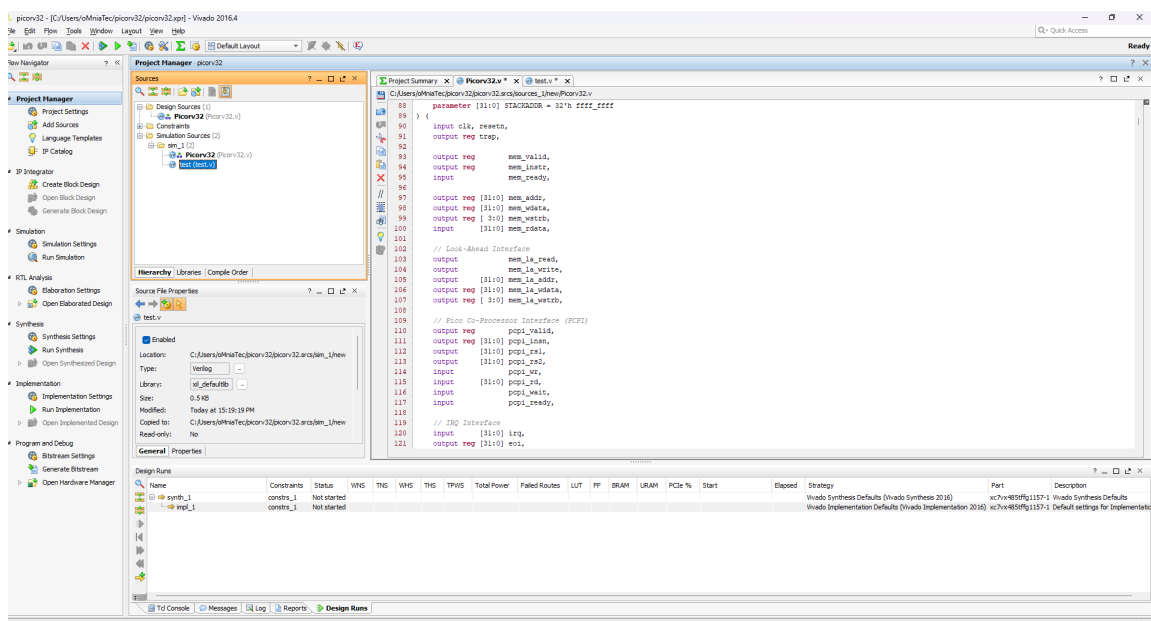


**Figure 3.1:** Vivado interface

### **3.3.2 Icarus Verilog:**

Icarus Verilog is an open-source Verilog simulation tool used for compiling and simulating hardware description language designs.

Icarus Verilog is a open source Verilog simulation tool used for compiling and simulating hardware description language designs.it was introduced for the first time in November 1998, and can be used in windows and linux. Ikarus works with the Verilog architecture and transforme it into a flagged parsing tree, after that transformed into a design graph. The process of modification involves creating modules, evaluating and deploying symbolic constants, verifying the connectivity and check for any problemes of all devices, and producing a refined and consistent design. The initial design schemas subsequently undergoes a series of optimization and logical synthesis processes. This processing refines the schema, making it more suitable for its intended purpose, before it is analyzed by the final code generator. Finally, the code generator checks the design for any errors or problemes and produces output in the suitable format. Ikarus Verilog generates C++ code using its built-in simulation class library. The software also can generate a code for sending the generated designs to other FPGAs.[69]

In this project, We use Icarus Verilog to simulate the complete RISC-V microcontroller system, and verify the IPs include the processor core picorv32 and other peripherals as the memory.

### **3.3.3 GTKWave:**

GTKWave is one of the open source tool that is a fully featured wave viewer. It supports various file formats and standard Verilog files. Primarily used for debugging Verilog or VHDL simulation models, it is designed for analyzing dumpfiles rather than real-time interaction during simulations. It can shows the visualization of the analog and digital signal data. GTKWave have a helper applications designed for specialized tasks such as file conversion, RTL parsing, and other data manipulation operations that extend beyond the typical scope of a visualization tool. GTKWave was developed for debugging large-scale integrated systems and serves as an offline alternative to external debugging tools. This 64-bit software is compatible with even the most complex designs, provided it runs on a workstation with sufficient physical memory. For Verilog, GTKWave allows users to debug simulation results of multiple signal values over varying periods of time and also at the RTL level through annotation of signal values back into the RTL for a given timestep.[70]

### 3.3.4 Laberlane:

LibreLane is a powerful open source tool that enables the construction of digital implementation flows for application specific integrated circuits (ASICs) based on electronic design automation (EDA) tools.

LibreLane have to many advantage, it is eazy to use you can configure the entrie ASIC implementation flow using one file, with full free access to all the complementary set as process design kits (PDKs), and freely modify both the infrastructure, underlying tools, and PDK to fit your needs. With flexibility in use, you can Create custom flows, by simple modifications to the default flows in the configuration file, or by writing Python scripts or plugins to implement advanced functionality[71]. The Laberlane interface shown infigure:3.2

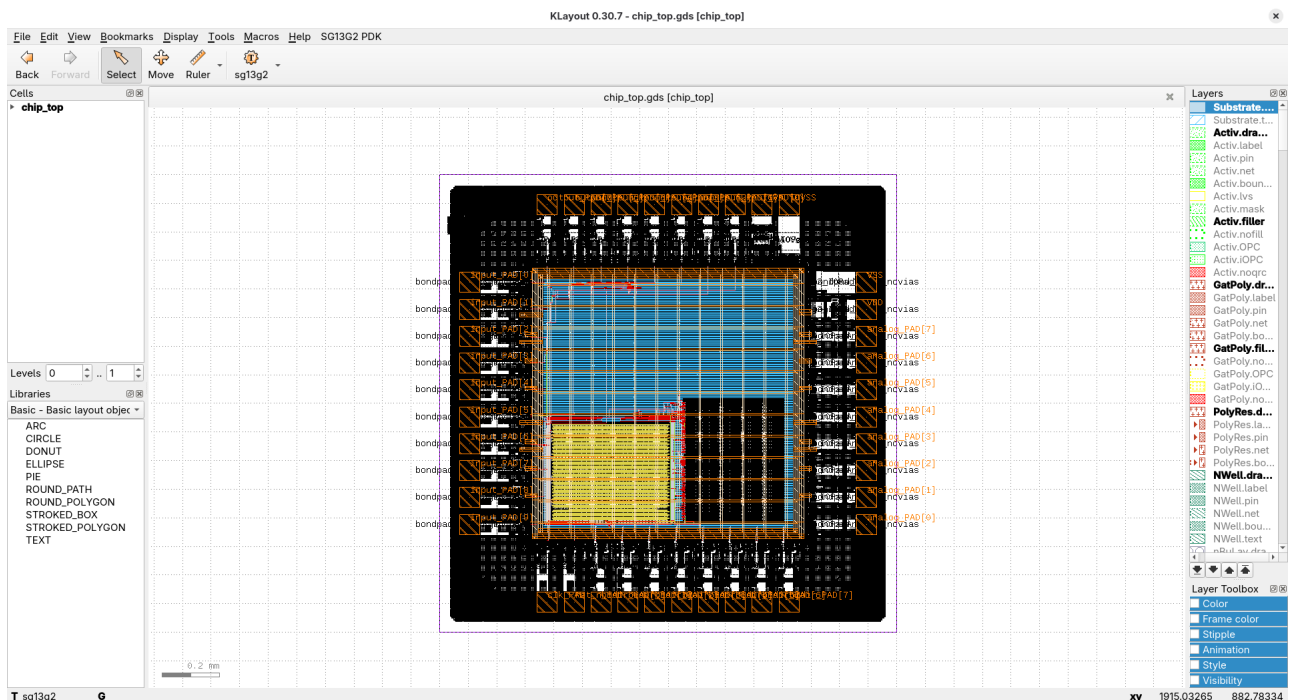


Figure 3.2: Laberlane interface

### 3.4 Simulation and verification:

First we test every IP each one tries it alone. we can use Vivado or Icarus verilog and GTKwave. for every IP we need two files: one for the main code and the second one is the test file. The type of the file end with (.v) for example: ucc-controller.v for the main file and iic-controller-tb.v for the test file code.

For the test of the IPs we need to follow some steps. After we find the IP, we need to open vivado and add the code file from add sources as shown in the figure:3.3. we will test the

I2C[72] and do the some steps for all the other IPs.



Figure 3.3: Add source in vivado.

To add the main code file we click on add or create design sources, for the test file we select add or create simulation source. After that we can run the simulation as we see in **figure:3.4** and get the result or correct any error that we find.

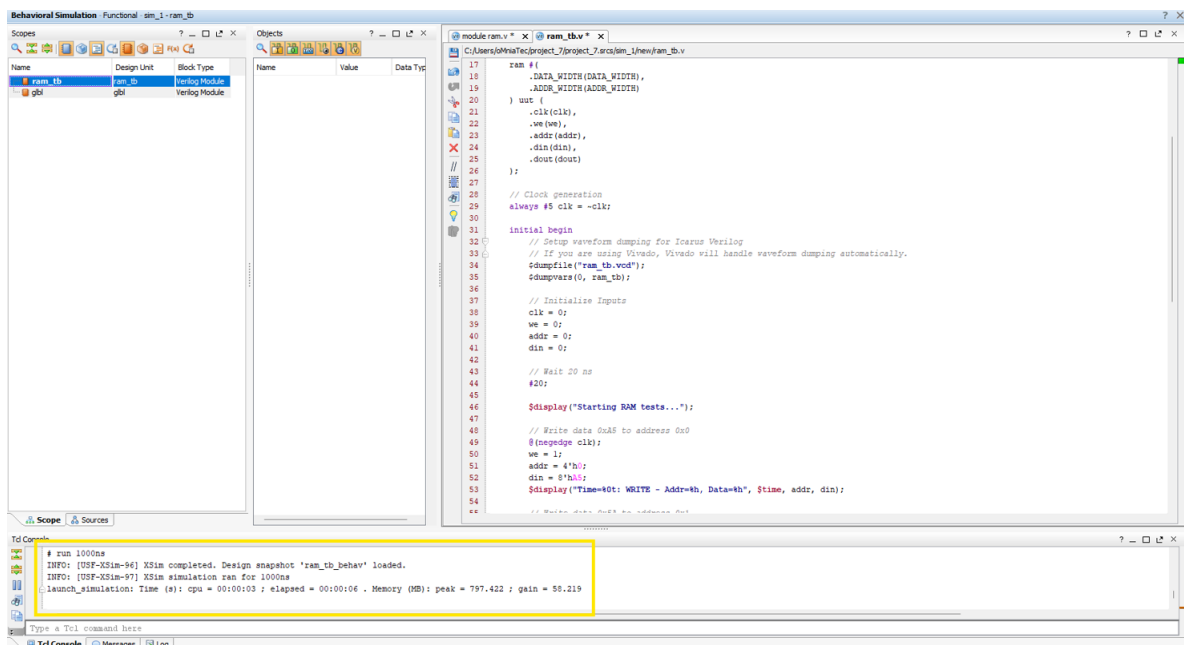


Figure 3.4: Simulation of the IP

We can also show a graph as shown in **figure:3.5** by writing "create wave config" in the console command.

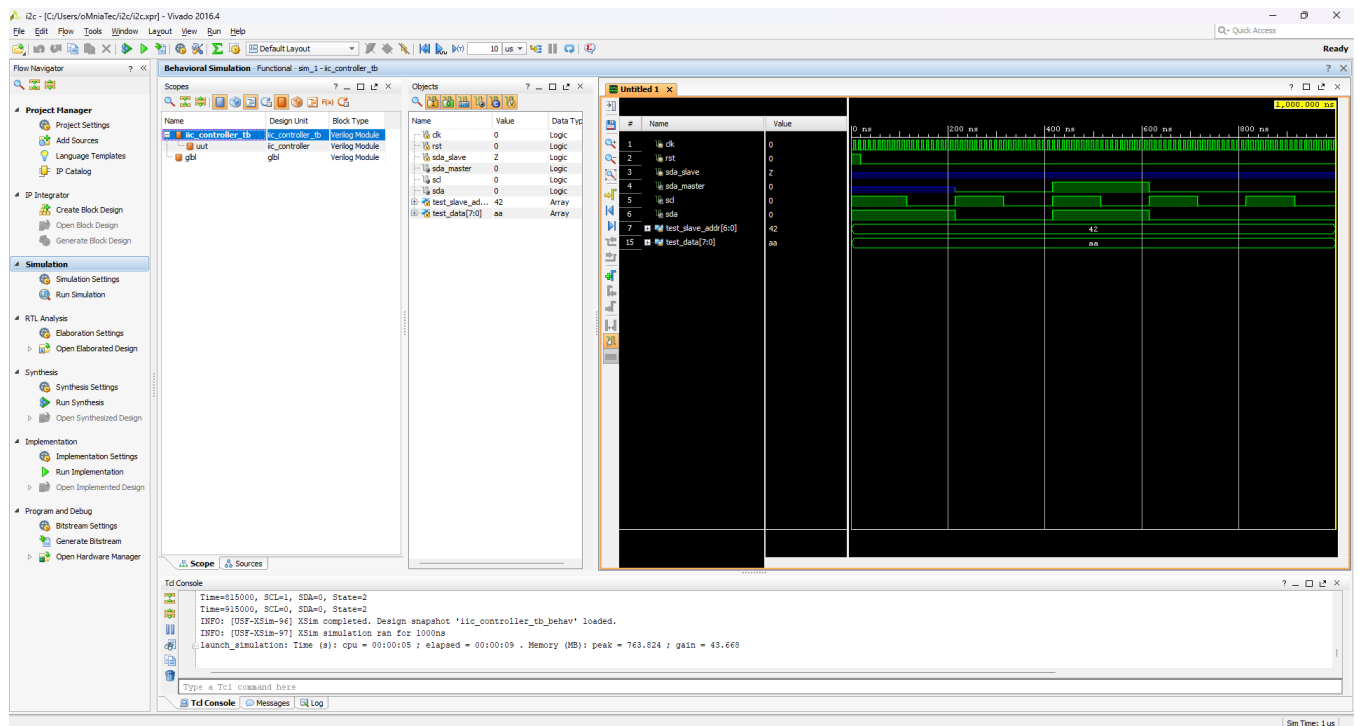


**Figure 3.5:** Simulation and the wave form of the I2C

the simulation shows some Statistics in the Vivado prints as:

- Time (s): cpu = 00:00:05 it means the simulator spent time.
- elapsed = 00:00:09 it is Real wall-clock time, it means the time you waited and it is about 9 seconds. with the actually Vivado computed is for 5 seconds.
- peak = 763.824 it means Maximum RAM used during simulation.
- gain = 43.668 it is Additional memory allocated during simulation. It means Memory end - Memory start= 43.7 MB

The graph is created by the testbench file with the neam of: iic-controller-tb.v. It creates stimulus (inputs) for the I2C controller and observes the outputs. From the testbench code we can generate signals as we can see in **figure:3.5**.The two most important signals are: SCL (Clock line) and SDA (Data line).

- The SCL signal is the actual I2C clock, each pulse means One bit is being transferred.
- The SDA signal contains the data with a rule:  
SDA changes while SCL = 0  
SDA stays stable while SCL = 1  
This allows the receiver to read the bit when SCL goes high.

- The CLK signal represent the FPGA system clock. We can almost ignore it when analyzing the I2C transfer.
- The test-slave-addr =42 is the responsible for the address transmission.

### 3.5 Peripheral integration:

After selecting the system specification, building the architecture design and selecting and testing the IPs, now we need to work for the integration of the IPs. This step can be done with tools as Vivado by Connecting all modules together so they behave as one complete microcontroller system. These IPs are integrated into the system through a memory-mapped bus architecture (communication method where memories and peripherals share the same address space), this allow the cpu to communicate and access using standard read and write operations. Each peripheral have an address range within the system memory map. The address decoding logic selects the appropriate peripheral based on the address generated by the processor. This make the communication between the peripheral possible with the cpu and more organized The memory map that we use :

- ROM address between 0x00000000 and 0x0000FFFF
- RAM address between 0x10000000 and 0x1000FFFF
- GPIO address between 0x20000000 and 0x200000FF
- UART address between 0x30000000 and 0x300000FF
- I2C address between 0x40000000 and 0x400000FF

When the CPU generates an address it look to the range and select the right one.

In PicoRV32, communication occurs through[56]:

- mem-addr: Address generated by CPU(output).
- mem-wdata: Data written by CPU(output).
- mem-rdata: Data returned to CPU(input).
- mem-valid: Memory access request(output).
- mem-ready: Operation completed(input).

### 3.6 System verification:

After we integrate all the IPs(picorv32, GPIO, UART, I2C, ROM) and give them the addresses the next step is verify them with vivado as we can see in the picture **figure:3.6**.the verification done successfully.

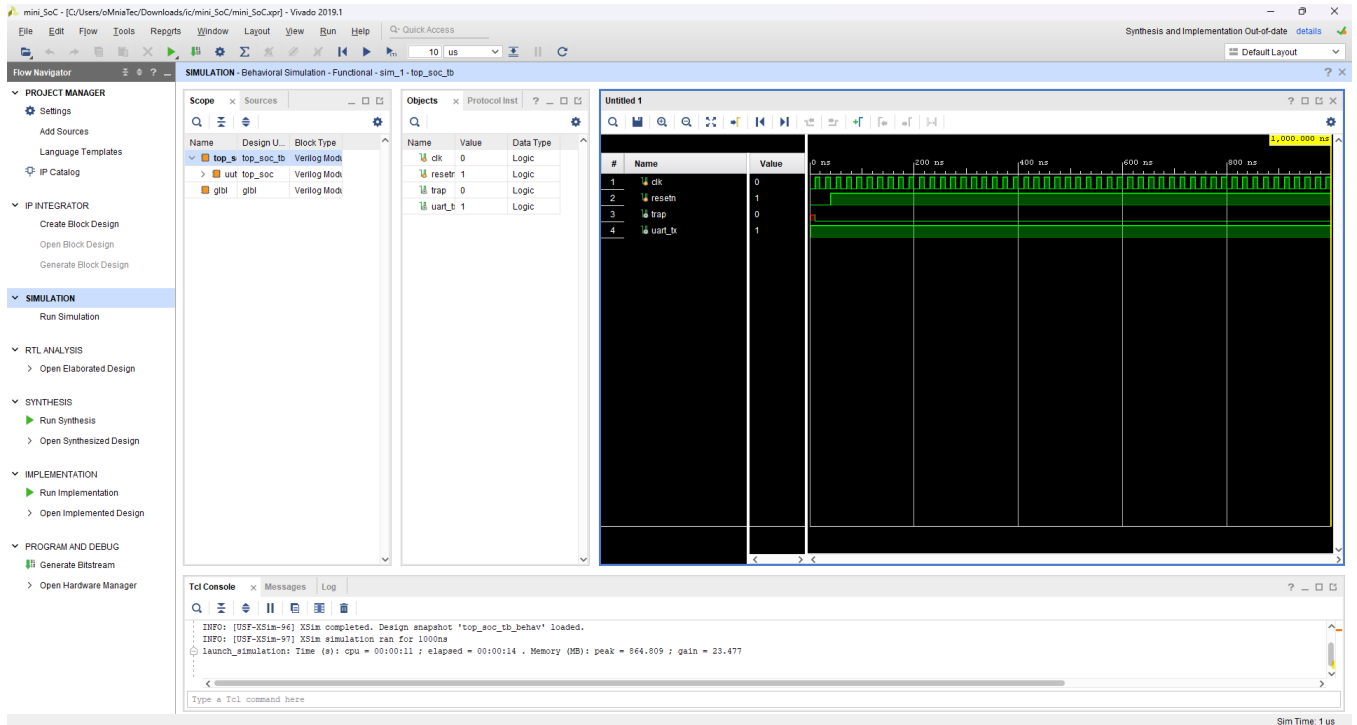


Figure 3.6: part from the configuration file

### 3.7 Synthesis flow:

Following simulation and verification, the design is prepared for synthesis using LibreLane. The synthesis process commences with an analysis of the Verilog RTL description of the microcontroller system. This RTL code is subsequently translated into a gate-level representation, comprising logic cells derived from the designated target technology library. Synthesis not only translates code but also work for more logic optimization, better performance(speed), less power consumption and silicon area to reduce hardware complexity while preserving the design's intended functionality. The resulting netlist is then utilized for subsequent implementation steps. It provides a bridge between the abstract, human-readable logic that designers create and the concrete, detailed hardware that manufacturing tools require[41].

Synthesis work first with RTL code, it describes the digital circuit in terms of registers, operations, and data flow. The synthesis process involves several key steps:

**Parsing and Elaboration:** The Laberlane tool processes the Register Transfer Level (RTL) code. This help for the errors checking and constructing an internal model of the design, which includes resolving module definitions and hierarchical relationships.

**Technology Mapping:** Laberlane translates the RTL logic into a netlist composed of standard logic gates and flip-flops that are part of the target technology library.

**Optimization:** the targets specific design objectives, including minimal area, reduced power consumption, and enhanced performance. This process frequently employs techniques such as logic minimization, retiming, and resource sharing.

**Constraint Handling:** Designers provide constraints such as clock periods, timing specifications, and area limits. The synthesis tool attempts to meet these constraints during optimization.

**Output Generation:** The final result is a output of gate-level netlist, This netlist serves as the input for subsequent physical design stages, including placement and routing.

These outputs enable the evaluation of the hardware cost and feasibility of the proposed microcontroller architecture prior to physical implementation[41].

### 3.7.1 The synthesis design:

The verification success so, next we run the synthesis design as we shown in **figure:3.7**.The full design in the pdf file Project Document

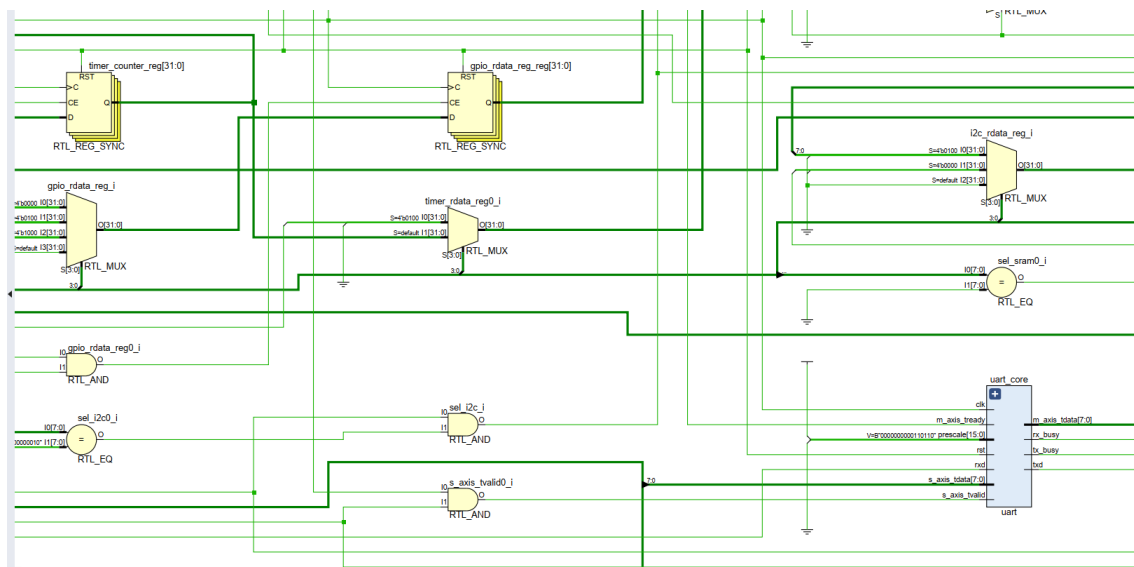


**Figure 3.7:** part from the configuration file

## 3.8 Top level design:

Now for the top level we use labrelane, to open labrelane we need to run "nix-shell" in the terminal to make all the packages bundled with LibreLane available. The next command is "librelane "the files directory " /config.json -pdk ihp-sg13g2"

config.json is a configuration file contains values set by the user, With them we can control the flows. This is part from our configuration file design:

**Listing 3.1:** OpenLane Configuration File

```
meta:
  version: 3
  flow: Classic
  substituting_steps:

    Checker.IllegalOverlap: null

    KLayout.DRC: null
    Checker.KLayoutDRC: null

    Magic.DRC: null
    Checker.MagicDRC: null

    KLayout.Antenna: null
    Checker.KLayoutAntenna: null

    KLayout.Density: null
    Checker.KLayoutDensity: null

    OpenROAD.IRDropReport: null
    KLayout.XOR: null
    Netgen.LVS: null
    Checker.LVS: null

    Checker.disconnectedpins: null
    Checker.linterrors: null

DESIGN_NAME: top_soc

VERILOG_FILES:
- dir:../src/top_soc.v
- dir:../src/picorv32.v
```

```
- dir:../src/i2c_master.v
- dir:../src/uart.v
- dir:../src/uart_rx.v
- dir:../src/uart_tx.v
- dir:../src/sram_ram_asic_macro.v

VERILOG_DEFINES: [FUNCTIONAL]

PRIMARY_GDSII_STREAMOUT_TOOL: klayout

PNR_SDC_FILE: dir::chip_top.sdc
SIGNOFF_SDC_FILE: dir::chip_top.sdc
FALLBACK_SDC: dir::chip_top.sdc

VDD_NETS:
- VDD

GND_NETS:
- VSS

CLOCK_PORT: clk
CLOCK_NET: clk

CLOCK_PERIOD: 125

FP_SIZING: absolute

DIE_AREA: [0, 0, 1700, 1700]
CORE_AREA: [50, 50, 1650, 1650]

PL_TARGET_DENSITY_PCT: 10
GRT_ALLOW_CONGESTION: true

PDN_CORE_RING: True

PDN_CORE_RING_VWIDTH: 15
PDN_CORE_RING_HWIDTH: 15

PDN_CORE_RING_VSPACING: 8
PDN_CORE_RING_HSPACING: 8
```

```

PDN_CORE_RING_CONNECT_TO_PADS: False
PDN_ENABLE_PINS: False

MAGIC_EXT_UNIQUE: notopports

FP_PIN_ORDER_CFG: dir::pin_order.cfg

PDN_CFG: dir::pdn_cfg.tcl

MACROS:
  RM_IHPSG13_1P_1024x32_c2_bm_bist:
    gds:
      - pdk_dir::libs.ref/sg13g2_sram/gds/RM_IHPSG13_1P_1024x32_c2_bm_bist.gds
    lef:
      - pdk_dir::libs.ref/sg13g2_sram/lef/RM_IHPSG13_1P_1024x32_c2_bm_bist.lef
    vh:
      - pdk_dir::libs.ref/sg13g2_sram/verilog/RM_IHPSG13_1P_1024x32_c2_bm_bist.v
      - pdk_dir::libs.ref/sg13g2_sram/verilog/
        RM_IHPSG13_1P_core_behavioral_bm_bist.v
    lib:
      "*_typ_1p20V_25C":
        - pdk_dir::libs.ref/sg13g2_sram/lib/
          RM_IHPSG13_1P_1024x32_c2_bm_bist_typ_1p20V_25C.lib
      "*_fast_1p32V_m40C":
        - pdk_dir::libs.ref/sg13g2_sram/lib/
          RM_IHPSG13_1P_1024x32_c2_bm_bist_fast_1p32V_m55C.lib
      "*_slow_1p08V_125C":
        - pdk_dir::libs.ref/sg13g2_sram/lib/
          RM_IHPSG13_1P_1024x32_c2_bm_bist_slow_1p08V_125C.lib
    instances:
      internal_sram.sram_0:
        location: [150, 150]
        orientation: N
      internal_sram.sram_1:
        location: [750, 150]
        orientation: N

```

In general, for any design at a minimum we need to specify the following variables: (DESIGN-NAME, VERILOG-FILES, CLOCK-PERIOD, CLOCK-PORT)[30]

After we run the commend "librelane "the files directory " /config.json -pdk ihp-sg13g2" the

simulation and the verification start with labrelane as we see in the **figure:3.8**. The simulation take a long time about 1 or 2 hours, and it give us the errors and the warning to find a solution for them by correcting the verilog code.

```

* nom_slow_1p08V_125C
* nom_typ_1p20V_25C
[18:10:48] WARNING [Checker.MaxCapViolations] Max Cap violations      flow.py:701
found in the following corners:
* nom_fast_1p32V_m40C
* nom_slow_1p08V_125C
* nom_typ_1p20V_25C
[18:10:48] WARNING [Misc.ReportManufacturability]                  flow.py:701
klayout__drc_error__count not reported.
Klayout.DRC may have been skipped.
[18:10:48] WARNING [Misc.ReportManufacturability]                  flow.py:701
magic__drc_error__count not reported. Magic.DRC
may have been skipped.
[18:10:48] ERROR    The following error was encountered while      __main__.py:189
running the flow:
One or more deferred errors were
encountered:
2 power grid violations (as reported by
OpenROAD PSM- you may ignore these if LVS
passes) found.
[18:10:48] ERROR    LibreLane will now quit.                        __main__.py:190
make: *** [Makefile:29: librelane] Error 2

```

**Figure 3.8:** picture that shows the simulation filed

After we correct all the errors the simulation will successes and open up the labrelane interface and we got the result as we see in **figure:3.9**. This is the microcontroller chip with Dimensions ore area of 2.55 mm<sup>2</sup> and a total die area of 2.89 mm<sup>2</sup> in 130 nm.

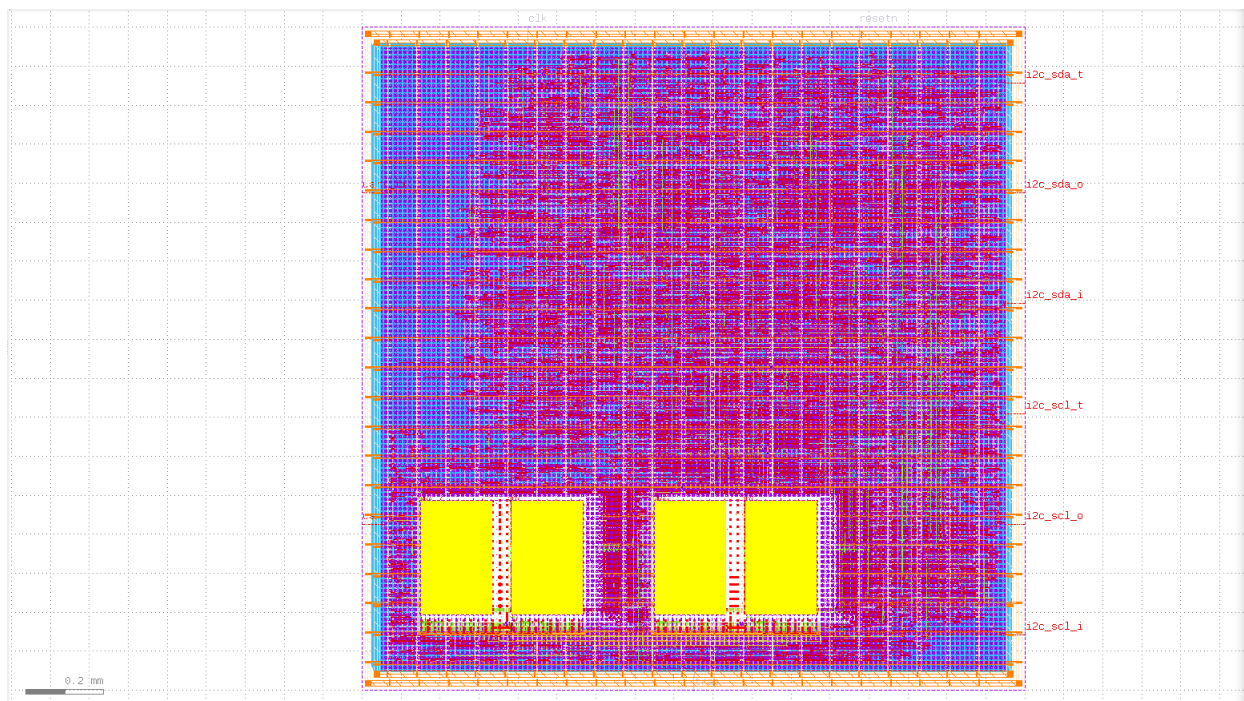


**Figure 3.9:** the top level of the microccontroller chip

### 3.9 The finale Soc:

We successfully implemented the final System-on-Chip (SoC), as shown in **figure:3.10**. The layout represents the complete physical design generated after the ASIC implementation flow, including the processor core, memory modules, peripherals, routing network, power distribution, and I/O pads. This demonstrates the successful integration of all IP cores into a single manufacturable chip. we got better result after finishing the chip as the power, the chip only consume 1.12 mw instead of 2.37 mw and less core area with 1.69 mm<sup>2</sup> instead of 2.55 mm<sup>2</sup>.

The finale SoC provides 16 external I/O pins (VDD, VCC, I2C-Scl ,I2C-SDA, DHT11 data, trap pad, RST, clk, IO VSS, IO VDD, VSS, VDD, UART.rx, UART.tx, I2C-Scl, I2C-SDA), allowing it to interface with external devices and peripherals. To validate the functionality of the design, we used the microcontroller to communicate with a DHT11 temperature and humidity sensor. The SoC successfully acquired the sensor measurements, demonstrating the correct operation of the processor, memory, GPIO interface, and the overall system integration.

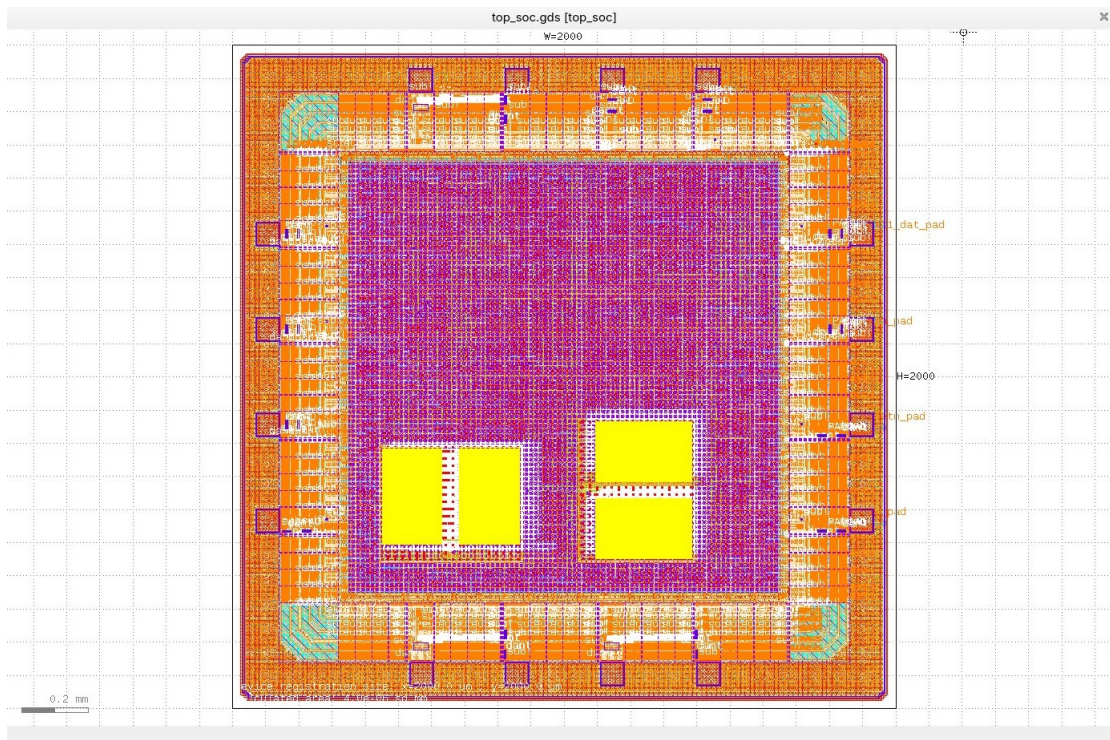


**Figure 3.10:** The finale chip

### 3.10 Result:

Following functional simulation and verification, the top-level RTL design underwent synthesis and technology mapping using the LibreLane design environment, converting the abstract,

human-readable structural Verilog code into an optimized, gate-level netlist referencing standard library cells. The system's integration relied on a strict memory-mapped bus architecture, establishing dedicated memory windows for the system components, including the PicoRV32 processor core, ROM, RAM, GPIO, UART, and I<sup>2</sup>C peripherals. Implemented using an IP-based design methodology and open-source EDA tools, the final design occupies a core area of 1.69 mm<sup>2</sup> and a total die area of 4 mm<sup>2</sup> in 130 nm technology, containing approximately 15,727 standard cells. Timing analysis confirmed correct operation at the target frequency of 8 MHz with zero setup and hold violations, while a positive timing slack indicates that the design could theoretically operate at frequencies approaching 28 MHz. Furthermore, power analysis reported a total power consumption of 1.12 mW, demonstrating the suitability of the design for low-power embedded applications. Finally, physical verification completed successfully with zero DRC and antenna violations, confirming the correctness and manufacturability of the layout based on this memory-mapped blueprint.

### **3.10.1 Performance Efficiency:**

The proposed microcontroller demonstrates efficient performance by combining a lightweight RISC-V processor core with a simple memory-mapped bus architecture. The PicoRV32 core provides an effective balance between computational capability and hardware complexity, making it suitable for embedded applications. The successful execution of all integrated modules and the absence of timing violations confirm that the system operates reliably under the specified clock constraints. Furthermore, the available timing margin indicates that the design maintains stable operation while preserving the possibility of future performance improvements.

### **3.10.2 Physical area utilization:**

The physical implementation results demonstrate an efficient utilization of the available silicon area. The adopted IP-based design methodology enables the integration of processing, memory, and peripheral resources within a compact layout while preserving sufficient free space for routing and future extensions. This moderate utilization ratio contributes to easier physical implementation, reduced routing congestion, and improved design scalability. Consequently, the proposed architecture achieves a favorable compromise between functionality, implementation complexity, and silicon cost.

**Gate-level footprint:** The synthesized design contains a moderate number of logic elements, reflecting the compact nature of the proposed microcontroller architecture. The gate-level implementation includes sequential elements responsible for data storage and control operations, as well as combinational logic used for data processing, address decoding, and peripheral

communication. The resulting gate-level footprint confirms that the design remains lightweight while providing the functionality required for a complete embedded processing platform.

**Peripherals footprint:** Minimizing complex interconnects by bypassing heavy buses (like AXI/Wishbone) in favor of a lean memory-mapped decoding bus kept the overall silicon area footprint small and highly optimized.

**Resource breakdown:** A significant portion of the hardware resources is allocated to the processor core, memory blocks, and bus interconnection logic. Additional resources are dedicated to peripheral interfaces, clock distribution, and timing optimization structures introduced during implementation. This resource distribution demonstrates that the design prioritizes processing and communication functionality while maintaining a balanced utilization of hardware resources.

### **3.10.3 Power consumption:**

Power analysis indicates that the proposed microcontroller achieves low overall power consumption, making it suitable for energy-constrained embedded systems. The total power consumption is composed of internal power, switching power, and leakage power. The relatively low leakage contribution reflects the efficiency of the selected technology and architecture. These results demonstrate that the proposed design successfully balances functionality and energy efficiency, which is an important requirement for modern embedded applications.

### **3.10.4 Timing closure and clock characteristics:**

Timing closure was successfully achieved across all analyzed operating conditions. Static timing analysis reported no setup or hold violations, confirming the correctness of signal propagation throughout the design. The implemented system operates at a target frequency of 8 MHz, while the positive timing slack indicates the availability of additional timing margin. This margin suggests that the architecture could support higher operating frequencies if required. The successful timing closure validates both the architectural design and the integration methodology adopted for the proposed microcontroller.

### 3.11 Comparison of the proposed microcontroller with existing chips:

A critical phase in validating a custom hardware architecture involves benchmarking it against established industry-standard solutions. To provide a comprehensive evaluation of the proposed PicoRV32-based RISC-V microcontroller, its primary technical indicators are contrasted with two ubiquitous commercial devices: the 32-bit ARM Cortex-M3 (STM32F103C8T6) and the 8-bit AVR-based Arduino Uno (ATmega328P). **Table 3.1** presents a multi-dimensional comparison covering architectural design, fabrication technology node, operating clock frequencies, license type, and overall economic scalability. Through this comparative framework, the tradeoffs between proprietary high-frequency silicon and flexible, cost-effective open-source Application-Specific Integrated Circuit (ASIC) design capabilities using the Sky130 process are clearly demonstrated.

**Table 3.1:** Microcontroller Feature Comparison

| Feature                  | Proposed MCU                              | STM32F103C8T6 | Arduino Uno (ATmega328P) |
|--------------------------|-------------------------------------------|---------------|--------------------------|
| CPU Architecture         | RISC-V (PicoRV32)                         | ARM Cortex-M3 | AVR 8-bit                |
| Technology               | Sky130 (130 nm)                           | 90 nm CMOS    | 350 nm CMOS              |
| Clock Frequency          | 8 MHz ( $\approx 28$ MHz theoretical max) | 72 MHz        | 16 MHz                   |
| Core Type                | Open-Source IP                            | Proprietary   | Proprietary              |
| Customization            | Very High                                 | Limited       | Limited                  |
| ASIC Design Capability   | Yes                                       | No            | No                       |
| Development Cost         | Very Low                                  | High          | High                     |
| Flexibility              | High                                      | Medium        | Low                      |
| Power Consumption        | 2.37 mW                                   | Higher        | Higher                   |
| Educational/Research Use | Excellent                                 | Good          | Good                     |
| Open-Source Design Flow  | Yes                                       | No            | No                       |

### 3.12 Architecture impact:

The achieved power, area, and performance results are strongly influenced by the architectural choices adopted in this work. The use of the lightweight PicoRV32 RISC-V processor, reusable IP cores, and a memory-mapped bus architecture reduces hardware complexity and minimizes unnecessary logic resources. In addition, the modular organization of the system facilitates

integration and future expansion without significantly increasing implementation cost. These architectural decisions contribute directly to the compact area, low power consumption, and satisfactory performance observed in the final implementation.

### **3.13 Implementation challenges:**

The implementation of the proposed microcontroller presented several technical challenges throughout the design flow. One of the main difficulties was the need to move between multiple development tools, as no single platform was capable of performing all stages of the design process except the paid platform as cadence. Vivado was used for IP verification and integration, while LibreLane was used for synthesis and physical implementation. Managing the compatibility and data exchange between these tools required additional effort and careful verification.

Another significant challenge was the long simulation and implementation time. Depending on the complexity of the design and the verification scenario, a complete simulation or implementation run could take several hours. In many cases, errors were discovered only after the completion of a long execution cycle, requiring modifications to the source code followed by a new simulation run. This iterative process considerably increased the development time.

Debugging was also a challenging task. Identifying the source of functional errors within a system composed of multiple integrated IP cores was often difficult. Errors could originate from incorrect signal connections, address mapping conflicts, synchronization issues, or interface mismatches between modules. Locating and correcting these issues required extensive waveform analysis and repeated verification.

The limited availability of documentation, tutorials, and community support for some open-source EDA tools represented another obstacle. Compared to commercial design environments, fewer educational resources were available, making the learning process slower and requiring additional experimentation to understand the implementation flow.

Several technical challenges were also encountered during system integration, including:

- Integration of multiple IP cores developed by different authors.
- Address decoding and peripheral selection logic.
- Clock and reset signal distribution across all modules.
- Signal synchronization between processor, memories, and peripherals.
- Verification of read and write transactions on the shared bus.

- Timing closure during synthesis and physical implementation.
- Understanding synthesis, placement, routing, and timing reports generated by the design tools.

Despite these challenges, all identified issues were successfully resolved, leading to the successful implementation, verification, and physical realization of the proposed RISC-V microcontroller.

### **3.14 Conclusion:**

This chapter presented the implementation of the proposed RISC-V microcontroller using reusable IP cores and open-source design tools. The integration of the PicoRV32 processor, memory modules, and peripherals was described in detail. The memory organization, bus architecture, address decoding logic, and peripheral interfaces were successfully implemented and verified through simulation.

The application of open-source tools proved to be an effective solution for developing and validating the microcontroller architecture, providing a flexible and low-cost design flow suitable for educational and research applications.

## Generale conclusion:

This thesis presented the complete design, verification, and implementation flow of a modular, open-source RISC-V-based microcontroller. The primary objective of this work was to demonstrate the feasibility, reliability, and cost-effectiveness of using open-source Electronic Design Automation (EDA) toolchains to develop complex mixed-signal and digital integrated circuits (ICs) under a reference 130 nm CMOS fabrication technology. In the conceptual phase, a rigorous architectural evaluation led to the selection of a single-cycle datapath layout to optimize simplicity and instruction throughput efficiency. At the core of the system, the open-source PicoRV32 processor core was adopted due to its compliance with the RISC-V Instruction Set Architecture (ISA) and its high efficiency in embedded applications. To realize a fully functional microcontroller environment, essential Intellectual Property (IP) peripheral blocks—including General-Purpose Input/Output (GPIO), a Universal Asynchronous Receiver-Transmitter (UART), and an Inter-Integrated Circuit I2C controller—were successfully integrated alongside memory modules (RAM/ROM). These sub-systems were interconnected using a unified, shared memory-mapped bus architecture, governed by a dedicated address decoder to ensure precise peripheral access and optimal data collision management. The validation strategy followed a comprehensive digital design flow. Individual peripheral components and the top-level system architecture were iteratively simulated and debugged using toolchains like Vivado were constructed to generate functional inputs (such as validating I2C clock/data stability and checking data framing). This ensured that all components executed instructions accurately, successfully read/wrote to registers, and maintained functional integrity before proceeding to implementation. Finally, the top-level integration was synthesized and analyzed using LibreLane and AMD Vivado to extract physical layout characteristics. The design successfully achieved timing closure and was fully evaluated across critical physical design metrics: performance efficiency, silicon area footprint, and power consumption. The results demonstrate that combining modular, reusable IP blocks with modern open-source EDA tools presents a highly scalable, reliable, and accessible methodology for contemporary IC development, lowering the financial barriers typically associated with proprietary chip engineering.

**Perspectives and future work:** While the proposed microcontroller architecture successfully fulfills its design parameters, several avenues remain open to expand its capabilities and optimize its integration in future mixed-signal hardware design:

**Physical layout tape-out (ASIC GDSII Generation):** The next physical milestone involves taking the synthesized gate-level netlist through full placement and routing (PR) via open-source tools like OpenLane to generate the final GDSII layout. This would allow a physical tape-out of the 130 nm chip on a multi-project wafer (MPW) run (e.g., SkyWater 130nm PDK).

**Advanced mixed-signal hardening:** A natural extension of this instrumentation-focused work is the physical integration of true analog macros onto the silicon die. Integrating on-chip Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) directly alongside the PicoRV32 core would create a self-contained, fully integrated mixed-signal SoC capable of direct real-world sensor acquisition.

**Pipelined datapath upgrade:** To significantly push the maximum operating frequency and performance of the CPU core, future architectural iterations can replace the single-cycle implementation with a 3-stage or 5-stage pipelined datapath architecture, allowing higher execution speeds for intensive computing tasks.

**Enhanced communication and security peripherals:** Integrating additional industry-standard communication hardware protocols (such as SPI or CAN bus) would improve the chip's adaptability within industrial automated systems and IoT node deployments. Additionally, adding lightweight hardware cryptographic accelerators would secure sensor data directly at the edge.

**Implementation:** As future work, the proposed microcontroller can be implemented on an FPGA platform to validate its functionality in real hardware. The next major step is the fabrication of the chip using the 130 nm technology. Additional peripherals and advanced features can also be integrated to improve functionality and performance, making the system suitable for a wider range of embedded applications.

# Bibliography

- [1] Z. Qiu, X. Shen, and Z. Zhao, “Development trends and prospects of semiconductor devices and technology,” *Development*, vol. 81, 2024.
- [2] G. Martin and H. Chang, “System-on-chip design,” in *ASICON 2001. 2001 4th International Conference on ASIC Proceedings (Cat. No.01TH8549)*, pp. 12–17, 2001.
- [3] A. S. Abdul-Qawy, P. Pramod, E. Magesh, and T. Srinivasulu, “The internet of things (iot): An overview,” *International Journal of Engineering Research and Applications*, vol. 5, no. 12, pp. 71–82, 2015.
- [4] G. K. Sheelam, *Advanced communication systems and next-gen circuit design: intelligent integration of electronics, wireless infrastructure, and smart computing systems*. Deep Science Publishing, 2025.
- [5] G. E. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, no. 8, pp. 114–117, 1965.
- [6] G. E. Moore, “Progress in digital integrated electronics,” in *IEDM Technical Digest*, pp. 11–13, IEEE, 1975.
- [7] E. Cui, T. Li, and Q. Wei, “Risc-v instruction set architecture extensions: A survey,” *IEEE Access*, vol. 11, pp. 24696–24711, 2023.
- [8] R. Balas, A. Ottaviano, and L. Benini, “Cv32rt: Enabling fast interrupt and context switching for risc-v microcontrollers,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 6, pp. 1032–1044, 2024.
- [9] M. Jahnke, L. Bublitz, and U. Kulau, “Performance evaluation of picorv32 risc-v softcore for resource-constrained devices,” in *2023 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pp. 1–6, 2023.
- [10] H. Veendrick, *Bits on Chips*, vol. 10. Berlin: Springer International Publishing, 2019.

- [11] Cadence Design Systems, “What is ic design?.” [https://www.cadence.com/en\\_US/home/explore/what-is-ic-design.html](https://www.cadence.com/en_US/home/explore/what-is-ic-design.html), n.d. Accessed: 2026-02-10.
- [12] EDISCO, “What is an integrated circuit?.” [https://www.edisco.it/mechpower/wp-content/uploads/sites/100/2019/04/What\\_is\\_an\\_integrated\\_circuit.pdf](https://www.edisco.it/mechpower/wp-content/uploads/sites/100/2019/04/What_is_an_integrated_circuit.pdf), Apr. 2019. Accessed: 2026-02-10.
- [13] H. Veendrick, *Bits on Chips*. Cham, Switzerland: Springer International Publishing, 2019.
- [14] A. S. Sedra and K. C. Smith, *Microelectronic Circuits*. New York, USA: Oxford University Press, 7 ed., 2015.
- [15] J. Lienig and M. Thiele, *Fundamentals of Electromigration-Aware Integrated Circuit Design*. Cham, Switzerland: Springer, 2018.
- [16] Synopsys, “What is a mixed-signal integrated circuit?,” 2024. Accessed: 14-Mar-2026.
- [17] ELE Times Research Desk, “Analog ic vs digital ic – difference and comparison,” Dec. 2024. Accessed: 14-Mar-2026.
- [18] ICDrex, “What is integrated circuit design? how does the design process work?.” <https://www.icdrex.com/what-is-integrated-circuit-design-how-does-the-design-process-work/>, 2025. Accessed: 2026-02-10.
- [19] H. S. Kousar, D. Srivastava, A. J. Karttunen, M. Karppinen, and G. C. Tewari, “p-type to n-type conductivity transition in thermoelectric cosbs,” *APL Materials*, vol. 10, no. 9, p. 091104, 2022.
- [20] V. Bernardo, C. P. Correia, R. A. M. L., and J. P. de M. C., “A revision of the semiconductor theory from history to applications,” *Discover Applied Sciences*, vol. 59, p. 6, 2024.
- [21] M. A. Rahman, “A review on semiconductors including applications and temperature effects in semiconductors,” *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, vol. 24, no. 1, pp. 1–9, 2016.
- [22] N. C. R. R. Sharmile and D. S., “A comprehensive review of quality control and reliability research in micro–nano technology,” *Technologies*, vol. 13, no. 3, p. 94, 2025.
- [23] C. P. Correia, V. Bernardo, R. A. M. L., and J. P. d. M. C., “A revision of the semiconductor theory from history to applications,” *Discover Applied Sciences*, vol. 59, p. 36, 2024.
- [24] BYJU’S, “P-n junction.” <https://byjus.com/physics/p-n-junction/>. Accessed: 2026-03-18.
- [25] D. T. Horn, “Electronic components,” in *A Complete Guide*, p. 181, 1992.

- [26] N. H. E. Weste and D. M. Harris, “Cmos vlsi design,” in *CMOS VLSI Design: A Circuits and Systems Perspective*, p. 2, 4th ed., 2011.
- [27] Shindengen Electric Manufacturing Co., Ltd., “What are mosfets?.” [https://www.shindengen.com/products/semi/column/basic/mosfet/what\\_are\\_mosfets.html](https://www.shindengen.com/products/semi/column/basic/mosfet/what_are_mosfets.html). Accessed: 2026-03-21.
- [28] L. Wang, “Research progress on metal-oxide semiconductor field effect transistors,” *Highlights in Science, Engineering and Technology*, vol. 80, pp. 92–94, Dec. 2023.
- [29] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective, 4th Edition*. Pearson, 2011. p. 4.
- [30] FOSSi Foundation, “Getting started – i am new to libreLane.” [https://libreLane.readthedocs.io/en/latest/getting\\_started/newcomers/index.html](https://libreLane.readthedocs.io/en/latest/getting_started/newcomers/index.html), 2026. LibreLane Documentation, Accessed: Jun. 10, 2026.
- [31] Editorial Board, *Basics of Electronics & Microprocessor*. 2018. e-book ed., pp. 100–109.
- [32] K. K. CH, “Digital logic design notes,” 2018. B.Tech II Year II Sem, Malla Reddy College of Engineering and Technology, Secunderabad, India, p. 93.
- [33] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson, 4th ed., 2011. p. 327.
- [34] MathWorks, “What is an ip core?.” <https://www.mathworks.com/discovery/ip-core.html>, 2026. Accessed: May 23, 2026.
- [35] A. George, “An overview of risc vs. cisc,” in *[1990] Proceedings. The Twenty-Second Southeastern Symposium on System Theory*, pp. 436–438, 1990.
- [36] P. P. M. (More) and J. T. More, *Microprocessor for Semester – IV, Second Year (S.Y.) page 1.1*.
- [37] A. S. Abdelmaksoud, “A critical analysis of microprocessor technologies, trends, and innovations,” tech. rep., Telecommunication Engineering Department, Faculty of Engineering, Egyptian Russian University, Badr City, Cairo, Egypt, sep 2025. Review Paper. page 2.
- [38] E. Board, *Basics of Electronics Microprocessor. pages 175-176*. 2018.
- [39] Y. Saini and L. Kumar, “Designing of microprocessor using vhdl model,” *International Journal of Economic Perspectives*, vol. 18, no. 12, pp. 37–49, 2024.

- [40] R. C. Rakesh, C. S. Chetan, and J. S. Baligar, "Design of 16 bit risc processor and implementation using mips technique," *Scholars Journal of Engineering and Technology (Sch J Eng Tech)*, p. 287, 2023.
- [41] EEE Project, "Microprocessor vs microcontroller." <https://eeepproject.com/microprocessor-vs-microcontroller/>, 2026. Accessed: Jun. 1, 2026.
- [42] R. S. Kaler, *Microprocessors and Microcontrollers*. New Delhi, India: I.K. International Publishing House pages. 472–473, 3rd ed., 2019.
- [43] J. W. Valvano, *Embedded Systems: Introduction to ARM Cortex-M Microcontrollers*. 5th ed., 2014. 2nd printing, June 2014.
- [44] K. Garza, C. Goble, J. Brooke, and C. Jay, "Framing the community data system interface," in *Proceedings of the 2015 British HCI Conference*, p. 3210, ACM, July 2015.
- [45] D. Gribble and J. Herzog, "A risc architecture for high-speed data acquisition," *IEEE Transactions on Instrumentation and Measurement*, vol. 43, no. 3, pp. 457–462, 1994.
- [46] GeeksforGeeks, "Differences between data paths," 2025. Accessed: 2026-03-24.
- [47] ScienceDirect, "Datapath - an overview," 2025. Accessed: 2026-03-24.
- [48] PatSnap, "What is a control unit in a cpu," 2024. Accessed: 2026-04-19.
- [49] P. Anandan, C. H. M. S. Kumar, and B. Venkataramanaiah, "Effective cu design based implementation of a 64-bit processor for signal processing applications," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 9, p. 2344, January 2020.
- [50] D. S. Dawoud and R. Peplow, *Digital System Design – Use of Microcontroller*. University of Kwa-Zulu Natal, 2007. p. 127.
- [51] A. J. Pasquale, "Microcontrollers," Dec. 2025. Accessed: page 10..20.
- [52] ElProCus, "Know about types of registers in 8051 microcontroller." <https://www.elprocus.com/know-about-types-of-registers-in-8051-microcontroller/>, 2013. Accessed: 2026-04-19.
- [53] M. Geden and K. Rasmussen, "Regguard: Leveraging cpu registers for mitigation of control- and data-oriented attacks." arXiv preprint arXiv:2110.10769, 2021. Accessed: 2026-04-19, referenced page 1.
- [54] J. Ahmed, "Frequency divider in system verilog," Jan. 2025.

- [55] Q. Madariaga, “Development of a risc-v microprocessor for asic integration,” Master’s thesis, Grenoble-INP, Palaiseau, France, 2022. Master Thesis, Nanotech Program, OMEGA Microelectronics page 10.
- [56] C. Wolf, “Picorv32 - a size-optimized risc-v cpu.” <https://github.com/YosysHQ/picorv32/blob/main/README.md>, 2015. GitHub repository, accessed: 2026-04-19.
- [57] D. S. Dawoud and R. Peplow, “Digital system design — use of microcontroller,” in *Proceedings of the University of KwaZulu-Natal*, p. 130, n.d.
- [58] R. Soniwala, “Gpio pins: Behind the scenes,” 2023. Accessed: 2026-04-20.
- [59] Texas Instruments, *TMS320DM357 DMSoC General-Purpose Input/Output (GPIO) User’s Guide*. Texas Instruments, August 2010. Accessed: 2026-04-20.
- [60] Hilscher, “Uart / universal asynchronous receiver-transmitter,” n.d. Accessed: 2026-04-20.
- [61] J. Wu, *A Basic Guide to I2C*. Texas Instruments. Application Note, Accessed: 2026-04-20.
- [62] G. Kanda and K. Ryoo, “Design of an integrated cryptographic soc architecture for resource-constrained devices,” *International Journal of Electrical and Electronics Research*, vol. 10, pp. 230–244, Jun 2022.
- [63] “Hdlbits — verilog practice.” Website, Oct. 2024. A collection of small circuit design exercises for practicing digital hardware design using Verilog.
- [64] A. Sengupta, “Design flow of a digital ic: The role of digital icsoc design in ce products,” *IEEE Consumer Electronics Magazine*, vol. 5, no. 2, pp. 58–62, 2016.
- [65] S. Kono, S. Yonemitsu, and A. Yamawaki, “Development of memory mapped register and bus switch circuit for hardware virtualization in gedy,” in *Proceedings of The 6th IIAE International Conference on Intelligent Systems and Image Processing 2018*, pp. 137–141, The Institute of Industrial Application Engineers, 2018.
- [66] P. Coussy, A. Baganne, and E. Martin, “A design methodology for ip integration,” in *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, vol. 4, pp. IV–IV, 2002.
- [67] J. Zaytoon and B. Riera, “Synthesis and implementation of logic controllers – a review,” *Annual Reviews in Control*, vol. 43, pp. 152–168, 2017.
- [68] AMD, “Amd vivado design suite.” <https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html>, 2026. Accessed: May 18, 2026.

- [69] R. Love, “Kernel korner - sleeping in the kernel.” <https://www.linuxjournal.com/article/4428>, 2003. Accessed: May 14, 2026.
- [70] GTKWave Authors, “Gtkwave documentation.” <https://gtkwave.github.io/gtkwave/>, 2026. Accessed: May 14, 2026.
- [71] FOSSi Foundation, “Librelane.” <https://fossi-foundation.org/librelane>, 2026. Accessed: May 15, 2026.
- [72] N. Sharma, “I2c-in-verilog.” <https://github.com/Nainikas/I2C-in-verilog>, 2026. GitHub repository, Accessed: May 30, 2026.