Design and Implementation of Core Micro-architecture Based on RISC-V ISA

A
thesis submitted
for the award of degree of

DOCTOR OF PHILOSOPHY

Submitted by
Satyajit Bora

Under the supervision of
Prof. Roy Paily

Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati-781039, Assam, India
September 2021

Dedicated
to
My Parents,
Friends and All My Teachers

DECLARATION

*This is to certify that the thesis entitled "**Design and Implementation of Core Micro-architecture Based on RISC-V ISA**", submitted by me to the Indian Institute of Technology Guwahati for the award of the degree of **Doctor of Philosophy** is a bonafide work carried out by me under the supervision of **Prof. Roy P. Paily**. The contents of this thesis, in full or in parts, have not been submitted to any other institute or university for the award of any degree or diploma.*

Signed: _____

Satyajit Bora
Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati-781039, Assam, India

Date: _____

CERTIFICATE

*This is to certify that the thesis entitled "**Design and Implementation of Core Micro-architecture Based on RISC-V ISA**", submitted by **Satyajit Bora (156102010)**, a research scholar in the Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati, for the award of the degree of **Doctor of Philosophy**, is an original research work carried out by him under my supervision and guidance. The thesis has fulfilled all the requirements as per the regulations of the institute and, in my opinion, has reached the standard needed for submission. The contents of this thesis, in full or in parts, have not been submitted to any other institute or university for the award of any degree or diploma.*

Signed: _____

Prof. Roy P. Paily
Department of Electronics and Electrical Engineering
and Centre for Nanotechnology
Indian Institute of Technology Guwahati
Guwahati-781039, Assam, India

Date: _____

Acknowledgement

First of all, I would like to thank GOD for giving me a good and healthy life during my PhD work. I would also like to thank my parents and my sister for their love, support and for never pressurising me to do anything of their interest. Mother and father are the two most beautiful relationships in this world. I have seen my parents sacrificing so many things to give us a better life. I have seen my father managing in two pairs of clothes and saving money for our education. He is the role model of my life. He may not have very high educational qualifications, but the kind of ethics he follows in his life always inspires me to be a person like him. I will always be grateful for my parents' hard work and sacrifices, which made me eligible for a PhD work.

I want to express my deepest and most sincere gratitude to my supervisor Prof. Roy P. Paily, for his fundamental role in my doctoral work. He was always there to guide me in the correct direction in spite of his busy schedule. My heartful thanks to my supervisor for the continuous support and patients towards me. He always tries to motivate us by appreciating our achievements. I am also thankful to him for providing me with all the required facilities to carry out the work.

I am very thankful to my doctoral committee members, Prof. Shaik Rafi Ahamed, Dr Tony Jacob and Dr John Jose, for sparing their precious time to evaluate the progress of my work and providing valuable suggestions. I am also grateful to all my teachers during my course work for their knowledge transfer which helped me a lot in my research work. I would also like to extend my gratitude to the Department of Electronics and Electrical Engineering and its staff members. A special thanks to our lab in charge, Josephine Ma'am, for her help in all aspects.

I would like to thank all my friends from VLSI lab, Vimal Kumar Singh Yadav, Birjit Singha, Pralay Chakrabarty, Rahul Sharma, Shruti Konwar, Prateek Sharma, and Kamlesh Badiyari, for all the technical and non-technical discussions. I am also thankful to all my other friends, Nilakshi, Himakshi, Pranjal, Pallab, Prarthana, Rani ji, Anu ji, and Sangeeta, for being my partner in all the dinner parties and adventure trips. All of you have made my journey a more special one.

I want to thank the Ministry of Electronics & Information Technology (MeitY), the Government of India, for providing the Visvesvaraya PhD scholarship and necessary software/hardware resources to carry out this work.

If I have forgotten to mention anyone's name, please forgive me as a friend.

Guwahati *Satyajit Bora*
October, 2021

Abstract

The demand for low-power and low-cost CPUs is increasing due to cloud computing and IoT endpoint applications. In this work, three 32-bit core micro-architectures are proposed to meet the requirements of modern world devices. The cores are targeted to three different applications: low-power, data-centric IoT devices and high-security. The cores are based on RISC-V ISA with the support of some new custom instructions those help in enhancing the core performance. The low-power core is a four-stage pipelined micro-architecture that supports RV32IM of RISC-V instruction set. The core is verified on simulation as well as on Xilinx Virtex-7 FPGA platform. This core could achieve a Dhrystone benchmark score of 1.71 DMIPS per MHz which is higher than ARM Cortex-M3 (1.50 DMIPS per MHz) and ARM Cortex-M4 (1.52 DMIPS per MHz). The CoreMark benchmark is also tested on this core, and it gives 4.13 CoreMark per MHz. The physical design result of the core using commercial tools shows that it can achieve a maximum frequency of 198.02 MHz with 0.036 mm^2 area and 17.36 μW/MHz power requirement at UMC 40 nm technology node. The core consumes a dynamic power of 19.75 μW/MHz at UMC 90 nm, which is 36% and 40% better than ARM Cortex-M3 and Cortex-M4, respectively and also lower than many other cores. The results show that this core can outperform many existing commercial and open-source cores.

The IoT core is based on RISC-V ISA with the support of eleven proposed custom instructions for enhancing the performance in data processing algorithms, convolution, matrix multiplication, digital filters, cryptographic kernels, etc. The synthesis result at 65 nm shows that the core requires an additional 7% resources due to the integration of the proposed custom instructions. However, the core performance is improved by 1.74×due to the use of these instructions. The core is also capable of working up to a maximum frequency of 328 MHz. The high-security core is integrated with a dedicated hardware unit for AES encryption/decryption. There are six proposed instructions to make use of this hardware unit. The core can perform an AES encryption in 63 clock cycles, whereas the same encryption requires 54265 and 19258 clock cycles in the proposed low-power and IoT cores, respectively. At 65 nm technology node, the AES throughput of the core is found to be 0.71 Gbps. The core clock frequency is limited at 352 MHz.

During the implementation of the cores, it has been observed that there is very little research in hardware implementation of binary divider. Therefore, design and implementation of a novel binary divider are also presented in this work. The work presents an analysis of the proposed divider for different data widths (8, 16, 32, 64, 128) and different radix values (4, 8, 16). The area, power, delay of the designs are estimated at 40nm technology node. The analysis is helpful for designers in determining the correct divider circuit for their designs.

CONTENTS

Chapter 1: Introduction 1
1.1 Introduction . . . 2
1.2 RISC-V ISA . . . 2
1.3 Motivation . . . 5
1.4 Design Objectives . . . 6
1.5 Thesis Overview . . . 7

Chapter 2: Literature and Methodology 9
2.1 Introduction . . . 10
2.2 Shakti . . . 10
2.3 PULPino . . . 13
2.4 Rocket SoC . . . 15
2.5 Existing Core Results . . . 17
2.5.1 Performance Measurement . . . 18
2.5.2 Power and Area Measurement . . . 19
2.5.3 Results . . . 20
2.6 Methodology . . . 21
2.6.1 Software Verification . . . 22
2.6.2 Hardware Verification . . . 23
2.6.3 Evaluation . . . 24
2.7 Conclusion . . . 24

Chapter 3: Design and Implementation of RISC-V Core for Low-power Applications 25
3.1 Introduction . . . 26
3.2 Micro-architecture . . . 27
3.2.1 Instruction Fetch . . . 28
3.2.2 Instruction Decode . . . 28
3.2.3 Instruction Execute . . . 28
3.2.4 Register Write . . . 33
3.3 Results and discussion . . . 33

3.3.1 ASIC . . . 34
3.3.2 FPGA . . . 35
3.4 Conclusion . . . 37

Chapter 4: Design and Implementation of Adaptive Binary Divider for Fixed-point and Floating-point Numbers. 38
4.1 Introduction . . . 39
4.2 Literature Review . . . 40
4.3 Algorithm for Fixed-point Binary Divider . . . 40
4.4 Hardware Architecture of Fixed-point Binary Divider . . . 42
4.5 Single-precision Floating-point Divider . . . 44
4.6 FPGA Synthesis . . . 46
4.7 ASIC Synthesis . . . 47
4.8 Core Integration . . . 49
4.9 Conclusion . . . 51

Chapter 5: Design and Implementation of RISC-V Core for Data-centric IoT Applications 53
5.1 Introduction . . . 54
5.2 ISA and its Extension . . . 55
5.3 Micro-architecture . . . 56
5.3.1 Pipeline . . . 57
5.3.2 ALU . . . 57
5.3.3 MAC . . . 58
5.4 Results and Discussion . . . 60
5.4.1 ASIC . . . 60
5.4.2 FPGA . . . 60
5.4.3 Performance Measurement . . . 61
5.5 Conclusion . . . 64

Chapter 6: Design and Implementation of RISC-V Core for High-security Application 65
6.1 Introduction . . . 66
6.2 ISA and its Extension . . . 66
6.3 Micro-architecture . . . 67
6.3.1 Instruction Fetch . . . 68
6.3.2 Instruction Decode . . . 68
6.3.3 Instruction Execute . . . 68
6.4 Results and Discussion . . . 74
6.4.1 ASIC . . . 75
6.4.2 FPGA . . . 76
6.5 Comparison of Proposed IoT and High-security ISA Extensions . . . 78
6.6 Conclusion . . . 79

Chapter 7: Conclusion 80
7.1 Conclusion . . . 81
7.2 Summary of Contributions . . . 81
7.3 Future Directions . . . 82

LIST OF TABLES

2.1 Evaluation of RISC-V cores. 21

3.1 Synthesis results of multiplier circuits at UMC 40 nm technology node. . . . 30
3.2 Synthesis results of divider circuits at UMC 40 nm technology node. 31
3.3 ASIC post-layout results of the proposed low-power core at UMC 90 nm and 40 nm technology nodes. 34
3.4 FPGA synthesis result of the proposed low-power core. 35

4.1 FPGA synthesis result of 32-bit radix-16 divider. 46
4.2 ASIC synthesis results of 32-bit and 16-bit radix-16 divider. 47
4.3 Power, area, delay and latency of the proposed divider circuit for different data width and radix values at 40 nm. 48
4.4 FPGA synthesis result of the proposed designs after core integration. 50
4.5 ASIC implementation results of the proposed designs after core integrating. . 51

5.1 Summation of 10 numbers at instruction level with and without the use of custom instructions. 58
5.2 ASIC post-layout result of the proposed IoT core at UMC 65 nm technology node. 60
5.3 Performance improvement with the use of proposed ISA extension. 62

6.1 Synthesis results of AES hardware implementations at UMC 65 nm technology node. 71
6.2 Area occupied by different submodules of the AES-L40 design. 74
6.3 ASIC post-layout result of the proposed high-security cores. 75
6.4 FPGA synthesis results of the proposed high-security cores. 77
6.5 ASIC post-layout result of core1 (RV32IM), core2 (RV32IM+EX1), and core3 (RV32IM+EX2) at UMC 65 nm technology node. 78
6.6 FPGA synthesis result of core1 (RV32IM), core2 (RV32IM+EX1), and core3 (RV32IM+EX2) for Xilinx Virtex-7. 79

7.1 Summary of the proposed low-power, IoT and high-security RISC-V cores. . 82

LIST OF FIGURES

2.1 Shakti E class [1]. 11
2.2 Shakti C class [1]. 12
2.3 Block diagram of RI5CY core [2]. 14
2.4 Memory map of PULPino [2]. 15
2.5 Fisrt stage of rocket [3]. 16
2.6 Last four stages of rocket [3]. 17
2.7 Xilinx FPGA boards. 19
2.8 Design, verification, and evaluation process of the proposed cores. 22
2.9 Functional verification process of the propose cores. 22
2.10 FPGA hardware verification process of the proposed cores. 23
2.11 Xilinx Virtex-7 vc707 FPGA. 23
2.12 Evaluation of the proposed cores. 24

3.1 Pipeline stages of the proposed micro-architecture. 27
3.2 Micro-architecture of Instruction Execute (IE) stage. 29
3.3 Addition of partial products of 33×33 signed multiplier. 30
3.4 Block diagram of 32-bit Baugh Wooley signed/unsigned Multiplier. 31
3.5 32-bit radix-16 unsigned divider. 32
3.6 Comparison of Dhrystone and CoreMark scores with other commercial cores [4, 5]. 34
3.7 Distribution of FPGA power and hardware resources for the proposed low-power core. 36
3.8 FPGA verification set-up. 37

4.1 Radix-16 unsigned divider, n is 32 for a 32-bit divider and 24 for a 24-bit divider which is used in floating-point divider (Section 4.5). 42
4.2 Iteration count determination circuit for a 32-bit dividend in radix-16 implementation 43
4.3 Constant multiplier in radix-16 divider 43
4.4 IEEE-754 single-precision floating-point format 44
4.5 Block diagram of the proposed floating-point divider 46

4.6 Area, power, delay and latency of the proposed architecture vs. data width for different radix values. 49

5.1 Extended custom instructions. 56
5.2 4-stage pipeline of the proposed core 57
5.3 Block diagram of MAC unit 59
5.4 Block diagram of 32-bit radix-8 divider 59
5.5 Distribution of FPGA power and hardware resources for the proposed IoT core. 61
5.6 Instructions per cycle (IPC) while executing various algorithms. 62
5.7 (a) C and (b) assembly code of convolution without using the proposed extension. 63
5.8 (a) C and (b) assembly code of convolution using the proposed extension. . . 63
5.9 (a) C and (b) assembly code of sum without using the proposed extension. . 64
5.10 (a) C and (b) assembly code of sum using the proposed extension. 64

6.1 Extended custom instructions for AES encryption and decryption. 67
6.2 4-stage pipeline of the proposed core. 67
6.3 Addition of partial products of 33×33 signed multiplier. 69
6.4 Block diagram of multiplier unit. 70
6.5 Block diagram of 32-bit radix-8 divider. 70
6.6 Area, power, delay, and latency of the AES encryption hardware designs. . . 72
6.7 Block diagram of AES-L40 encryption unit. 73
6.8 Block diagram of AES-L40 decryption unit. 73
6.9 ShiftRows, SubBytes, and MixColumn stages. 74
6.10 Distribution of FPGA power and hardware resources for the proposed core with AES-L10. 76
6.11 Distribution of FPGA power and hardware resources for the proposed core with AES-L40. 77
6.12 Comparison of area, delay, and power of core1 (RV32IM), core2 (RV32IM+EX1), and core3 (RV32IM+EX2). 78
6.13 Comparison of area, delay, and power of core1 (RV32IM), core2 (RV32IM+EX1), and core3 (RV32IM+EX2) on Xilinx Virtex-7 FPGA. . . . 79

LIST OF ACRONYMS

ABI	Application Binary Interface
AES	Advanced Encryption Standard
AHB	Advanced High-performance Bus
ALU	Arithmetic Logic Unit
AMBA	Advanced Micro-controller Bus Architecture
ASIC	Application Specific Integrated Circuit
CMOS	Complementary Metal-oxide Semiconductor
CPU	Central Processing Unit
CSR	Control and Status Register
DBP	Dynamic Branch Prediction
DCT	Discrete Cosine Transform
DDR	Double Data Rate
DMIPS	Dhrystone MIPS
DSP	Digital Signal Processing
FA	Full Adder
FF	Flip-flop
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FPU	Floating-point Unit
GPIO	General Purpose Input/Output

HA	Half Adder
HDL	Hardware Description Language
ID	Instruction Decode
IE	Instruction Execute
IF	Instruction Fetch
IoT	Internet of Things
ISA	Instruction Set Architecture
LSB	Least Significant Bit
LUT	LookUp Table
MIPS	Millions of Instructions Per Second
MMU	Memory Management Unit
MSB	Most Significant Bit
MSO	Most Significant One
PCIe	Peripheral Component Interconnect Express
RISC	Reduced Instruction Set Computer
RW	Register Write
SHA-1	Secure Hash Algorithm 1
UART	Universal Asynchronous Receiver-transmitter
UMC	United Microelectronics Corporation

CHAPTER

1

INTRODUCTION

Contents

1.1 Introduction 2
1.2 RISC-V ISA 2
1.3 Motivation 5
1.4 Design Objectives 6
1.5 Thesis Overview 7

1.1 Introduction

The era of increasing processor performance through CMOS scaling is coming to an end as CMOS technology is approaching physical limitations. So, processor designers are forced to look into the other aspects of processor design. The two most important aspects of processor design are Instruction Set Architecture (ISA) and micro-architecture. There are mainly two ISAs that are dominating the current processor industry. ARM, which is used in portable devices like mobile and x86, which is used in personal computers and servers. These ISAs may not be the best ISAs developed ever, but with time, they had become so popular that no other ISA could replace them. Also, these ISAs have proprietary issues. Because of this, individual hardware designers can not use them for their designs. To overcome these issues and in order to support computer architecture research, education as well as industry implementation, a new ISA called RISC-V [6] had been developed by UC, Berkeley. The main advantage of this ISA is that it is free, and because of which anyone can use it for their research. An open-source instruction set architecture (ISA) is a desirable starting point for processor design, as it can potentially decrease dependence from a single IP provider and cut costs while, at the same time, allowing freedom for application-specific instruction extensions.

Based on this RISC-V ISA, many cores have been developed, and many are under development. Rocket [7, 8] is the first approach towards RISC-V micro-architecture by the same researchers who have developed the RISC-V instruction set. It is a 6-stage single-issue in-order pipeline processor based on RV64G ISA. The authors achieved 10% higher performance in DMIPS/MHz score compared to Cortex-A5 and 49% area improvement. Pulpino [9–11] is an open-source microcontroller system based on RV32IMC RISC-V core developed at ETH Zurich. It is a platform organised in clusters of RISC-V cores sharing a tightly coupled data memory. Shakti-F [12] is another approach where researchers are trying to develop a reliable processor for radiation prone environments like nuclear and space applications. Some more variants of Shakti like Shakti-E, Shakti-C, Shakti-I are also under development [1, 13]. Few more examples of RISC-V core are ORCA [14], mriscv [15], VexRiscv [16], LowRISC [17], RI5CY [18] etc.

1.2 RISC-V ISA

RISC-V (pronounced as 'risk-five') was originally designed to support computer architecture research and education, but it is now becoming a standard free and open architecture for industry implementations. Some of the features of RISC-V are:

- A completely open ISA that is freely available to academia and industry.
- Suitable for direct native hardware implementation, not just simulation or binary translation.
- An ISA that avoids over-architecting for a particular micro-architecture style (e.g., micro coded, in-order, decoupled, out-of-order) or implementation technology (e.g., full-custom, ASIC, FPGA), but which allows efficient implementation in any of these.
- Separated into a small base integer ISA, usable by itself as a base for customised accelerators or for educational purposes and optional standard extensions, to support general-purpose software development.

- Supports extensive user-level ISA extensions and specialised variants.
- Both 32-bit and 64-bit address space variants.
- Support for highly-parallel multi-core or many core implementations, including heterogeneous multiprocessors.
- Optional variable-length instructions to both expand available instruction encoding space and to support an optional dense instruction encoding for improved performance, static code size and energy efficiency.

The most significant and obvious benefit of using an existing commercial ISA is the large and widely supported software ecosystem, both development tools and ported applications. Other benefits include the existence of large amounts of documentation and tutorial examples. However, these benefits are smaller in practice and do not outweigh the disadvantages, which are:

- **Commercial ISAs are proprietary**
  Except for SPARC V8, which is an open IEEE standard, most owners of commercial ISAs carefully guard their intellectual property and do not welcome freely available competitive implementations. This is much less of an issue for academic research and teaching using only software simulators but has been a major concern for groups wishing to share actual RTL implementations.
- **Commercial ISAs are only popular in certain market domains**
  The most obvious example is that the ARM architecture is not well supported in the server space, and the Intel x86 architecture (or for that matter, almost every other architecture) is not well supported in the mobile space, though both Intel and ARM are attempting to enter each other's market segments. Another example is ARC and Tensilica, which provide extensible cores but are focused on the embedded space. This market segmentation dilutes the benefit of supporting a particular commercial ISA as in practice, the software ecosystem only exists for certain domains and has to be built for others.
- **Commercial ISAs come and go**
  Previous research infrastructures have been built around commercial ISAs that are no longer popular (SPARC, MIPS) or even no longer in production (Alpha). These lose the benefit of an active software ecosystem, and the lingering intellectual property issues around the ISA and supporting tools interfere with the ability of interested third parties to continue supporting the ISA. An open ISA might also lose popularity, but any interested party can continue using and developing the ecosystem.
- **Popular commercial ISAs are complex**
  The dominant commercial ISAs (x86 and ARM) are both very complex to implement in hardware to the level of supporting common software stacks and operating systems. Worse, nearly all the complexity is due to bad, or at least outdated, ISA design decisions rather than features that truly improve efficiency.
- **Commercial ISAs alone are not enough to bring up applications**
  Most applications need a complete ABI (application binary interface) to run, not just the user-level ISA. Most ABIs rely on libraries, which in turn rely on operating system

support. To run an existing operating system requires implementing the supervisor-level ISA and device interfaces expected by the OS. These are usually much less well-specified and considerably more complex to implement than the user-level ISA. Popular commercial ISAs were not designed for extensibility. The dominant commercial ISAs were not particularly designed for extensibility, and as a consequence, have added considerable instruction encoding complexity as their instruction sets have grown. Companies such as Tensilica (acquired by Cadence) and ARC (acquired by Synopsys) have built ISAs and toolchains around extensibility but have focused on embedded applications rather than general-purpose computing systems.

The ISA is perhaps the most important and basic interface in a computing system, and there are many advantages of RISC-V over proprietary ISAs. The dominant commercial ISAs are based on instruction set concepts that were already well known over 30 years ago. Software developers should be able to target an open standard hardware target, and commercial processor designers should compete on implementation quality.

Instruction Set Overview

The RISC-V ISA [19, 20] is designed as a base integer ISA, which must be present in any implementation, plus optional extensions to the base ISA. The base integer ISA is very similar to that of the early RISC processors except with no branch delay slots and with support for optional variable-length instruction encoding. The base is carefully restricted to a minimal set of instructions sufficient to provide a reasonable target for compilers, assemblers, linkers and operating systems (with additional supervisor-level operations) and so provides a convenient ISA and software toolchain 'skeleton' around which more customised processor ISAs can be built. Each base integer instruction set is characterised by the width of the integer registers and the corresponding size of the user address space. There are two primary base integer variants, RV32I and RV64I, which provide 32-bit or 64-bit user-level address spaces, respectively. Hardware implementations and operating systems might provide only one or both of RV32I and RV64I for user programs. RV32E, a subset variant of the RV32I base instruction set, has been added to support small micro-controllers.

RISC-V has been designed to support extensive customisation and specialisation. The base integer ISA can be extended with one or more optional instruction-set extensions, but the base integer instructions cannot be redefined. RISC-V instruction-set extensions are divided into standard and non-standard extensions. Standard extensions should be generally useful and should not conflict with other standard extensions. Non-standard extensions may be highly specialised or may conflict with other standard or non-standard extensions. Instruction-set extensions may provide slightly different functionality depending on the width of the base integer instruction set.

Standard extensions are provided for integer multiply, divide, atomic operations, single and double-precision floating-point arithmetic to support more general software development. The base integer ISA is named 'I' (prefixed by RV32 or RV64 depending on integer register width) and contains integer computational instructions, integer loads, integer stores and control-flow instructions and is mandatory for all RISC-V implementations. The standard integer multiplication and division extension is named 'M' and adds instructions to multiply and divide values held in the integer registers. The standard atomic instruction extension, denoted by 'A', adds instructions that atomically read, modify and write memory for inter-processor synchronisation.

The standard single-precision floating-point extension, denoted by 'F', adds floating-point registers, single-precision computational instructions, and single-precision loads and stores. The standard double-precision floating-point extension, denoted by 'D', expands the floating-point registers and adds double-precision computational instructions, loads, and stores. An integer base plus these four standard extensions ('IMAFD') is given the abbreviation 'G' and provides a general-purpose scalar instruction set. RV32G and RV64G are currently the default target of RISC-V compiler toolchains.

1.3 Motivation

During the last few years, there has been a massive escalation in the requirement of modern Internet of Things (IoT) endpoint devices. These devices usually capture real-world data using sensors and transmit the data to the cloud for processing. The applications of these devices include processing, elaboration and transmission of biomedical signals such as ECG, EEG or EMG (e-health), signals coming from low-power imagers or cameras (smart surveillance), vibrations or audio signals in the factories (smart industry), or just low-bandwidth information such as temperature or humidity (smart monitoring). These applications require high computational capabilities with extreme energy efficiency and low cost. To meet these requirements, the IoT endpoint devices need to equip with highly efficient processors, low-power sensors, specialised hardware units, wireless transmitter/receiver, memory, etc.

The processor is the most crucial part of an IoT endpoint device as it can decide the overall power, area, and cost. Due to the variety of computational requirements found in the application workloads, different cores can perform better in different parts of the workload. These cores vary from small, ultra-low-power cores with limited performance but very low area and power; to cores integrated with Digital Signal Processing (DSP) capabilities targeting near-sensor data processing. Commercial vendors also provide a wide range of different core architectures to cover the large variety of energy, performance, power and area constraints. For example, ARM provides cores for low-power and low-area, as well as superscalar cores for high performance computing. Heterogeneous multi-core architectures is another approach in commercial systems for IoT [21]. A tightly coupled cluster of DSP processors operating at low voltage can also significantly increase energy efficiency [22].

Most modern IoT devices are battery-powered, and therefore they need to work within an extremely tight power budget. Since these devices are built around a microprocessor, the processing power has to be as low as possible to extend the battery life of the devices. At the same time, the processor should also have a good processing speed. Otherwise, it will not be able to synchronise with modern world systems. Therefore, we are focused on implementing a low-power processor with good performance capabilities in the initial phase of this work.

If a device is portable, it has to communicate with other devices through wireless transmission. In such a device, wireless transmission consumes most of its power, reducing the battery backup. Therefore it is highly preferable to reduce the amount of data to be transmitted. The data reduction can be made using various Digital Signal Processing (DSP) algorithms. DSP algorithms extract the important features from a data set, and the extracted features are transmitted over the wireless medium. This eventually reduces the transmission power but also increases the load on the processing unit. There are many examples in the literature to increase the processing capabilities of the processor. For example, ARM Cortex-M4 [23] is a low-power, highly efficient processor with digital signal processing (DSP) capabilities. In the second phase of this work, we concentrate on the data processing capabilities of the

processor.

Performing analytics such as feature extraction or classification directly on end-node devices does not address the security concerns of sensitive data. A distilled data, stored or sent over the network at several stages of the analytics pipeline, is more privacy-sensitive than the raw data stream [24]. Protecting sensitive data at the boundary of the on-chip analytics engine is a way to address these security issues. Cryptographic algorithms such as AES, SHA-1, etc., are used for data protection. However, these algorithms significantly increase the processor's workload, which can easily be 100 to 1000 processor instructions per encrypted byte. Therefore, the third part of this work focuses on the security of data.

1.4 Design Objectives

In this work, our plan is to develop three cores based on RISC-V ISA. The cores are targeted to three different applications low-power, data-centric IoT and high-security. The first core is a low-power core, which supports the basic integer instructions (RV32I) and multiplication-division extension (M) of RISCV ISA. The core can also be used as a base for developing high-performance cores. The second core will be designed to perform better in data-centric Internet of Things (IoT) applications. The plan is to integrate FPUs (Functional Processing Unit) and DSP (Digital Signal Processing) units to improve the core performance in the mentioned applications. The third core will be designed for high-security devices. The Advanced Encryption Standard (AES) is a standard technique used for electronic data encryption. In the third core, dedicated hardware will be integrated into the core to boost the AES encryption and decryption.

In order to realise the above mentioned objectives, we have developed a framework/methodology for the work. The starting point is to build a software environment for the functional verification of RISC-V cores. The second step is to build core micro-architectures using Verilog HDL and verify the functionality of the designs. If the functionality verification shows incorrect results, then we need to go back to the design step and debug the HDL design. After successful verification of the functionality, the designs are verified on different FPGA hardware platforms. If the designs work successfully on FPGA, their ASIC implementations are carried out using industrial tools. The ASIC implementation gives an estimation of the power, area, and delay of the designs. Benchmark programs and application-specific algorithms are also needed to be executed on the designs to measure the performance. In summary, the point by point objectives of this work are listed below:

1. Development of software/hardware environment for RISC-V core design.
    - Development of software environment using python and C++.
    - FPGA hardware verification set-up.
    - Physical design flow. (ASIC)
2. Design and implementation of first core targeted to low-power applications.
    - Support of full RV32IM instructions.
    - Software verification of the HDL design.
    - Verification on FPGA platform.
    - Estimation of area, power, delay and performance in ASIC.

3. Design and implementation of fixed-point and floating-point binary divider.
    - Design of fixed-point divider and extending it to floating-point divider.
    - Estimation of area, power, and delay for data widths: 8, 16, 32, 64, 128
    - Estimation of area, power, and delay for radix values: 4, 8, 16
4. Design and implementation of second core targeted to data-centric IoT applications.
    - Support of full RV32IM instructions.
    - Support of 11 new instructions to enhance performance in digital data processing.
    - Software verification of the HDL design.
    - Verification on FPGA platform.
    - Estimation of area, power, delay and performance in ASIC.
5. Design and implementation of third core targeted to high-security application.
    - Support of full RV32IM instructions.
    - A dedicated hardware for faster AES encryption/decryption.
    - Support of six new instructions to control the AES unit.
    - Software verification of the HDL design.
    - Verification on FPGA platform.
    - Estimation of area, power, delay and performance in ASIC.

1.5 Thesis Overview

The contributions of this thesis are organised into five major chapters. The following chapter, chapter 2, includes a survey of existing RISC-V cores and the methodology used for the design of proposed cores. The methodology explains the implementation of a software verification environment. This software environment is later used for debugging the HDL designs. Chapter 2 also gives an overview of the FPGA hardware verification process and the ASIC implementation process.

Chapter 3 explains the implementation of the first core micro-architecture. This core is designed for low-power applications. The core supports the base integer instructions (RV32I) and the multiplication-division extension (M) of RISC-V ISA. This chapter reports the area, power, and delay of the core in FPGA as well as in ASIC. The results of two benchmark programs, Dhrystone and CoreMark, are also reported to give an idea about the performance of the core.

A fixed-point and a floating-point binary dividers implementation is proposed in chapter 4. The chapter first explains the design and implementation of a novel fixed-point binary divider. The same idea is then extended to a floating-point binary divider. A comparison of the proposed fixed-point binary divider with different data widths (8, 16, 32, 64, 128) and different radix values (4, 8, 16) is also reported in this chapter. This would eventually help designers in adapting the correct divider circuit for their design.

Chapter 5 discusses the design and implementation of the second core targeted to data-centric IoT applications. This core is integrated with a proposed ISA extension, EX1. Chapter 5 first explains the instructions included in EX1, and then the micro-architectural changes

are discussed in detail. The micro-architectural changes include MAC unit integration, ALU modification, elimination of multiplier unit, etc. The FPGA and ASIC results of the core are reported in the result section of the core. A performance measure of the core is also reported for different algorithms related to data-centric applications such as FIR, DCT, SHA-1, AES, etc.

Chapter 6 discusses the third and the last core of this work, which is targeted to high-security applications. The core has a dedicated hardware unit for AES encryption/decryption, and it supports a proposed ISA extension, EX2, to control the AES hardware unit. The chapter first explains the instructions proposed in the EX2 and then the micro-architectural optimisations. This chapter also includes a comparison of five different implementations of AES hardware encryption units. The result section of the chapter reports the area, power, and delay of the core in FPGA and ASIC. Towards the end of this chapter, a comparison of three different ISA supports, RV32IM, RV32IM+EX1 and RV32IM+EX2, is also reported and discussed.

CHAPTER

2

LITERATURE AND METHODOLOGY

Contents

2.1 Introduction 10
2.2 Shakti 10
2.3 PULPino 13
2.4 Rocket SoC 15
2.5 Existing Core Results 17
2.5.1 Performance Measurement 18
2.5.2 Power and Area Measurement 19
2.5.3 Results 20
2.6 Methodology 21
2.6.1 Software Verification 22
2.6.2 Hardware Verification 23
2.6.3 Evaluation 24
2.7 Conclusion 24

2.1 Introduction

In the current world, devices not only need to work under extremely tight power envelope of a few milliwatts but also need to be flexible in their computing capabilities [11]. The designers claim that RISC-V CPUs can achieve higher speeds and smaller, lower power and cost electronics than some comparable commercial CPUs [7,8,12,25–31]. In contrast to most ISAs, the RISC-V ISA can be freely used for any purpose, permitting anyone to design, manufacture and sell RISC-V chips and software. While not the first open ISA, it is significant because it is designed to be useful in modern computerised devices such as warehouse-scale cloud computers, high-end mobile phones and the smallest embedded systems. Such uses demand that the designers consider both performance and power efficiency [32, 33]. The instruction set also has a substantial body of supporting software, which fixes a usual weakness of new instruction sets. The RISC-V ISA is adopted by many CPU designers, and the development of different micro-architectural styles is going on. Some of these efforts (Shakti, PULPino, Rocket) are explained in the following part of this chapter.

2.2 Shakti

The Shakti [13] processor project is building different variants of processors based on the RISC-V ISA from UC Berkeley (www.riscv.org). The project is developing a complete reference SoC for each family, which will serve as an exemplar for that category of processor. While the cores and most of the SoC components (including bus and interconnect fabrics) is open source, some standard components like PCIe controller, DDR controller and PHY IP are 3rd party IP. All sources are licensed using a 3rd party BSD license and royalty and patent free (as far as IIT-Madras is concerned). While the primary focus is research, the SoCs are being designed to be competitive with commercial processors with respect to features, silicon area, power profile and frequency. We will be discussing two variants (C-class, E-class) here, which were available during the study. Two variants of Shakti, E class and C class, are discussed in the following part of this section.

Shakti E class

Shakti E class micro-controller supports RISC-V base integer ISA - RV32IMA. It supports basic arithmetic instructions such as add, sub, mul, div, etc., basic logical instructions such as and, or, xor etc., atomic instructions, basic branch, jumps, load/store instructions. It also supports RVC (RISC-V Compressed ISA), i.e. 16-bit instructions for efficient resource utilisation for low-end applications. The design has the following features:

- 32-bit 3 stage in-order core aimed at 10 - 50 MHz micro-controller variants.
- Optional memory protection.
- Very low-power static design.
- Variants with compressed/reduced ISA support.
- Bus - AHB Lite.

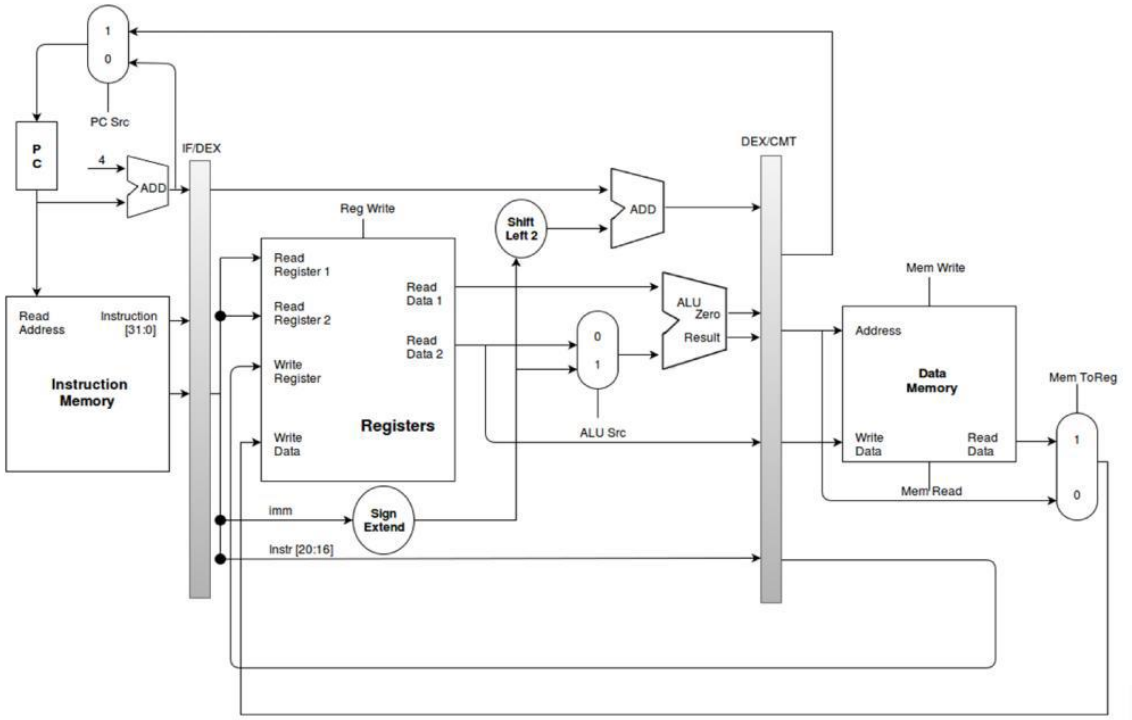


Figure 2.1: Shakti E class [1].

Figure 2.1 shows the block diagram of Shakti E class. It is a three-stage pipeline, in-order execution architecture. The three pipelining stages are Instruction Fetch Unit, Decode & Execution Unit and Commit Unit. For interconnect, AHB (Advanced High-performance Bus) is used instead of AXI4. AHB uses 33% less power than AXI4 while their latency and throughput are comparable [34]. It is targeted for low end embedded applications like GPS navigation systems, storage controllers, FPGA based controllers, designs running at 50-100 MHz Max Frequency. It is also targeted to minimal computation applications. For security purposes, it can be used as a co-processor to ensure security and privileged access to main processor peripherals. It can also be used as a reference RTL model for teaching the basics of computer architecture in academia.

Shakti C class

Shakti c-32 class is 32-bit 5 stage pipeline with branch prediction. The five stages are Instruction Fetch Unit, Instruction Decode Unit, Operand Fetch and Execute Unit (Branches are handled here itself), Memory Access Unit and Write Back Unit. It supports all integer instructions of RV32I and M extension. The features of the design are:

- 32-bit and 64-bit 3-8 stage in-order core aimed at 10MHz - 1GHz controller requirements.
- Optional memory protection and MMU.
- Very low-power static design variants.
- Fault Tolerant variants for ISO26262 applications.
- IoT variants will have compressed/reduced ISA support.

- Optional FPU, VPU.
- Bus - AHB variants.

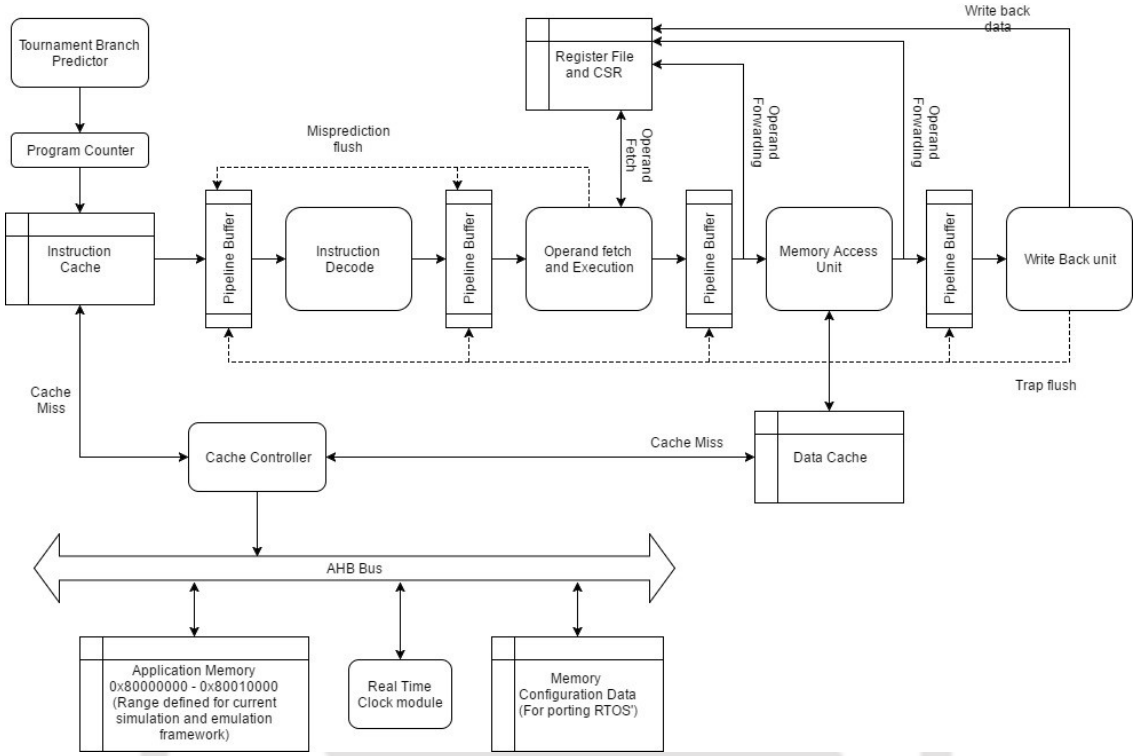


Figure 2.2: Shakti C class [1].

Figure 2.2 shows the 5 stages of Shakti C-32. The branch prediction unit provides the next Program Counter (PC) that needs to be fetched. The Instruction fetch stage sends the PC to the Instruction Cache (I-Cache). If there is a hit in the I-Cache, then the respective instruction is returned to the instruction fetch stage. In case of a miss, the address of the corresponding line in the cache is sent to the cache controller, which in turn fetches the line from the application memory through the AHB Bus.

The fetched instruction is sent to the decode stage, where operand addresses, type of instruction etc., are deduced and sent over to the next stage. In this stage, the operands are fetched. During operand fetch, the operand forwarding lines from the following stages are polled (in priority order) to check if any of them can provide the necessary operands for the current instruction. If the current instruction is independent of the instructions in the following stages, then it simply fetches the operands from the Register File. Once the operands are available, the same stage executes the instruction. For loads/stores, this stage simply calculates the effective address and sends it to the next stage - Memory Access Unit. In the case of branch instructions, the prediction of the Branch prediction unit is validated. In case of a miss prediction, the preceding two pipeline buffers are flushed (i.e. data is removed), and the PC is reset to the required address.

If the instruction is not a Load/Store instruction, then it bypasses any operation in the memory access unit and is buffered into the write-back unit. However, in the case of a Load/Store, the address and data are passed to the Data cache to perform the respective write/read from memory. Once the data cache has responded with valid data/acknowledgement, the instruction then passes over to the write-back stage. In the write-back stage, the following checks happen in the specified order.

- Are there any pending interrupts?
- Has the current instruction generated any exceptions?
- If it is a system instruction, does it make valid access to the CSR registers?

If any of the above conditions evaluates to be true, then a trap is taken, which causes the pipeline buffers to be flushed, resetting the PC. In case if no trap is taken, then the instruction commits and updates the respective register in the Register File and terminates.

2.3 PULPino

PULPino [35] is an open-source micro-controller system based on a small 32-bit RISC-V core developed at ETH Zurich. It implements several ISA extensions such as hardware loops, post-incrementing load and store instructions, ALU and MAC operations, which increase the efficiency of the core in low-power signal processing applications. To allow embedded operating systems such as FreeRTOS to run, a subset of the privileged specification is supported. When the core is idle, the platform can be put into a low-power mode, where only a simple event unit is active, and everything else is clock-gated and consumes minimal power (leakage). A specialised event unit wakes up the core in case an event/interrupt arrives. For communication with the outside world, PULPino contains a broad set of peripherals, including I2S, I2C, SPI and UART. The platform internal devices can be accessed from outside via JTAG and SPI, which allows pre-loading RAMs with executable code. In standalone mode, the platform boots from an internal boot ROM and loads its program from an external SPI flash.

The PULPino platform is available for RTL simulation as well FPGA. PULPino has been taped out as an ASIC in UMC 65 nm in January 2016 [9]. It has full debug support on all targets. In addition, it supports extended profiling with source code annotated execution times through KCacheGrind in RTL simulations.

RI5CY core

The core used in PULPino platform is RI5CY. It is a 4-stage in-order 32b RISC-V processor core. The ISA of RI5CY was extended to support multiple additional instructions, including hardware loops, post-increment load and store instructions and additional ALU instructions that are not part of the standard RISC-V ISA. Figure 2.3 shows a block diagram of the core. RI5CY supports the following instructions:

- Full support for RV32I Base Integer Instruction Set
- Full support for RV32C Standard Extension for Compressed Instructions
- Full support for RV32M Integer Multiplication and Division Instruction Set Extension
- PULP specific extensions
- Post-Incrementing load and stores
- Multiply-Accumulate extensions
- ALU extensions
- Hardware Loops

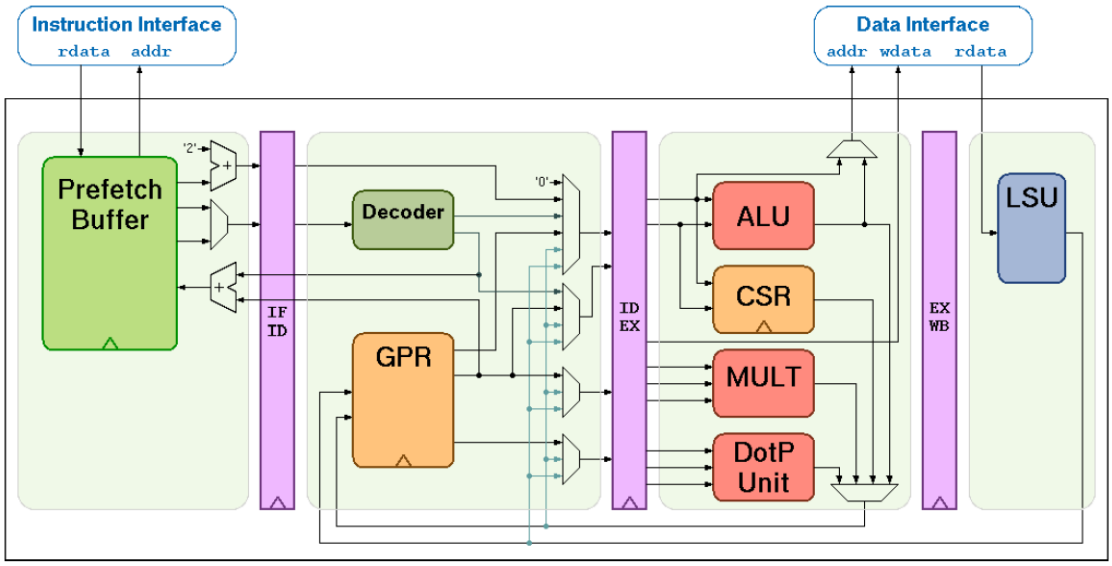


Figure 2.3: Block diagram of RI5CY core [2].

Instruction Fetch

The instruction fetcher of the core is able to supply one instruction to the ID stage per cycle if the instruction cache or the instruction memory is able to serve one instruction per cycle. The instruction address must be half-word-aligned due to the support of compressed instructions. It is not possible to jump to instruction addresses that have the LSB bit set. For optimal performance and timing closure reasons, a prefetcher is used, which fetches instructions from the instruction memory or instruction cache. There are two prefetch flavours available:

- 32-bit word prefetcher. It stores the fetched words in a FIFO with three entries.
- 128-bit cache line prefetcher. It stores one 128-bit wide cache line plus 32-bit to allow for cross-cache line misaligned instructions.

Load-Store-Unit (LSU)

The LSU of the core takes care of accessing the data memory. Load and Store on words (32-bit), half-words (16-bit) and bytes (8-bit) are supported. The LSU is able to perform misaligned accesses, meaning accesses that are not aligned on natural word boundaries. However, it needs to perform two separate word-aligned accesses internally. This means that at least two cycles are needed for misaligned loads and stores.

The protocol that is used by the LSU to communicate with a memory works as follows: The LSU provides a valid address in data_addr_o and sets data_req_o high. The memory then answers with a data_gnt_i set high as soon as it is ready to serve the request. This may happen in the same cycle as the request was sent or any number of cycles later. After a grant is received, the address may be changed in the next cycle by LSU. In addition, the data_wdata_o, data_we_o and data_be_o signals may be changed as it is assumed that the memory has already processed and stored that information. After receiving a grant, the memory answers with a data_rvalid_i set high if data_rdata_i is valid. This may happen one or more cycles after the grant has been received. Note that data_rvalid_i must also be set when a write was performed, although the data_rdata_i has no meaning in this case.

Post-Incrementing Load and Store Instructions

Post-incrementing load and store instructions perform a load/store operation from/to the data memory while at the same time increasing the base address by the specified offset. For memory access, the base address without offset is used. Post-incrementing load and stores reduce the number of required instructions to execute code with regular data access patterns, which can typically be found in loops. These post-incrementing load/store instructions allow the address increment to be embedded in the memory access instructions and get rid of separate instructions to handle pointers. Coupled with hardware loop extension, these instructions allow to reduce the loop overhead significantly.

Memory Map

Figure 2.4 shows the default memory-map of PULPino assuming 32 KB of data and instruction RAM. This can be changed in the PULPino top-level SystemVerilog file.

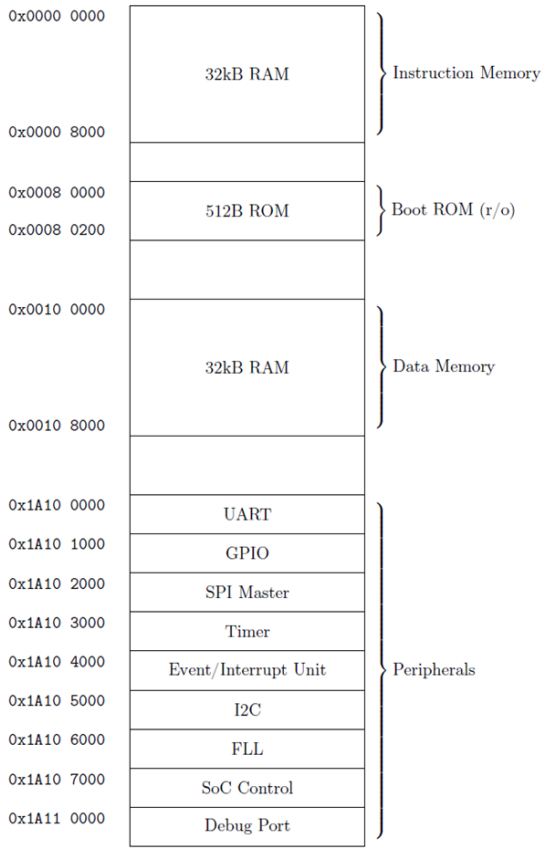


Figure 2.4: Memory map of PULPino [2].

Multiply-Accumulate

RI5CY uses a 32-bit×32-bit multiplier, requiring a single-cycle for the lower 32-bit of the result. All instructions of the RISC-V M instruction set extension are supported. The multiplications with the upper-word result (MSP of 32-bit×32-bit multiplication) take 4 cycles to compute. The division and remainder instructions take between 2 and 32 cycles. The number of cycles depends on the operand values. Additionally, RI5CY supports non-standard extensions for multiply-accumulate and half-word multiplications with an optional post-multiplication shift.

PULP ALU Extensions

RI5CY supports advanced ALU operations that allow performing multiple instructions that are specified in the base instruction set in one single instruction and thus increases the efficiency of the core. For example, those instructions include zero-/sign-extension instructions for 8-bit and 16-bit operands, simple bit manipulation/counting instructions and min/max/avg instructions. The ALU does also support saturating, clipping, and normalising instructions, which make fixed-point arithmetic more efficient.

2.4 Rocket SoC

This is an open-source System-on-Chip implementation based on 64-bit CPU 'Rocket-chip' distributed under BSD license [36, 37]. SOC source files either include a general set of

peripheries, FPGA CADs projects files, own implementation of the Windows/Linux debugger and several examples that help to run your firmware on almost any FPGA board. Satellite Navigation (GPS/GLONASS/Galileo) modules were stubbed in this repository and can be requested on gnss-sensor.com.

Rocket core

The Rocket core [3] is an in-order scalar processor that provides a 5-stage pipeline. It implements the RV64G variant of the RISC-V ISA. The Rocket core has one integer ALU and an optional FPU. The Rocket core contains a fast L1 instruction cache and L1 data cache. An accelerator or co-processor interface, called RoCC, is also provided.

The Rocket core is sometimes described as a 6-stage pipeline with the addition of a 'pcgen' stage. While it is useful to layout the figure in this way, the stage is perhaps best considered as part of the other stages and is not a distinct pipeline stage in the traditional sense. The pcgen and fetch stages are shown below. Instruction fetch is assisted by a gshare predictor, Return Address Stack (RAS) and Branch Target Buffer (BTB). These caches communicate through a simple bus to a simulated DRAM that acts as main memory for the system. Rocket SoC Features are:

- Pre-generated single-core 'Rocket-chip' core (RISC-V). This is 64-bit processor with I/D caches, MMU, branch predictor, 128-bits width data bus, FPU (if enabled) and etc.
- Custom 64-bits single-core CPU 'River'(RISC-V).
- Set of common peripheries: UART, GPIO (LEDs), Interrupt controller, General Purpose timers and etc.

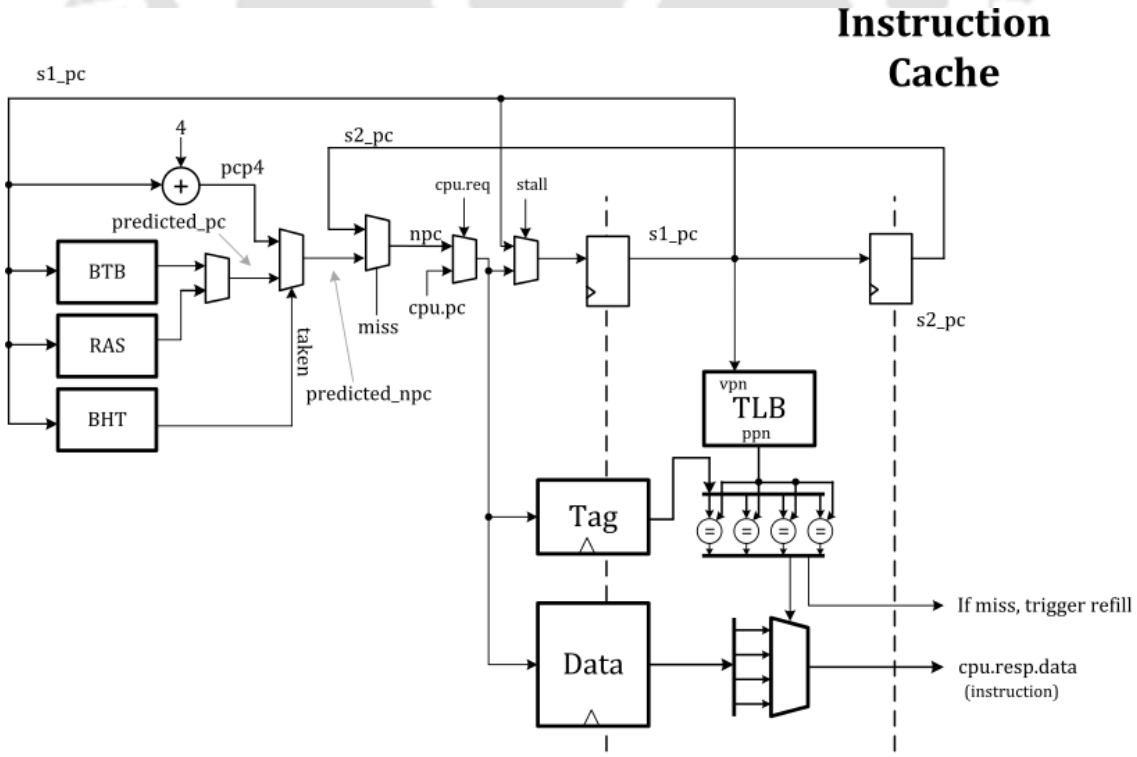


Figure 2.5: Fisrt stage of rocket [3].

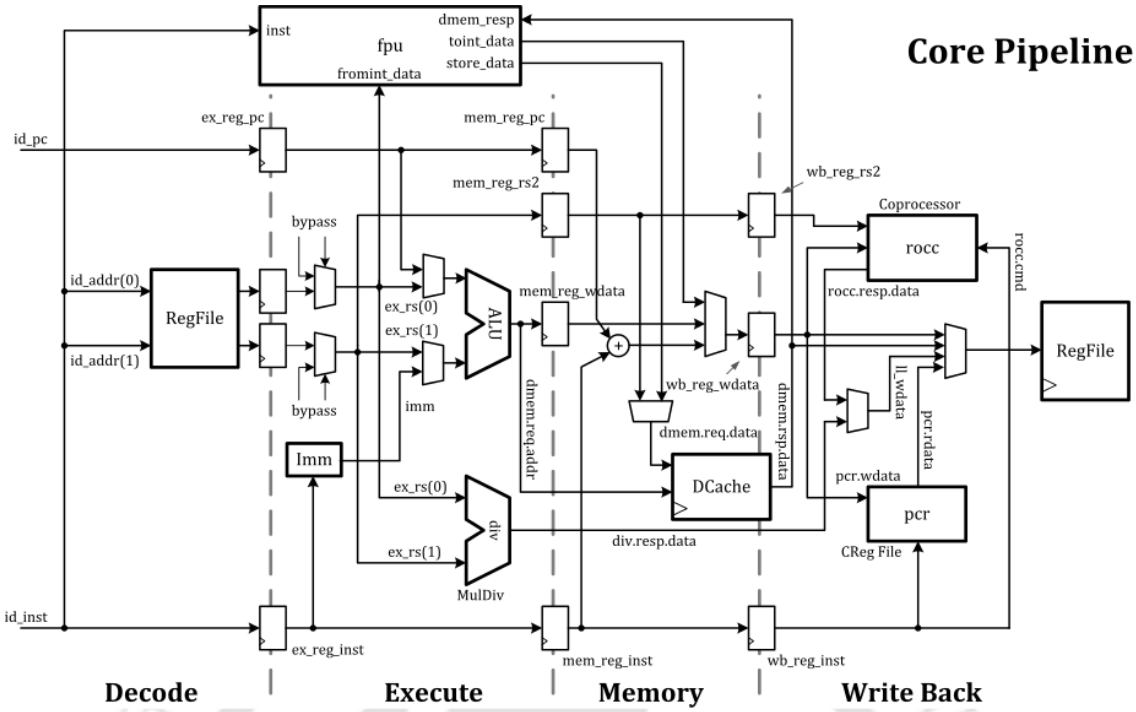


Figure 2.6: Last four stages of rocket [3].

- Debugging via Ethernet using EDCL capability of the MAC. This capability allows to redirect UDP requests directly on system bus and allows to use external debugger from the Reset Vector.
- Debug Support Unit (DSU) for the RIVER CPU with full debugging functionality support: run/halt, breakpoints, stepping, registers/CSRs and memory access. Also, it provides general SoC run-time information: Clock Per Instruction (CPI), Bus Utilization for each master device and etc.
- Templates for the AXI slaves and master devices with DMA access.
- Configuration parameters to enable/disable additional functionality, like GNSS Engine, Viterbi decoder etc.

2.5 Existing Core Results

While exploring the literature, four RISC-V open-source cores have been evaluated by measuring their power, performance and area. These RISC-V cores are,

- Shakti : Developed by IIT Madras.
- PULPino: A small 32-bit RISC-V core developed at ETH Zurich.
- Rocket SoC: RISC-V core SoC developed by founders of RISC-V.
- River SoC: A RISC-V core SoC developed by Sergey.

2.5.1 Performance Measurement

For measuring the performance, benchmarks like Dhrystone and CoreMark are run on these RISC-V cores, and the outputs are compared to rate the cores. Dhrystone and CoreMark benchmarks are available free of charge and are small enough to execute on any processor, including small micro-controllers.

Dhrystone

Dhrystone is a synthetic computing benchmark program developed in 1984 by Reinhold P. Weicker intended to be representative of system (integer) programming. Dhrystone remains in use 33 years after it was designed by Weicker, longer life than most software.

Dhrystone may represent a result more meaningfully than MIPS (million instructions per second) because instruction count comparisons between different instruction sets (e.g. RISC vs. CISC) can confound simple comparisons. For example, the same high-level task may require many more instructions on a RISC machine but might execute faster than a single CISC instruction. Thus, the Dhrystone score counts only the number of program iteration completions per second, allowing individual machines to perform this calculation in a machine-specific way. Another common representation of the Dhrystone benchmark is the DMIPS (Dhrystone MIPS) obtained when the Dhrystone score is divided by 1757 (the number of Dhrystones per second obtained on the VAX 11/780, nominally a 1 MIPS machine).

Another way to represent results is in DMIPS/MHz, where DMIPS result is further divided by CPU frequency to allow for easier comparison of CPUs running at different clock rates. For example, for a CPU

Frequency = 5 MHz
Dhrystone iterations = 10,000
Total time = 9.36 Sec

Then,

Dhrystone per Sec = 100000/9.36 = 10684
DMIPS = 10684/1757 = 6.08
DMIPS per MHz = 6.08/5 = 1.21

CoreMark

CoreMark is a synthetic benchmark that measures the performance of central processing units (CPU) used in embedded systems. It was developed in 2009 by Shay Gal-On at EEMBC and is intended to become an industry standard. The code is written in C and contains implementations of the following algorithms: list processing (find and sort), matrix manipulation (common matrix operations), state machine (determine if an input stream contains valid numbers), and CRC.

CoreMark avoids issues such as the compiler computing the work during compile-time and uses real algorithms rather than being completely synthetic. CoreMark also has established rules for running the benchmark and for reporting the results.

CoreMark result can be measured in the number of iterations per sec. Further, to make it independent of frequency, the result can be divided by frequency. This will give us the number of CoreMark iterations per Sec per MHz.

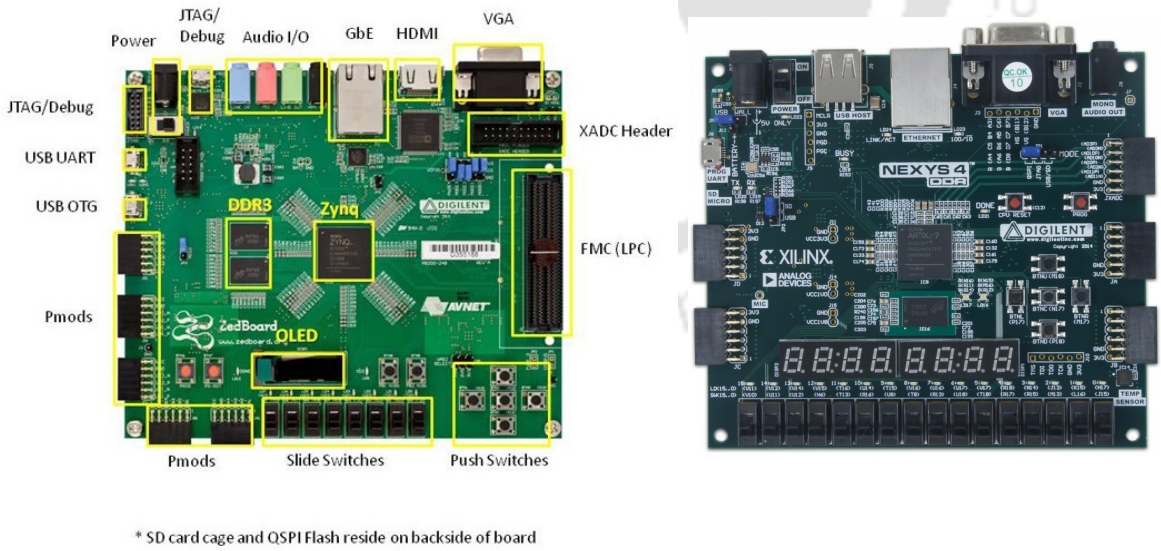
2.5.2 Power and Area Measurement

For power and area measurement, the cores are implemented in Xilinx FPGA boards. Mainly two Xilinx boards, zedboard and nexys4 DDR are used for this purpose. The reason behind choosing these boards is that these cores are compatible with these boards, i.e. the boards have the peripherals required for the cores.

Zedboard

ZedBoard is a low-cost development board for the Xilinx Zynq-7000 all programmable SoC (AP SoC). It is targeted to applications like video processing, motor control, software acceleration, Linux/Android/RTOS development, embedded ARM processing, general Zynq-7000 AP SoC prototyping etc. Its some important features are:

- Xilinx Zynq-7000 AP SoC XC7Z020-CLG484
- 53,200 LUTs, 140 Block RAM, 220 DSP slices
- Dual-core ARM Cortex-A9
- 512 MB DDR3
- 256 MB Quad-SPI Flash
- On-board USB-JTAG Programming
- USB OTG 2.0 and USB-UART
- PS & PL I/O expansion (FMC, Pmod, XADC)



(a) Xilinx zedboard (b) Xilinx nexys4 DDR

Figure 2.7: Xilinx FPGA boards.

Nexys4 DDR

The nexys4 DDR board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7 Field Programmable Gate Array (FPGA) from Xilinx. With its large, high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C), generous external memories, and collection of USB, Ethernet, and other ports, the nexys4 DDR can host designs ranging from introductory combinational circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, a speaker amplifier, and several I/O devices, allow the nexys4 DDR to be used for a wide range of designs without needing any other components. Its some important features are:

- Xilinx Artix-7 FPGA XC7A100T-1CSG324C
- 15,850 logic slices, each with four 6-input LUTs and 8 flip-flops
- 4,860 Kbits of fast block RAM
- Six clock management tiles, each with phase-locked loop (PLL)
- 240 DSP slices
- USB-UART Bridge
- 16 user switches, 16 user LEDs
- Two 4-digit 7-segment displays
- Pmod for XADC signals, Four Pmod ports

2.5.3 Results

Table 2.1 shows the evaluation of existing RISC-V cores. The filled up boxes are those, who are successfully completed and verified. Some of the boxes are empty; they could not be completed due to the lack of enough material and resources provided by the founder.

The initial plan was to measure the performance of the cores in simulation as well as in emulation. The simulation and emulation are completed successfully for Shakti C-class 32-bit. But, for PULPino, only emulation is completed using zedboard; as for simulation, we need QuestaSim. For Rocket SoC, we were able to run Dhrystone on a GUI using Qt and SystemC libraries. For Shakti E-class, according to IIT Madras, the core is ready, but the environment required for the testing of the core is not ready. Also, the repository has some bugs which have not been solved till now. So, we could not run any benchmark on it.

From Table 2.1, if we compare the Dhrystone and CoreMark benchmarks result, we could see that Rocket is the fastest CPU among all. Rocket can run 1.72 Dhrystone per sec per MHz, whereas Shakti C-32 can run 0.52, PULPino can run 1.21, and River can run 0.33. PULPino CoreMark result is better than the others. PULPino can run 2.8 CoreMark per Sec per MHz, whereas Shakti C-32 can run 1.85 CoreMark per Sec per MHz. All the cores are synthesised in Xilinx Vivado 2015.04, and their area and power measurement are done. Area wise, Shakti E-class is the smallest one. It requires only 10,651 LUTs on Xilinx nexys4 DDR. PULPino requires 15,406 LUTs on Xilinx zedboard, and Shakti C-32 requires 18,613 LUTs on Xilinx nexys4 DDR. Rocket SoC requires 30,773 LUTs on Xilinx zedboard, and River

Table 2.1: Evaluation of RISC-V cores.

			Shakti		PULPino	Rocket	River
			C-32bit	E-32bit			
Toolchain			Iverilog, RISC-V		Vivado, RISC-V	Vivado	Vivado
Simulation	Functional Verification		AAPG, CSMITH, Torture test		NA	NA	NA
	Benchmark	DMIPS (Per MHz)	0.58	NA	NA	NA	43 VAX MIPS
		CoreMark (Per MHz)	NA	NA	NA	NA	NA
Emulation	Synthesys	Xilinx	18,613 LUTs 7,722 FFs	10,651 LUTs 1,652 FFs	15,406 LUTs 9,756 FFs	30,773 LUTs 14,424 FFs	28,720 LUTs 10,563 FFs
		Altera	Non Synthesizable (Memory error)	6,358 ALMs 11,071 ALUTs	Uses xilinx IP	(Targeted to Xilinx Virtex6/Kintex7)	Targeted to Xilinx
	Benchmark	DMIPS (Per MHz)	0.52	NA	1.21	1.72 (from presentation [38])	0.33 (from repository)
		CoreMark (Per MHz)	1.85	NA	2.8		NA
	FPGA Power		12.2 mW/MHz	NA	18.37 mW/MHz (SoC)	NA	8.13 mW/MHz (SoC)
Targeted FPGA	Xilinx nexys4 DDR		Verified				
	Xilinx zedboard		Verified				

SoC requires 28,720 LUTs on Virtex7, which is a little higher compared to others. But, it is acceptable because it is the complete SoC containing the CPU as well as other peripherals. Even if the area is more, the power requirement for the River SoC is the least. It requires 8.13 mW/MHz. Whereas for Shakti C-32, it is 12.2 mW/MHz, and for PULPino, it is 18.37 mW/MHz.

2.6 Methodology

The work can be divided into two parts: the first part is to design a software environment where the hardware cores can be developed and verified. This includes functional verification using HDL simulators, hardware verification in FPGA, and estimation of delay, power, performance and area. The second part is to develop cores based on the requirements of modern world devices. We have developed three different cores based on RISC-V ISA. The first core is targeted to low-power applications. It supports the basic integer instructions (RV32IM) of RISC-V ISA. This core shows better results than many existing commercial/non-commercial cores. The core can also be used as a base for developing high-performance cores. The second core is designed to perform better in data processing related to IoT devices. We integrated FPU (Functional Processing Unit) and DSP (Digital Signal Processing) unit to improve the core performance in IoT applications. The third core is designed for high-security applications. The Advanced Encryption Standard (AES) is a standard technique used for electronic data encryption. In the third core, a dedicated hardware unit is incorporated into the core to boost the AES encryption and decryption. Figure 2.8 shows the design and verification flow of the cores.

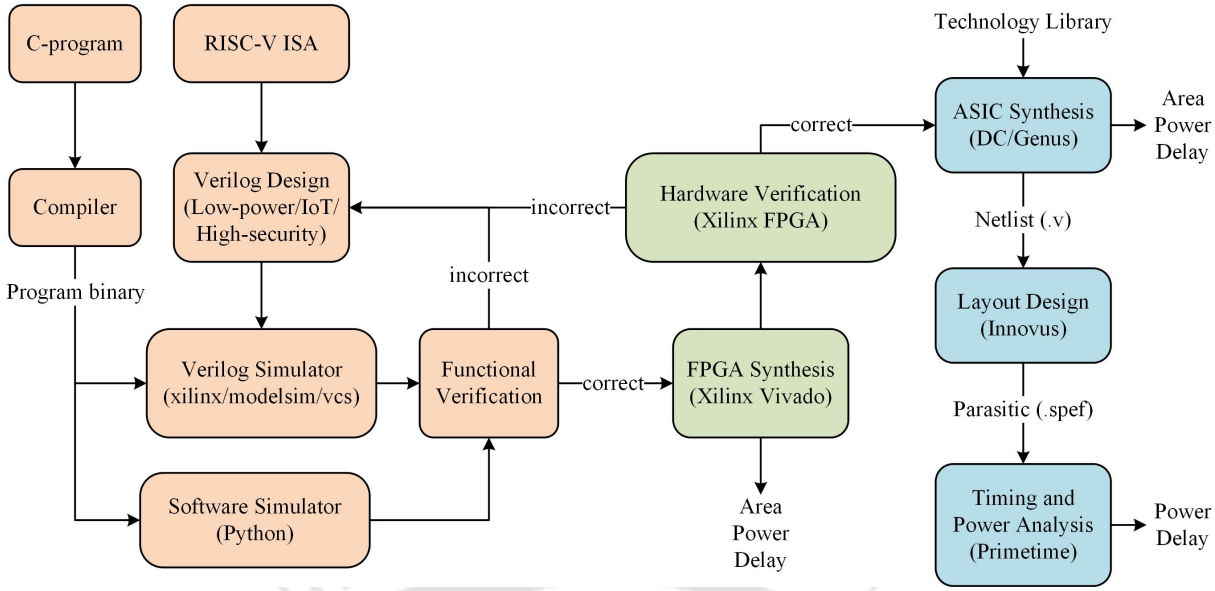


Figure 2.8: Design, verification, and evaluation process of the proposed cores.

2.6.1 Software Verification

The proposed cores are designed using Verilog HDL, and simulation is done in Xilinx Vivado 2016.4. Once the core (Verilog code) is ready for verification, the functionality of the core is tested first. For that, some random high-level language code (C or Assembly) is compiled using RISC-V gnu compiler and then run on a python environment. The python environment executes the instructions and stores the register and memory read/write status after every instruction in a file. The compiled code is also run on Verilog simulators, and register memory read/write status is stored in a separate file. If the result of python environment and Verilog simulator matches, then the functionality of the core is correct. Otherwise, we need to go back to the design step and debug the HDL code. Figure 2.9 shows the Functional verification process flow. During the functional verification process, the core is tested with various C programs, addition, multiplication, loops, etc. to find any possible bug in the design. If the core runs all the input programs correctly, then it is processed to verify on FPGA hardware platform.

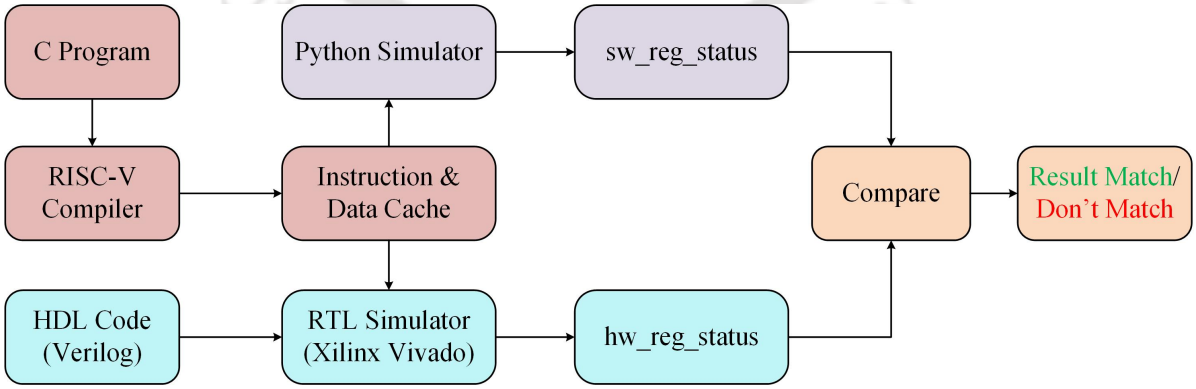


Figure 2.9: Functional verification process of the propose cores.

2.6.2 Hardware Verification

For hardware verification, we used FPGA boards and the process flow is shown in Figure 2.10. As in functional verification process, the first step is compilation of the C or Assembly code. Then the compiled output and HDL code is synthesised using Xilinx Vivado 2016.4 for the targeted FPGA board. The next step is to Implement the design and generate the bit file. Once the bit has been generated, it is uploaded to the targeted board and verified.

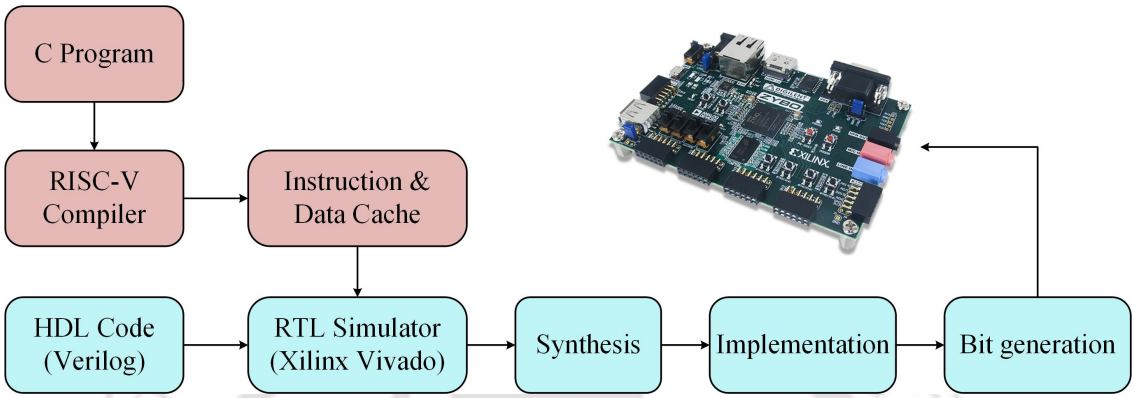


Figure 2.10: FPGA hardware verification process of the proposed cores.

The FPGA used for the hardware verification is Xilinx Virtex-7 vc707 board shown in Figure 2.11. The vc707 is a full-featured, highly flexible, high-speed serial base platform using the Virtex-7 XC7VX485T-2FFG1761C and includes basic components of hardware, design tools, IP, and pre-verified reference designs for system designs that demand high-performance, serial connectivity and advanced memory interfacing. The Virtex-7 board has the following features:

- 485760 Logic cells, 2800 DSP slices, 37080 Kb block memory
- 200 MHz Fixed Oscillator, 56 GTX 12.5 GB/s Transceivers

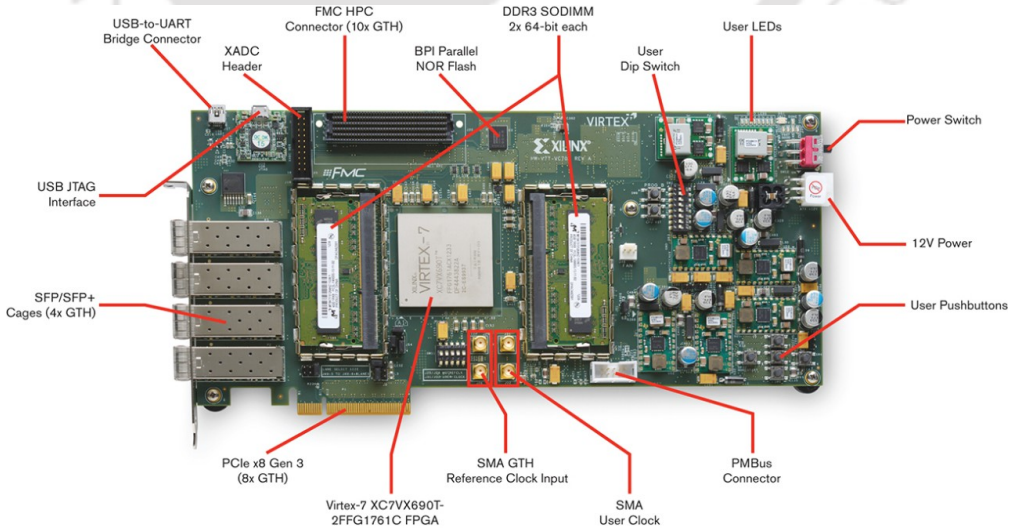


Figure 2.11: Xilinx Virtex-7 vc707 FPGA.

2.6.3 Evaluation

The final step of the work is evaluation of the core in terms of power, area, delay, and performance. Figure 2.12 shows different possible ways of estimating these parameters. These parameters can be estimated while doing the FPGA hardware verification process. But, those results are valid only for FPGAs and may vary while implementing on silicon. So, industrial tools are also used to estimate the power, area, delay, and performance of the designs. The tool used for the ASIC synthesis of the designs is Synopsys Design Compiler (DC), and the technology nodes used are UMC 40 nm, UMC 65 nm, and UMC 90 nm. For better estimation of the results, the layouts of the designs are also implemented. The place and route of the synthesised design is done in Cadence Innovus, and Synopsys PrimeTime tool is used for the post-layout estimation of power and delay.

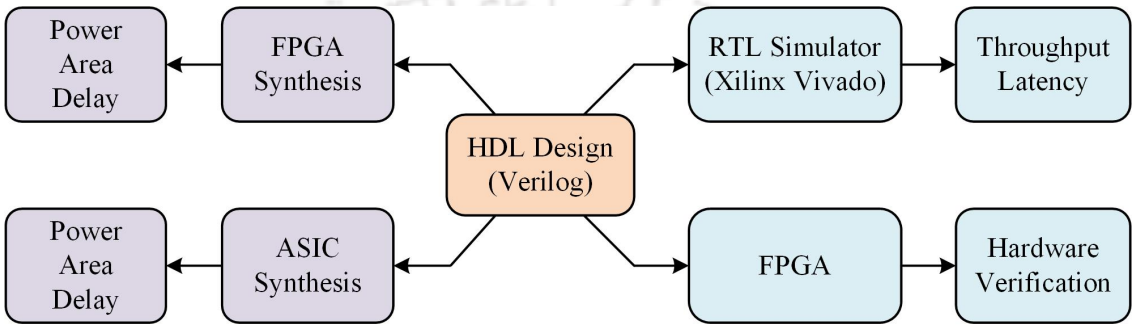


Figure 2.12: Evaluation of the proposed cores.

2.7 Conclusion

The power, performance, and area for the cores, Shakti, PULPino, Rocket, and River, have been calculated using simulation and emulation. In the simulation, the functional verification of the cores have been done, and then Dhrystone and CoreMark benchmarks are run on it to measure the performance. The software tools used for that are RISC-V gnu toolchain (spike), iVerilog, Altera Modelsim. For power and area estimation, the cores are synthesised using Xilinx Vivado 2015.04. Further, these cores are programmed into the FPGA boards (zedboard, nexys4 DDR), and benchmarks are tested on them.

Verification and evaluation are two important parts of processor design. In this work, we have developed a software environment using python for functional verification of the proposed designs. The python environment reads the program binary and executes the instructions to generate the expected register values. These expected results are then used to verify the Verilog HDL during simulation. This chapter also explains the hardware verification process of the proposed designs. The evaluation process includes estimation of the area, power, delay and performance of the designs. These parameters are estimated using various industrial tools, Synopsys DC, Synopsys PrimeTime, Cadence Innovus, Xilinx Vivado, etc.

CHAPTER

3

DESIGN AND IMPLEMENTATION OF RISC-V CORE FOR LOW-POWER APPLICATIONS

Contents

3.1 Introduction 26
3.2 Micro-architecture 27
3.2.1 Instruction Fetch 28
3.2.2 Instruction Decode 28
3.2.3 Instruction Execute 28
3.2.4 Register Write 33
3.3 Results and discussion 33
3.3.1 ASIC 34
3.3.2 FPGA 35
3.4 Conclusion 37

3.1 Introduction

In the modern technological era, the demand for appliances such as portable phones, medical devices, wired and wireless communication devices, several Internet of Things (IoT) devices, etc., is increasing. All these systems require a high-performance processor to carry out their tasks. Also, most of these applications are portable and battery-powered. Therefore, these systems demand a high-performance, low-power processor that can execute the instruction without draining much power from the battery. In the last four decades, processor performance has increased through CMOS scaling. But, now it is coming to its end as CMOS technology is approaching physical limitations. So, designers are forced to look into other aspects of processor design to get better power and performance within the aforementioned constraints. The two most important aspects of processor design are Instruction Set Architecture (ISA) and micro-architecture. There are mainly two ISAs that are dominating the current processor industry: ARM, which is used in portable devices like mobile and x86, which is used in personal computers and servers. These ISAs may not be the best developed so far, but with time, they have become so popular that no other ISA could replace them. However, these ISAs have proprietary issues because of which the hardware designers cannot use them for their designs. A new ISA called RISC-V [6] had been developed by UC, Berkeley to overcome these issues and to support computer architecture research, education, and industry implementation. The main advantage of this ISA is that it is in the public domain, and anyone can use it for their research.

Based on RISC-V ISA, many cores have been developed, and many are under development. Rocket [7, 8] is the first approach towards RISC-V micro-architecture by the same researchers who have developed the RISC-V instruction set. It is a 6-stage single-issue in-order pipeline processor based on RV64G ISA. The authors achieved a 10% higher performance in DMIPS/MHz score compared to Cortex-A5 and 49% area improvement. Pulpino [9–11] is an open-source micro-controller system, based on RV32IMC RISC-V core developed at ETH Zurich. It is a platform organized in clusters of RISC-V cores sharing a tightly coupled data memory. Shakti-F [12] is another approach where researchers at IIT Madras are trying to develop a reliable processor for radiation prone environments like nuclear and space applications. Some variants of Shakti like Shakti-E, Shakti-C, Shakti-I are also under development [13]. Few more examples of RISC-V core are ORCA [14], mriscv [15], VexRiscv [16], LowRISC [17], RI5CY [18] etc. In this chapter, a processor micro-architecture is proposed, which is capable of achieving high performance at the cost of very low-power requirements. The salient features of the design are: 1) the pipeline is organized in four stages, and all integer instructions except load and store are executed in four stages, 2) Memory instructions, load, and store are separated from the mainstream of pipeline but with an extra stage of memory read/write, 3) most of the arithmetic operations are executed in a single clock cycle. But, the multiplication and division operations are executed in multiple cycles to reduce the critical path delay, 4) Modified Baugh Wooley multiplier so that it can operate in dual clock cycles, 5) 32-bit radix-16 divider is designed to complete in 8 clock cycles, 6) dynamic branch prediction (DBP) unit with a 2-bit counter is incorporated. The results showed that the core could achieve better performance compared to many existing core micro-architectures with the lesser cost of area and power.

The chapter is organized into three sections. The following section will discuss the micro-architecture and reasons behind adapting different functional blocks. In Section 3.3, the experimental results are discussed. Finally, the work is concluded in Section 3.4.

3.2 Micro-architecture

In this section, we will detail micro-architectural optimizations for increasing the efficiency of the processor core. The starting point of this work is RISC-V ISA. The core supports the base integer instruction set of RISC-V ISA with integer multiplication and division extension. This micro-architecture can be used as the base of an advanced core for supporting extensions and features. The main focus of this work is on optimizing the core power and performance. The efficient design of primary components of any system leads to an overall improvement of that system. So, in the proposed micro-architecture, basic components of the core like multiplier, divider, DBP, etc., are optimized to get the best possible result. The critical path of the core is also optimized to achieve a maximum frequency close to 200 MHz. Pipeline stages are organized in such a way that minimum data, control, and structural hazards can occur. These points are discussed in more detail in the following part of this section.

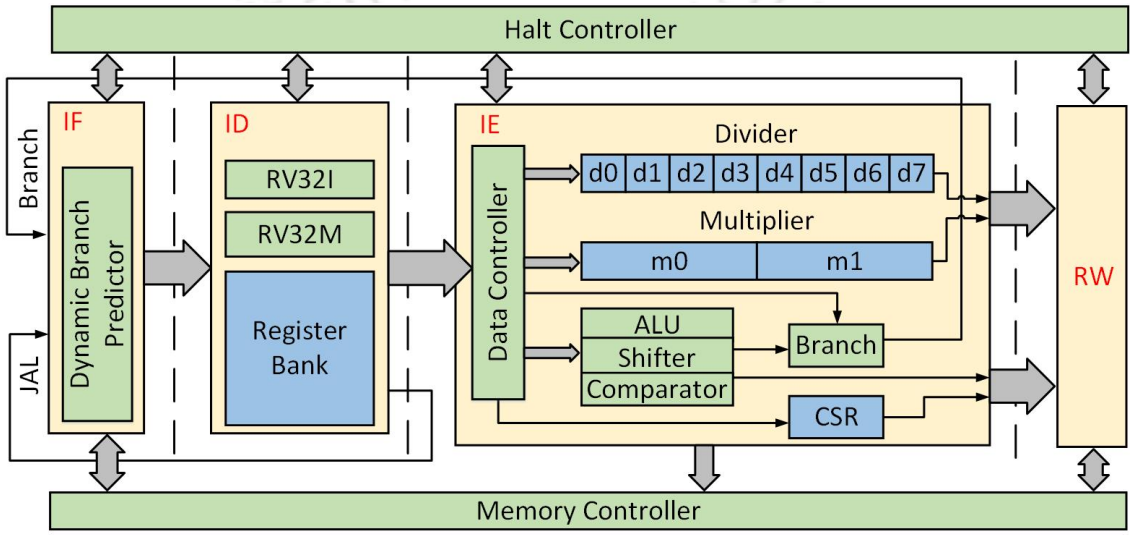


Figure 3.1: Pipeline stages of the proposed micro-architecture.

The number of pipeline stages is one of the critical aspects of any processor design. A higher number of pipeline stages allows a designer to achieve higher operating frequency, which leads to an increase in overall throughput. However, it also increases data and control hazards and reduces the IPC (Instructions per Cycle). For achieving high-performance, processors are also optimized using branch predictions, pre-fetch buffers, and speculation. However, these optimizations increase power consumption and are usually not suitable for low-power applications. For example, ARM Cortex-M4 is a three-stage pipelined micro-architecture with a single write-back port in the register bank. The absence of the second write-back port and fourth pipeline stage brings stalls while executing load operations. A three-stage pipeline is good for achieving high IPC, but the frequency of such a pipeline is limited because of multiple operations in a single stage. The proposed micro-architecture is designed in four pipeline stages in order to achieve a balance between IPC and frequency. The pipeline stages of the proposed core are Instruction Fetch (IF), Instruction Decode (ID), Instruction Execute (IE) and Register Write (RW), as shown in Figure 3.1. Most of the instructions can be completed within these four stages. But, there are few instructions for which this pipeline is modified to get optimum performance. The unconditional jump instruction JAL is completed after the first stage as it does not involve any register read/write operation. Load instructions require another stage before RW stage because the execution

of these instructions involve memory access delays. Load instructions require another stage (Memory Read) before RW stage because the execution of these instructions involve memory access delays.

The proposed core supports base integer instruction set RV32I with multiplier extension M. Multiplication and division of 32-bit numbers is an expensive process in terms of critical path delay. So, these operations are executed in multiple clock cycles to reduce the critical path delay.

3.2.1 Instruction Fetch

The IF stage generates a memory read request for instructions. It requests the next instruction as soon as it receives the previously requested instruction. The instruction fetching process takes one clock cycle, as every memory read process requires a minimum of one clock cycle. Therefore, the IF stage has two 32-bit registers, one to hold the address of the current instruction and the other to hold the address of the next instruction. This stage does not introduce any new delay to the circuit except the delay due to the memory read operation.

The micro-architecture includes a DBP that fetches instructions depending on the past result of branch instructions. The DBP maintains a history table with the help of program counter, branch address, and 2-bit prediction counter. The value of the counter decides the prediction. The four states of the 2-bit counter are assigned as strongly not taken, weakly not taken, weakly taken and strongly taken. For the first two states, the branch is predicted as false, and for the subsequent two states, the branch is predicted as true. If a wrong prediction occurs, the whole pipeline needs to be flushed, and correct instruction has to be fetched again. The number of bits for prediction counter can be one, two or more than two. But, every option has its own advantages and disadvantages [39]. A 1-bit counter is too simple, and prediction accuracy is not very good. A 2-bit predictor works well when branches predominantly go in one direction. A second check is made to make sure that a short and temporary change of direction does not change the prediction away from the dominant direction. Increasing the counter size further does not affect much the accuracy of prediction [40], but increases the complexity of the design. So, a 2-bit saturating counter based DBP is used for the proposed design.

3.2.2 Instruction Decode

In this stage, the fetched instructions are decoded to generate different control signals and added for the IE stage. All the instructions have to go through this stage of operation. All control transfer (jump & branch) instructions complete their operation in IE stage itself except the unconditional jump instruction, JAL. JAL instruction does not involve any register read/write operation; hence it is separated from other control transfer instructions and is forced to complete its operation in ID stage. As a result, the flushing of IE stage is not required for JAL. This leads to overall performance improvement of the core.

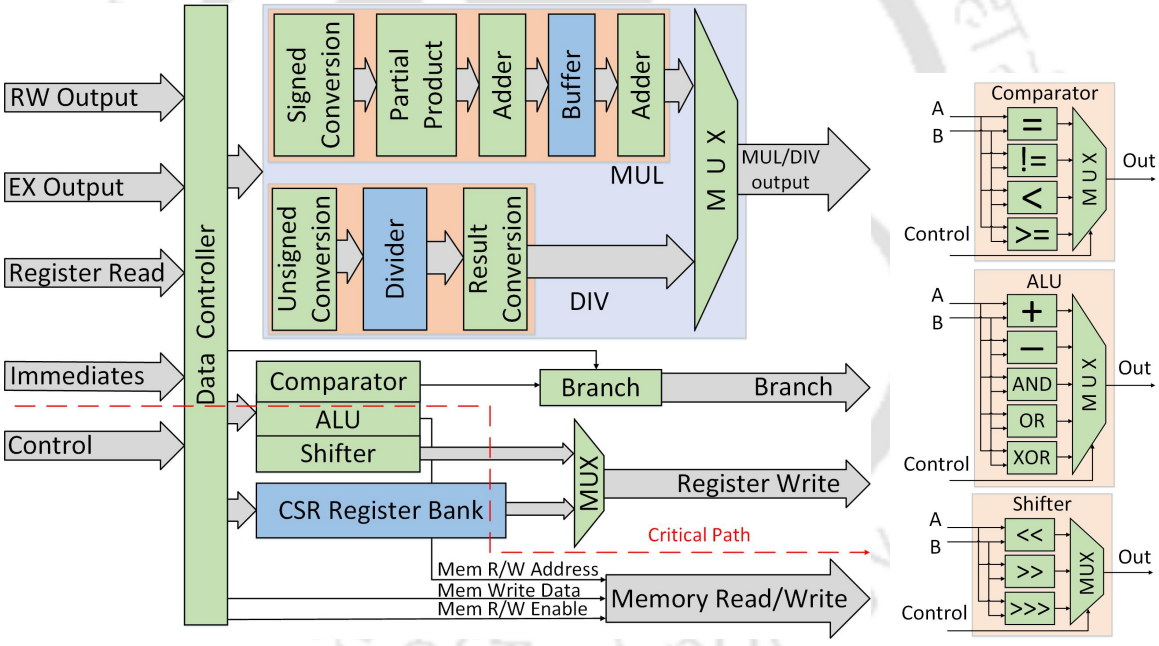
3.2.3 Instruction Execute

Figure 3.2 shows the block level representation of the IE stage. In IE stage, register values are first checked. If they are ready to read, execution continues; otherwise, values are fetched from IE/RW stage, depending on previous instructions. These register values are then fed

into different Functional Units (FU) of IE stage for execution. The IE stage has five FUs. These FUs with their operations and clock cycles are mentioned below:

1. ALU - add, sub, and, or, xor (1 clock)
2. Comparator - signed/unsigned comparison (1 clock)
3. Shifter - logical/arithmetic shift (1 clock)
4. Multiplier - signed/unsigned multiplication (2 clocks)
5. Divider - signed/unsigned division (8 clocks)

IE stage also generates the memory read/write signals and branch signals. Memory read/write address is obtained from the ALU unit, and memory read/write data is obtained from the second source register, i.e. rs2. Memory read/write enable signal comes from the instruction ID stage. These three signals are then combined to control the memory read/write operations. Branch consists of two signals; branch enable and branch address. Branch address is assigned from ALU output, and comparator output triggers the branch enable signal depending on the type of instruction.



(a) Instruction Execute (IE) (b) Functional units

Figure 3.2: Micro-architecture of Instruction Execute (IE) stage.

Multiplier and Divider circuits are vital parts of any core micro-architecture, and their hardware complexity is very high compared to other functional units of IE stage. So, implementing a correct hardware circuit for multiplication and division has a vital role in the overall area, power, and performance of a core.

Multiplier

There is a lot of binary multiplication methods [41–43] available in literature namely Baugh-Wooley [44], Booth [45], Vedic [46], Dadda [47] etc. The hardware circuits for some popular

Table 3.1: Synthesis results of multiplier circuits at UMC 40 nm technology node.

Algorithm	Signed/ Unsigned	Area (μm^2)	Power (μW)	Delay (ns)
Baugh Wooley	Unsigned	5375.9	101.2	5.94
Baugh Wooley	**Signed**	**5579.5**	**102.2**	**5.97**
Booth Radix-2	Signed	8197.6	144.7	6.03
Booth Radix-4	Signed	5937.0	113.6	6.16
Booth Radix-8	Signed	5912.5	107.7	7.08
Vedic	Unsigned	7485.1	113.4	5.25
Vedic Dadda	Unsigned	6652.7	121.3	5.1
Vedic Dadda	Signed	8001.8	141.8	6.36

multiplication methods are implemented using Verilog HDL, and the most suitable circuit is selected based on our requirements. These designs are then synthesized using Synopsys DC at UMC 40 nm technology node, and results are shown in Table 3.1. Among all the implementations, Baugh Wooley signed multiplier is found to be the most suitable one for our micro-architecture.

$$A * B = a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2} a_i 2^i \sum_{j=0}^{n-2} b_j 2^j \sum_{i=0}^{n-2} \overline{a_i b_{n-1}} 2^i + \sum_{j=0}^{n-2} \overline{a_{n-1} b_j} 2^j + 2^n - 2^{2n-1} \quad (3.1)$$

From Table 3.1, it can be seen that the delay of Baugh Wooley signed multiplier circuit is 5.97 ns. If this circuit is used directly in the IE stage, the critical path of the overall system increases. Therefore, the hardware circuit of multiplier is modified into two stages. In the first stage, partial products are calculated using (3.1). Then, they are divided into six parts as shown in Figure 3.3 and are added. The adder results are then buffered to the next stage. In the second stage, the six buffered results are added again to get the final result, as shown

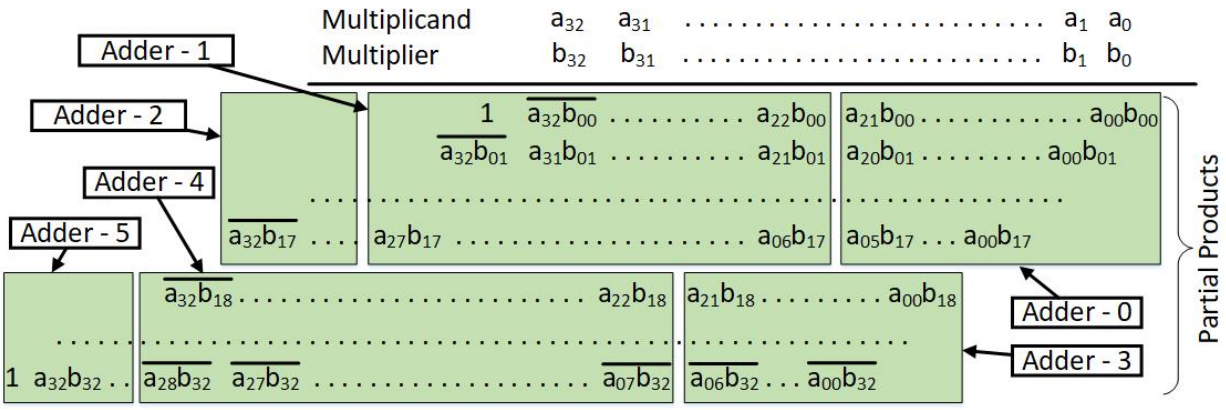


Figure 3.3: Addition of partial products of 33×33 signed multiplier.

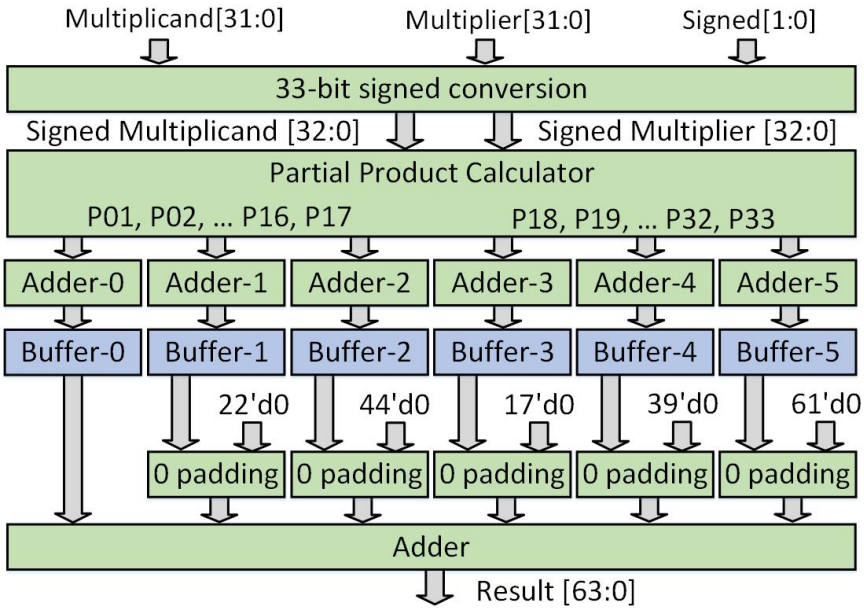


Figure 3.4: Block diagram of 32-bit Baugh Wooley signed/unsigned Multiplier.

in Figure 3.4. The critical path of the first stage consists of 1 HA (Half Adder) and 29 FAs (Full Adders). In comparison, the second stage consists of 1 HA and 47 FAs. The whole partial product segmentation is done in such a way that the delay of the first stage remains lesser than the second stage. This is because the input data of multiplier may come from register bank or from other stages of the pipeline. So, a multiplexer delay is expected at the inputs of multiplier circuit.

Similar to the implementation of multiplication, the hardware circuits for available division algorithms [48] are implemented using Verilog HDL and synthesized at UMC 40 nm technology node. Table 3.2 shows the area, power, delay, latency and throughput of these circuits. The throughput is measured in terms of the number of inputs the circuit can execute per clock cycle. From Table 3.2, it can be seen that as we move to higher radix algorithms, latency of the circuit decreases, and as a result, throughput increases. If the results of radix-16 and radix-32 are compared, it can be seen that there is not much improvement in latency and throughput. But, there is a significant increment in power and area. So, radix-16 is se-

Table 3.2: Synthesis results of divider circuits at UMC 40 nm technology node.

Algorithm	**Area** (μm^2)	**Power** (μW)	**Delay** (ns)	**Latency**	**Throughput** (per Hz)
Four Stages	6416.3	71.2	2.97	32	0.125
Combinational	9516.6	69.8	60.32	1	1.000
Radix-2	862.0	11.4	3	32	0.031
Radix-4	1521.8	13.8	4.35	16	0.062
Radix-8	3048.8	20.4	4.8	11	0.090
Radix-16	**6092.3**	**35.3**	**5.12**	**8**	**0.125**
Radix-32	9475.6	45.6	5.29	7	0.142

lected as the divider unit for the proposed micro-architecture. The pseudo-code for a 32-bit radix-16 unsigned divider is described in Algorithm 1.

Algorithm 1 32-bit radix-16 divider

```
Radix = 16; Rn = log2(Radix); Quotient = 0; D = Divisor;
if (32%Rn) != 0 then
  Zn = 32 + Rn - (32%Rn);
else
  Zn = 32;
end if
N = { Zn{0}, Dividend};
Nmax = size of N; Nmin = Nmax-32-Rn;
for i = 1 to ceil(32/Rn) do
  for n = (Radix-1) downto 0 do
    if N[(Nmax-1):Nmin] >= n*D then
      Quotient = { Quotient, n };
      N[(Nmax-1):Nmin] = N[(Nmax-1):Nmin] - n*D;
      Break;
    end if
  end for
  Remainder = N[(Nmax-1):Nmin];
  N << Rn ;
end for
```

The block diagram of the hardware architecture for the divider is shown in Figure 3.5. In the hardware circuit, divisor is first multiplied with constants, and the results are stored in registers. To compute these multiplications with constants, a lot of multipliers are required, which are highly expensive in terms of area, delay and power. So, while implementing the hardware circuits, these multiplications are implemented using shifters and adders. Multiple of 2, 4 and 8 are calculated by left shifting the divisor, and the other multiples are determined by adding the results among them. The critical path of the circuit is found to be 3.10 ns at

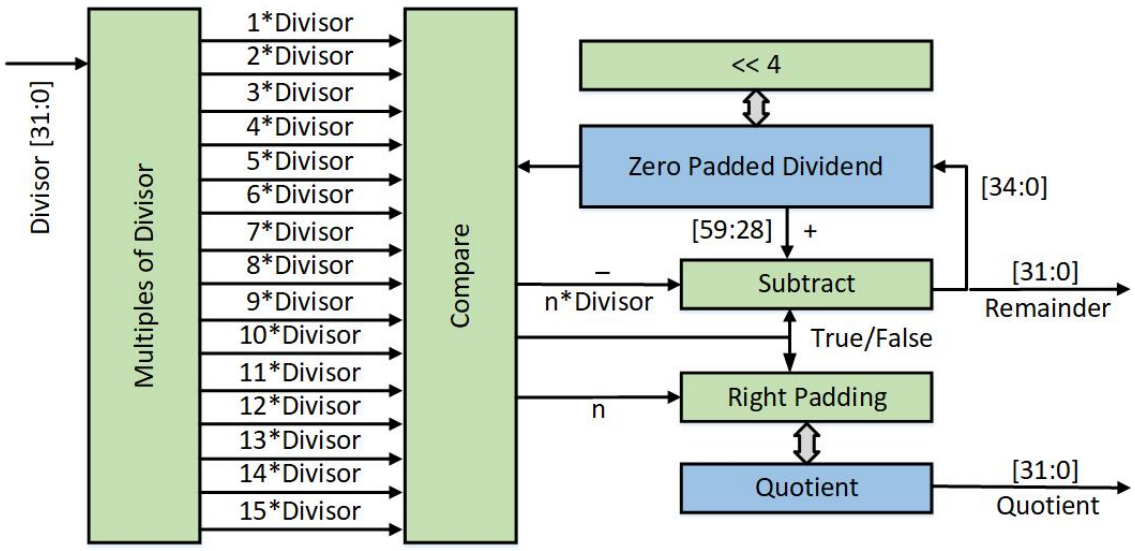


Figure 3.5: 32-bit radix-16 unsigned divider.

40 nm technology node.

In the division process, the next step is to update the quotient and remainder. At every clock cycle, the most significant 36-bits of the dividend is compared with the pre-calculated multiples of dividend. At first, the most significant bits of dividend is compared with 15*divisor. If the result says that the most significant bits of dividend is greater or equal to 15*divisor, then the quotient is left-shifted by 4-bits with a value '1111', 15*divisor is subtracted from the higher 36-bits of dividend and remainder is updated with the subtraction result. If the comparison result indicates that it is lesser than 15*divisor, then the comparison continues for lower multiples of divisor. This process continues for eight clock cycles, and at the end of 8^{th} cycle, we get the desired quotient and remainder values.

The divider unit discussed above takes eight clock cycles to complete one division and is an unsigned divider. But, the ISA requires both signed and unsigned division. To enable signed operations, the magnitudes of the inputs are first determined. These 32-bit magnitudes are then fed to the divider circuit, and the magnitude of the result is obtained. This result is finally converted to 2's complement depending on the type of inputs (signed/unsigned) and the type of instruction. Whenever the result produces a signed negative number, the result of an unsigned divider circuit should be converted to a 32-bit signed negative number using 2's complement representation. Otherwise, it should not be changed. Because of these signed-unsigned conversions, two additional clock cycles are required, one prior to division, to determine the magnitude of the inputs and one after division to convert back the result to signed number.

3.2.4 Register Write

This stage receives three different register write instructions. One from ALU-shifter-comparator, one from multiply-divide units and one because of memory read operations. These three register write operations are executed through two write ports of the register bank. If one or two register write requests come in the same clock cycle, the requests are assigned to available write ports of register bank. Whenever three write requests come in the same clock cycle, the RW stage generates a halt signal for the pipeline and executes any two of the write requests. The third request is executed in the next clock cycle, and the pipeline continues from the stalled state.

3.3 Results and discussion

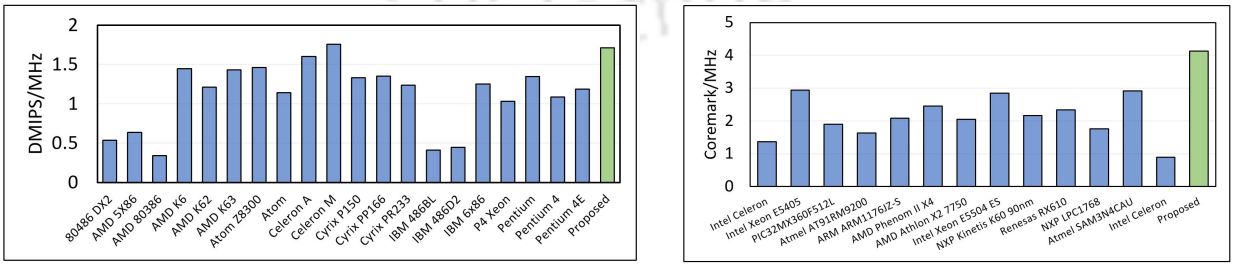
The proposed core supports base integer instruction set, RV32I of RISC-V ISA with multiplier and divider extension, M. The entire design is implemented using Verilog HDL. For the estimation of hardware, power, and energy efficiency, the proposed micro-architecture is synthesized using Synopsys DC. The place and route of the synthesized design is done in Cadence Innovus at UMC 90 nm and UMC 40 nm technology nodes. For the post layout estimation of power and delay, Synopsys PrimeTime tool is used, and Synopsys VCS is used for simulation. The area, power, delay, and performance results are compared with some commercial as well as some RISC-V cores in Table 3.3. It can be seen that the core results are better than some of the existing micro-architectures, and it is suitable for low-power applications.

Table 3.3: ASIC post-layout results of the proposed low-power core at UMC 90 nm and 40 nm technology nodes.

Core	Proposed		[11]		[49]	[23]	[50]	[51]	[52]
ISA	RV32IM		RISC-V basic	RISC-V Ext	OpenRISC	ARMT32	ARMT32	icyflex	RV32IMAFC
Technology Node	40 nm	90 nm	65 nm	65 nm	65 nm	90 nm	90 nmLP	180 nm	55 nm
Area (mm^2)	0.036	0.173	0.068	0.077	0.064	0.119	0.09	1.6	0.083
Max Freq (MHz)	198.02	166.94	357	357	362	216	NA	50	210
Dynamic Power (µW/MHz)	**17.36**	**19.75**	26.28	26.68	33.8	32.8	31	120	67.8
DMIPS/MHz	**1.71**		NA	NA	NA	1.52	1.50	NA	1.38
CoreMark	817.8	689.4	867.5	1013	962.9	734.4	NA	NA	651.0
CoreMark/MHz	**4.13**		2.43	2.84	2.66	3.40	3.34	NA	3.10
CoreMark/mW	**237.8**	**209.0**	92.46	106.3	78.6	103.6	NA	NA	45.7
CoreMark/μm^2	0.022	0.003	0.012	0.013	0.015	0.006	NA	NA	0.007

3.3.1 ASIC

At UMC 40 nm technology node, the core occupies an area of 0.036 mm^2, which consists of 20 K cells and 2.7 K buffers, which is almost half of the area occupied by [11]. The critical path delay of the proposed micro-architecture is 5.05 ns and 5.39 ns at UMC 40 nm and UMC 90 nm, respectively. That means the core could achieve a maximum frequency of 198.02 MHz at 40 nm and 166.94 MHz at 90 nm. While running at 100 MHz clock frequency and 40 nm technology node, the core consumes 1.745 mW total power. 99.5%, i.e. 1.736 mW, of this power is dynamic, and the leakage is only 0.5% (7.63 µW). The core consumes 2.225 mW total power at 90 nm technology node while operating at a clock frequency of 100 MHz. The dynamic power consumption is 1.975 mW which is 88.8% of the total power. To make a fair comparison among the cores, dynamic power consumption is normalized to the clock frequency. The proposed micro-architecture consumes 19.75 µW/MHz dynamic power, which is 36% better than Cortex-M3 and 40% better than Cortex-M4. The dynamic power consumption is also comparable to some other cores evaluated at lower technology nodes shown in Table 3.3.



(a) Dhrystone score

(b) CoreMark score

Figure 3.6: Comparison of Dhrystone and CoreMark scores with other commercial cores [4,5].

For performance measurement, two benchmark programs, Dhrystone and CoreMark are

run on the FPGA module of the proposed micro-architecture. The core performs 1.71 DMIPS per MHz and 4.13 CoreMark per MHz. These Dhrystone and CoreMark scores are better than almost all other cores. The energy efficiency of the core is measured in terms of CoreMark/mW. The maximum operating frequencies of [11] and [49] are higher than the proposed design. This is due to the different variants of a single technology node. For example, UMC 40 nm technology node has variants like ULVt(Ultra Low Vt), HVT(High Vt), RVT(Regular Vt), etc. Every variant has its own advantages and disadvantages. In this work, the RVT variant of UMC 40 nm technology is used. The use of ULVt variant can give better frequency result, but it also increases the power losses. Similarly, HVT gives better power results by reducing the leakage but slows down the design. Hence, the RVT variant is used to get a balance between power and speed. From Table 3.3, it can be seen that the energy efficiency of the proposed core is much higher than the other existing micro-architectures. These benchmark values are also compared with some commercial Intel and ARM single-core processors in Figure 3.6. The graph shows that the proposed core outperforms many commercial single-core processors. The comparison of the proposed design with existing micro-architectures also shows an improvement in power, performance and area efficiency of the proposed design. Therefore, this micro-architecture can be used for low-power applications like IoTs, which demands high performance at low power.

3.3.2 FPGA

The design is primarily implemented for ASIC, but for initial hardware verification, the design is tested on an FPGA platform, Xilinx Virtex-7 board. For FPGA verification, the core part remains untouched; however, the cache memory part is modified depending on the resources available on the board. The FPGA module has a dual-port memory of 128 KB for data as well as for instructions. This memory is implemented in Block RAM of FPGA, and depending on the available resources, it is limited to 128 KB. The processor has instruction start address at 0x00000200 and runs at a clock frequency of 100 MHz. The core functionality is first verified with some basic C programs like addition, multiplication, factorial, loops, string, etc. Once the functionality of the core is verified, two popular benchmark programs, i.e., Dhrystone

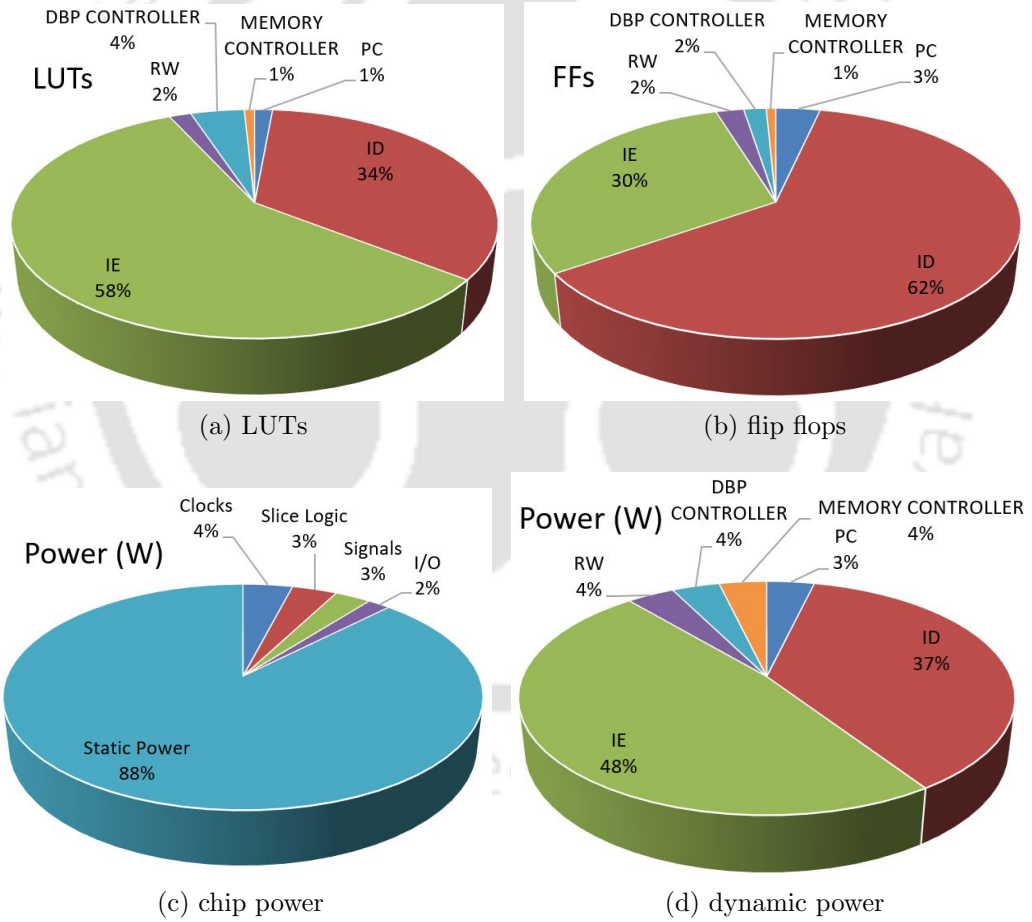
Table 3.4: FPGA synthesis result of the proposed low-power core.

processor	LUTs	FFs	power (mW/MHz)	DMIPS per MHz	CoreMark per MHz
ShaktiC32 [1]*	18613	7722	12.2	0.52	1.85
ShaktiE32 [1]*	10651	1652			
PULPino [2]*	15406	9756	18.37	1.21	2.8
Rocket [36]*	30773	14424		1.72	
River*	28720	10563	8.13	0.33	
VexRiscv max perf [16]*	1813	1424		1.44	
VexRiscv with MMU [16]*	2070	1913		1.26	
proposed	7617	2319	3.4	1.71	4.13

* These results are not claimed by the core developers. These results are obtained by downloading the cores from repositories and verifying them in the required environments.

and CoreMark are run on the core to measure its performance. All of these programs are in C and compiled using riscv-gnu-toolchain, a RISC-V cross-compiler, to generate binary of the programs. The binary data is uploaded to the memory of the core on FPGA to execute. The FPGA synthesis results show that the core can be implemented with 7617 LUTs and 2319 FFs. The timing report shows that the core has a maximum delay path of 7.427 ns, which means it can go up to a maximum frequency of 134 MHz on Virtex-7. The design has a power requirement of 3.4 mW/MHz. Figure 3.7 shows the distribution of hardware resources and power in different components of the proposed design.

For comparison, the FPGA result of the proposed micro-architecture is compared with some open-source cores in Table 3.4. From the table, it can be seen that the LUTs and FFs count of the proposed core is much less compared to [1], [2], [36]. The proposed core shows a Dhrystone score of 1.71 DMIPS/MHz and a CoreMark score of 4.13 CoreMark/MHz. The proposed core outperforms all the cores in terms of CoreMark score. Dhrystone score is comparable to [36] and higher than all others. Figure 3.8 shows a picture of the FPGA setup used during the hardware verification.



(a) LUTs (b) flip flops

(c) chip power (d) dynamic power

Figure 3.7: Distribution of FPGA power and hardware resources for the proposed low-power core.

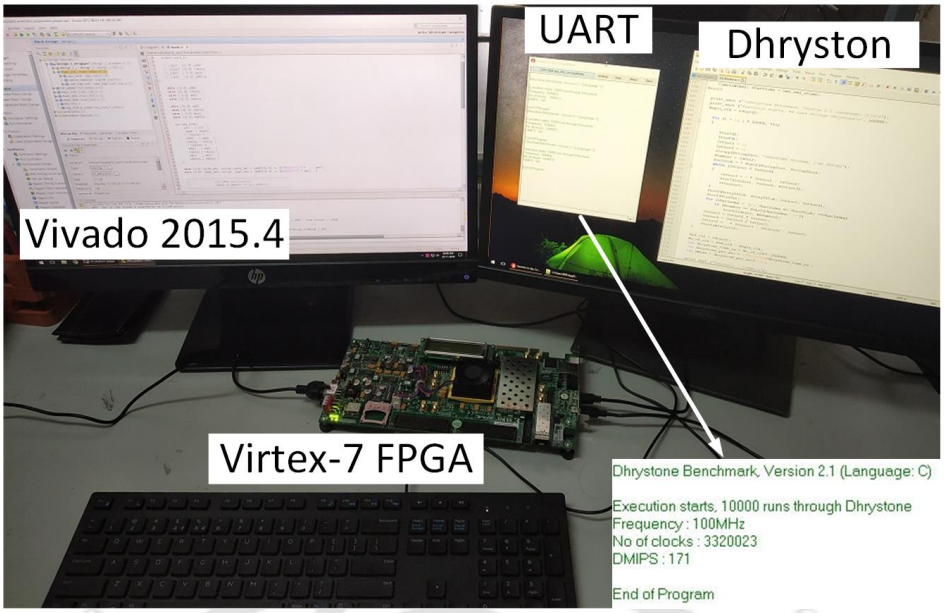


Figure 3.8: FPGA verification set-up.

3.4 Conclusion

This work presents a 4-stage pipelined micro-architecture for low-power applications. The core supports full base integer instruction set of RISC-V along with multiplication and division instructions. The core micro-architecture is arranged in such a way that the execution of instructions can be carried out with a minimum possibility of data, control and structural hazards. Since low-power application is the target of this work, the core is designed with minimum essential features. The work has mainly focused on optimizing the area, power, delay and performance of hardware circuits for multiplier, divider, DBP, etc. and optimizing their control in the pipeline. The proposed core is suitable for applications where we need high-performance with a low-power requirement. Among the benchmarks related to these applications, such as Dhrystone, CoreMark, the proposed core shows better performance by consuming lesser power. This micro-architecture can be used as the base for RISC-V processor design, and more extensions can be added to it depending on specific applications.

In this work, a micro-architecture has been proposed which is highly efficient in terms of power and performance. This can be used as the base for any RISC-V processor, and other extensions like atomic and floating-point can be added to the proposed micro-architecture depending on applications. Memory management is a vital part of processor design, but this work did not concentrate on that. In future, the proposed core can be integrated with some better memory modules to get improved performance. Also, integration of external peripherals like I2C, USB, DDR3 etc., could be another future perspective of this work.

CHAPTER

4

DESIGN AND IMPLEMENTATION OF ADAPTIVE BINARY DIVIDER FOR FIXED-POINT AND FLOATING-POINT NUMBERS.

Contents

4.1 Introduction 39
4.2 Literature Review 40
4.3 Algorithm for Fixed-point Binary Divider 40
4.4 Hardware Architecture of Fixed-point Binary Divider 42
4.5 Single-precision Floating-point Divider 44
4.6 FPGA Synthesis 46
4.7 ASIC Synthesis 47
4.8 Core Integration 49
4.9 Conclusion 51

4.1 Introduction

Division is one of the most important components in engineering applications, such as computer arithmetic, signal and image processing, etc. It is standard hardware found in every sophisticated high-speed digital system. In modern world devices, digital signal processing (DSP) algorithms have a vital role in data processing. The hardware implementation of DSP algorithms consists of several arithmetic units like adder, subtractor, multiplier [53], divider, etc. Among these arithmetic units, divider is the most expensive in terms of hardware resources and complexity. Division operations are typically performed sequentially, making them expensive in contrast with other arithmetic operations, such as addition, subtraction, multiplication, etc., in terms of hardware complexity and latency (propagation delay) [54].

The design of arithmetic and logical unit (ALU) is an integral part of processor design. ALUs are used to perform arithmetic as well as logical operations. The common logical operations (AND, OR, XOR, SHIFT) are implemented using bitwise processing, where each bit is processed independently. Therefore, the delay of such implementation is the delay of a single gate which is minimal. However, the hardware implementation of arithmetic operations is much more complex. The processing of higher bits depends on the results of lower bits. The delay of these circuits is a significant concern. So, designing efficient hardware circuits for arithmetic operations is very important to get optimum performance of the overall system. There are a lot of scientific papers on the hardware implementation of addition, subtraction and multiplication. However, there are very few papers on division algorithms [55] and their hardware implementation [56–62]. Division is an essential part of designing hardware circuits for any system and is more complex than adder and multiplier circuits [63]. So, a fast, efficient divider circuit is crucial for all system designs. In [64], a radix-2 digit-by-digit division algorithm is presented which generates quotient digits by observing the two most-significant radix 2 digits of the partial remainder. According to the authors, the design is area efficient but slightly slower than designs with input scaling. In another divider named as New Svoboda-Tung (NST) [65], which is a modified version of the Svoboda-Tung divider. Both the dividers are very area-efficient, but the slow in terms of latency as the quotient bits are generated one after another.

In this chapter, an efficient hardware implementation is proposed for binary division. In general, division involves a lot of iterations and hence, becomes very slow. So, in this work, the division process is made faster by reducing the number of iterations. The term 'adaptive' is used for the proposed architecture to imply that the latency of the circuit may vary depending on the inputs. The work is also extended to the design of a single-precision floating-point divider. All the hardware architectures are designed using Verilog HDL, and the functionality is verified in Xilinx Vivado 2015.4. The hardware designs are synthesised for Virtex-7 (vc707) FPGA platform. The designs are also compatible with ASIC, synthesised in Synopsys design compiler (DC) at 40 nm technology node.

The chapter is organised as follows: Section 4.2 throws some light upon the different existing divider architectures in the literature. In Section 4.3, the theory behind the proposed divider algorithm is explained. Section 4.4 describes the hardware implementation of the proposed algorithm. Section 4.5 elucidates the floating-point divider and its hardware implementation. In Sections 4.6 and 4.7, synthesis results of FPGA and ASIC are reported and analysed. Finally, the conclusion of the work is drawn in Section 4.9.

4.2 Literature Review

Some of the widely used division algorithms available in literature are restoring [66–69], non-restoring [56, 58], SRT (Sweeney, Robertson, and Tocher)[70], Newton-Raphson[71], Goldschmidt[72] etc. In restoring and non-restoring algorithms, one quotient bit is generated every cycle. The divisor is subtracted/added to the dividend every cycle, and the quotient bit is predicted by the sign bit. The SRT division is a fast algorithm for two's complement numbers based on the technique of shifting over zeros. Newton-Raphson division is an approximate method. It uses Newton's method to find the reciprocal of the divisor and multiply the reciprocal by dividend to find the final quotient. The reciprocal computation of divisor can be considered as a special type of division where the dividend has a fixed value of one. This reciprocal computation unit plays a crucial role in the hardware implementation of Newton-Raphson algorithm. In another algorithm, the divisor and dividend are scaled using a common factor. The factors are selected such that the divisor converges to one and the numerator converges to the quotient. This computation of quotient is named as Goldschmidt computational division algorithm. This algorithm is an iterative process that uses several complex operations for division, where the precision is to be maintained high for very large data intervals.

There are some hardware implementations of fixed-point dividers in the literature. In [62], a divergent approach is proposed for division algorithm using simulated annealing algorithm. The algorithm is implemented and verified on a Xilinx Virtex-6 FPGA platform. The design outperforms many existing algorithms in terms of the number of iterations required to obtain the final quotient and generation of partial remainders. There are many algorithms in Vedic mathematics that can offer a computational advantage over conventional arithmetic algorithms. A purely combinational implementation of a divider is proposed in [73], based on the Dhwajanka Sutra of Vedic mathematics. In [74], a fixed-point divider is implemented using ensemble of moving average (EnMA) curves. The EnMA algorithm is used to generates the curve $y = 1/x$, and the curve values are used to decrease multiplications and subtractions required for the division.

Similar to a fixed-point divider, the floating-point divider is also used in many arithmetic and logical units and in many digital signal processing systems. An efficient implementation of a floating-point divider can enhance the performance of these systems. In [75], Goldschmidt fixed-point division algorithm is used to implement a single-precision floating-point division. In another approach [76], a dual-mode radix-4 modified Booth multiplier is used to implement the divider architecture. This architecture supports dual-mode functionality, which can either compute on a pair of double-precision operands or two pairs of single-precision operands in parallel. In this work, we proposed a novel division algorithm and its hardware implementation for fixed-point and floating-point numbers.

4.3 Algorithm for Fixed-point Binary Divider

The division algorithm discussed in this section is an iterative one. The divisor is compared with and subtracted from the dividend. This process iterates till the subtraction result becomes lesser than the divisor. For example, consider two 32-bit binary numbers, A and B. $A = (a_{n-1}a_{n-2}a_{n-3}...a_1a_0)$, $B = (b_{n-1}b_{n-2}b_{n-3}...b_1b_0)$, where n is 32. For radix-16 implementation of A divided by B, the first step is partitioning the dividend into eight groups $G_i = (a_{4i+3}a_{4i+2}a_{4i+1}a_{4i}), i = 0, 1, 2, ...7$. Then, G_7 is compared with multiples of the divisor,

and the closest smaller one is subtracted from G_7. These multipliers can be any number between 0 to 15. The quotient is updated with the correct multiplier value, and the subtraction result is padded on the left of G_6. This modified G_6 is then compared with multiples of the divisor. Based on the comparison result, the closest value is subtracted from G_6, and the resulting quotient is updated with the multiplier. This process continues for all the eight groups, G_i.

Algorithm 2 32-bit radix-16 divider

```
INPUT1 = Dividend = A = (a_{n-1}a_{n-2}a_{n-3}...a_1a_0)
INPUT2 = Divisor = B = (b_{n-1}b_{n-2}b_{n-3}...b_1b_0)
Radix = 16; R_n = log2(Radix)
for i = 0 to (32/R_n) - 1 do
  G_i = a_{4i+3}a_{4i+2}a_{4i+1}a_{4i}
end for
M = most significant one position
R = 0; Q = 0; N = M / 4
for i = N to 0 do
  R = CONCATENATE (R, G_i)
  for j = Radix-1 to 0 do
    if j×Divisor ≤ R then
      R = R - j×Divisor
      EXIT LOOP
    end if
  end for
  Q = CONCATENATE (Q, j)
end for
OUTPUT1 = Quotient = Q
OUTPUT2 = Remainder = R
```

Algorithm 2 shows the pseudo-code for the proposed algorithm for a 32-bit radix-16 divider. In a 32-bit radix-16 divider, the comparison and subtraction are executed for eight iterations. The number of iterations is entirely dependent on the size of the dividend. In radix-16, the dividend is partitioned into groups of 4-bits each, and the number of groups determines the number of iterations. For example, for a 64-bit dividend, the number of iterations would be 16. Similarly, for a 32-bit dividend, it would be 8. However, this number of iterations can be reduced depending on the position of the most significant one (MSO) of the dividend. If the MSO lies in the second group of the dividend, i.e. in between the 24^{th} to 27^{th} bit, then the number can be considered as a 28-bit number. So, the first iteration can be skipped, and the division requires seven iterations to complete. Similarly, if the first eight bits of the dividend are zero and the MSO lies between 20^{th} to 23^{rd} bit, then the number of iterations would be six. In this way, the number of iterations can vary from 1 to 8 depending on the value of the dividend. This number of iterations determines the latency of the divider, which can vary from 2 to 9 clock cycles for a 32-bit radix-16 divider. In Algorithm 2, the iteration count (N) is determined by dividing the position of the MSO by 4. As we go towards higher radix implementation, the latency or the iterations count of the algorithm decreases but the complexity increases. Because of this, the cost of the hardware implementation of the algorithm increases in terms of area, power and delay.

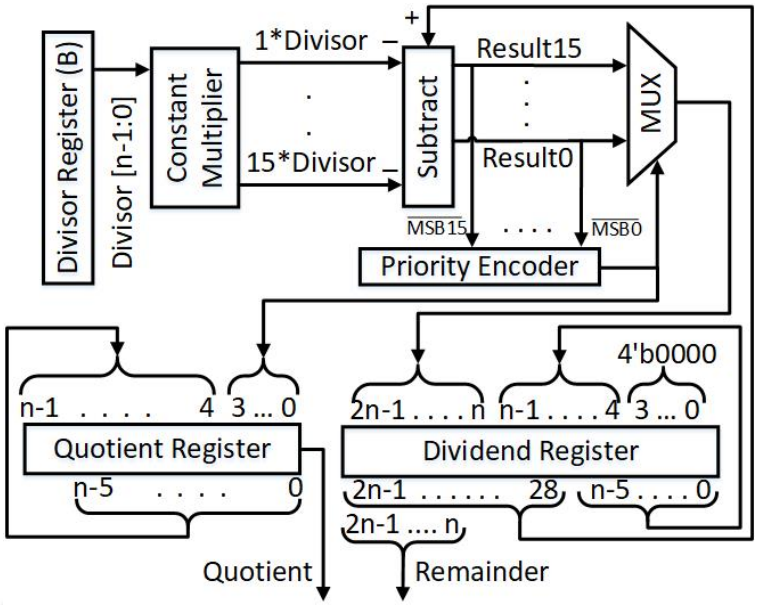


Figure 4.1: Radix-16 unsigned divider, n is 32 for a 32-bit divider and 24 for a 24-bit divider which is used in floating-point divider (Section 4.5).

4.4 Hardware Architecture of Fixed-point Binary Divider

This section describes the hardware implementation of a 32-bit radix-16 divider. The block diagram of the same is shown in Figure 4.1. It has five major stages, namely, register initialisation, multiplication, comparison, subtraction and shift. These stages are discussed in detail in the following part of this section.

The aim of the first stage of the algorithm is to initialise the registers. Although it looks like a simple step, the latency of the circuit is highly dependent on this stage. The proposed architecture has four registers, A (dividend), B(divisor), Q(quotient) and N(Iteration Count). Registers B and Q are always initialised with divisor value and zero, respectively. However, the initialisation of A and N varies depending on the value of the dividend. For a radix-16 32-bit divider, A is a 64-bit register where higher 32-bits are always initialised with zeros. The lower 32-bits are initialised with dividend's bits depending on the most significant 1's position. If the most significant 1's position is higher than 27, then the A is initialised with dividend directly; if it lies between 24^{th} and 27^{th} bit, then the dividend is left-shifted by 4-bits and initialised to A; and so on. The corresponding iteration count of the circuit is 8, 7, and so on. One extra clock cycle is utilised at the beginning for the initialisation of the registers. Hence, the latency of the circuits is 9 clock cycles, 8 clock cycles, and so on.

Figure 4.2 shows the hardware circuit used in initialising registers A and N. As discussed in section 4.3, the dividend is first partitioned into groups of 4-bits, then the number of iterations required and the number of left shifts required before dividend are determined using a priority encoder and four-input OR gates as shown in the figure. The depicted multiplier is a constant multiplier where the multiplicand is the 3-bit output of the priority encoder multiplied by 4. This multiplication is equivalent to a 2-bit logical left shifter. So, the hardware resources can be reduced by replacing the multiplier with a 2-bit logical left shifter which is nothing but appending two zeros to the right of the LSB.

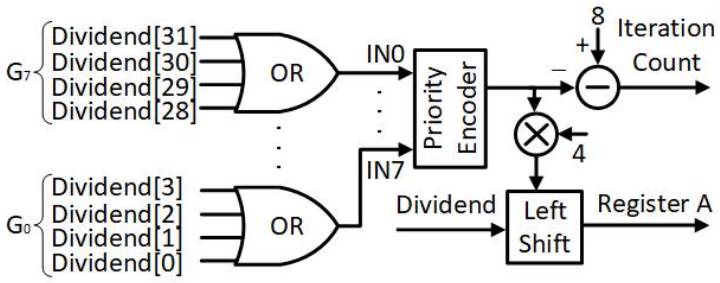


Figure 4.2: Iteration count determination circuit for a 32-bit dividend in radix-16 implementation

In the second stage, divisor register B is multiplied with constants. Many multipliers are required to compute these multiplications, which are highly expensive in area, delay and power. So, while implementing the hardware circuits, these multiplications are implemented using shifters, adders and subtractors. Multiples of 2,4,8 are calculated by left shifting the divisor through 1-bit, 2-bit and 3-bit, respectively. In general, shifters are implemented by appending zeros to the LSB and has near-zero delay. So, the use of shifters is preferred in the implementation of the constant multiplier circuit. For example, 15×divisor can be calculated by adding 9×divisor with 6×divisor or subtracting divisor from 16×divisor. The first process involves two adders, 35-bit and 36-bit, in the critical path, whereas the second involves only one 36-bit subtractor. This is because 9×divisor is the addition of 8×divisor and 1×divisor, and 6×divisor is the addition of 4×divisor and 2×divisor. So, the subtraction of divisor from 16×divisor is prioritised in the implementation of 15×divisor. This whole process of shift, add and subtract is represented in Figure 4.3. The critical path of the circuit is found to be 2.85ns at 40 nm technology node, which is shown by the red colour line in Figure 4.3.

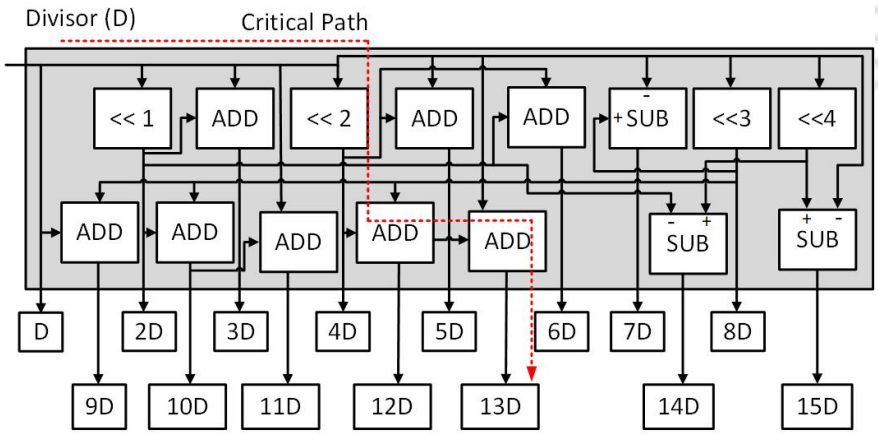


Figure 4.3: Constant multiplier in radix-16 divider

The third and fourth stages are for comparison and subtraction. In this stage, the multiples of B are first compared with higher bits of A. Then, depending on the comparison result, the closest one is subtracted from the higher bits of A. However, it has been observed that if subtraction is performed prior to comparison, then from the MSB of the subtraction results, the closest one can be determined using a priority encoder and multiplexer. This leads to an increase in hardware resources, but the delay of the circuit reduces by a huge amount which is a more significant parameter in divider circuits. As shown in Figure 4.1, the multiples of divisor are first subtracted from the most significant 36-bits of A (MSA). The

inverted MSBs of the subtraction results are then used as inputs to the priority encoder, and the output of the priority encoder is used to multiplex the correct subtraction result to the dividend register. The priority encoder output also gives the lower 4-bits of the quotient, as shown in Figure 4.1. At every clock cycle, the MSA is subtracted from the pre-calculated multiples of dividend and MSBs are encoded using a priority encoder. If the encoder output is 8, then the result (i.e., MSA - 8×B) is stored in dividend register (A), and the 4-bit binary equivalent of 4 is stored in quotient register (Q). This process continues until the number of iterations count becomes zero. Finally, the quotient is given by Q register, and the remainder is given by the 28th to 59th bits of register A. The performance of this proposed hardware architecture is analysed in FPGA and ASIC. The results are discussed in section 4.6 and section 4.7.

4.5 Single-precision Floating-point Divider

In many financial and commercial applications, the accuracy of a divider is as important as its speed. Fixed-point dividers are not a very good choice for these applications because they can not represent fractional decimal numbers. A floating-point divider can provide better accuracy than a fixed-point divider. In addition, a floating-point divider provides higher versatility which allows users to deal with a wide range of data values. So, in this section, the proposed concept of a fixed-point divider is used to design a single-precision floating-point (FP) divider. The proposed floating-point (FP) divider is based on modified IEEE-754 Standard [77]. In single-precision formats, there are three parts of a 32-bit floating-point number, namely Sign Bit, Exponent (8-bit) and Mantissa (23-bit), as shown in Figure 4.4.

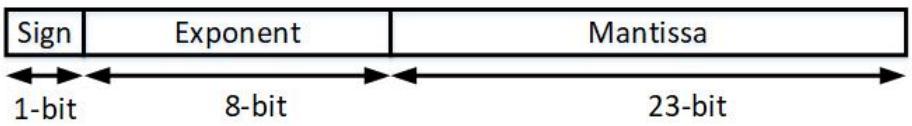


Figure 4.4: IEEE-754 single-precision floating-point format

Similar to fixed-point (FXP) division, floating-point (FP) dividers can also be classified into two major families, digit recurrence and functional iteration [78]. Digit recurrence algorithms are iteration-based methods, and they are very slow. Some of them produce a single output bit in a single clock cycle. On the other hand, functional iteration algorithms are faster, but they are approximate methods. Hence, these algorithms are lesser accurate. The accuracy of these algorithms can be improved by increasing the number of iterations. However, this leads to more complexity and more delay of the circuit.

The proposed single-precision floating-point divider is implemented using a 24-bit fixed-point divider. Algorithm 3 shows the proposed floating-point divider step by step. First of all, the sign, exponent and mantissa of numerator and denominator are separated out. The mantissa of the dividend is then shifted left until the most significant bit (MSB) becomes one, and the exponent is also adjusted accordingly. This step is done to get a higher value of mantissa of the result, which leads to better accuracy. Once the dividend mantissa is available, it is divided by the 24-bit mantissa of the divisor. This 24-bit division is same as the proposed fixed-point divider. The sign of the result is decided by the signs of dividend and divisor, as shown in Algorithm 3. Similarly, the mantissa of the result is obtained by adding 127 to the difference between dividend mantissa and divisor mantissa. Once the three

Algorithm 3 Single-precision Radix-16 floating-point divider

INPUT1 = N[31:0]
INPUT2 = D[31:0]
N_{sign} = N[31]; N_{exp} = N[30 : 23]; $N_{mantissa}$ = N[22 : 0]
D_{sign} = D[31]; D_{exp} = D[30 : 23]; $D_{mantissa}$ = D[22 : 0]
if N_{sign} = 00000000 **then**
  x = CONCATENATE(0, $N_{mantissa}$); $N_{exp} = -126$
else
  x = CONCATENATE(1, $N_{mantissa}$)
end if
if D_{sign} = 00000000 **then**
  y = CONCATENATE(0, $D_{mantissa}$); $D_{exp} = -126$
else
  y = CONCATENATE(1, $D_{mantissa}$)
end if
for i = 31 to 0 **do**
  if $N_{mantissa}$[23] = 1 **then**
    EXIT LOOP
  else
    SHIFT LEFT ($N_{mantissa}$[23]); $N_{exp} = N_{exp} - 1$
  end if
end for
for i = 0 to 5 **do**
  $G_i = x_{4i+3}x_{4i+2}x_{4i+1}x_{4i}$
end for
R = 0; Q = 0
for i = 5 to 0 **do**
  R = CONCATENATE (R, G_i)
  for j = 15 to 0 **do**
    if j×y ≤ R **then**
      R = R - j×y
      EXIT LOOP
    end if
  end for
  Q = CONCATENATE (Q, j)
end for
Mantissa = Q
Sign = N_{sign} XOR D_{sign}
Exponent = $N_{exp} - D_{exp} + 127$
OUTPUT = ERROR CORRECTION(Sign, Exponent, Mantissa)

components are obtained, they are adjusted and arranged in IEEE-754 standard format to give the final result.

Figure 4.5 shows the block diagram of the hardware implementation of the proposed single-precision floating-point divider. It has three major steps: initialisation, division and error correction. In the initialisation step, registers are initialised with sign, exponent and mantissa of dividend and divisor. During initialisation, the mantissa of dividend is shifted

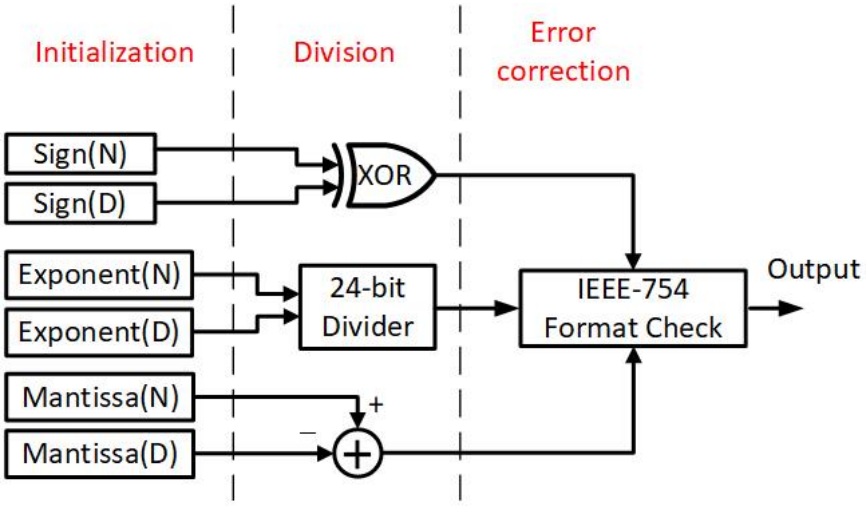


Figure 4.5: Block diagram of the proposed floating-point divider

to the left until the MSB becomes one. This step results in the maximum accuracy of the quotient in a minimum number of iterations. Another approach to increase the accuracy can be an increase in the number of iterations. However, this also increases the latency of the circuit. In the second step, the 24-bit mantissa of the dividend is divided by the 24-bit mantissa of the divisor. The quotient of the division gives us the mantissa of the floating-point divider. Figure 4.1 shows the hardware circuit of this 24-bit divider. In the third step, appropriate adjustments of the result are made to represent the result in IEEE-754 standard format.

4.6 FPGA Synthesis

The proposed algorithm is implemented using Verilog hardware description language (HDL). The implementations are synthesised and verified on Xilinx Virtex-7 FPGA platform. The chip used in this platform is XC7VX485T which is fabricated at 28nm technology node. The tool used is 2015.4 version of Xilinx Vivado. Performance parameters in terms of clock frequency, LUT, slice registers and latency have been observed and reported in Table 4.1.

Table 4.1: FPGA synthesis result of 32-bit radix-16 divider.

Algorithm	[59]	[60]	[61]	[62]	proposed
LUTs	4172	2704	1152	773	1271
slice registers	2275	1713	687	464	140
critical path delay (ns)	102.5	74.6	85	43.8	4.8
latency (clock cycle)	1	1	1	1	2(min) 9(max)
total delay (ns)	102.5	74.6	85	43.8	9.6(min) 43.2(max)

* There is no use of BRAM or DSP resource of FPGA in all the designs.

For functional verification, the design is simulated for random inputs using Vivado Verilog simulator. The synthesis result of the design shows that the proposed architecture can be implemented using 1271 LUTs and 140 slice registers. The design has a critical path delay of 4.8ns and latency of 2(min) to 9(max) clock cycles. In Table 4.1, the latency of the proposed design is higher than the existing designs. However, the main advantage of the proposed architecture is its lower critical path delay, because of which it can be used in high-speed circuits. The total delay of the circuit also does not seem to be very high. The total delay may vary from 9.6ns (2×4.8ns) to 43.2ns (9×4.8ns), depending on the value of the inputs. The proposed floating-point divider of Section 4.5 is also synthesised to analyse its performance. According to the synthesis result, the floating-point divider can be implemented using 980 LUTs and 144 slice registers. The circuit has a delay of 4.325ns and a maximum latency of 8 clock cycles.

4.7 ASIC Synthesis

The ASIC synthesis of the proposed design is carried out in a commercial synthesis tool, Synopsys DC (design compiler). Table 4.2 shows a comparison of the proposed design with some state-of-the-art works. The existing results are available at 45 nm and 130 nm technology nodes. Therefore, the proposed design is synthesised at 40 nm and 130 nm technology nodes to make the comparison meaningful.

At 40 nm technology, the proposed architecture consumes 3938μm^2 area with a critical path delay of 4.97ns and 2.82$\mu W/MHz$ dynamic power. From Table 4.2, [56] has total delay of 85.71ns, and the design of [58] has a total delay of 88.29ns whereas the total execution time for the proposed architecture varies from 9.94ns (2×4.97ns) to 44.73ns (9×4.97ns). The maximum possible total delay of the proposed design is almost half compared to [56] and [58]. If we compare with [80], the proposed design requires more area, but it also provides

Table 4.2: ASIC synthesis results of 32-bit and 16-bit radix-16 divider.

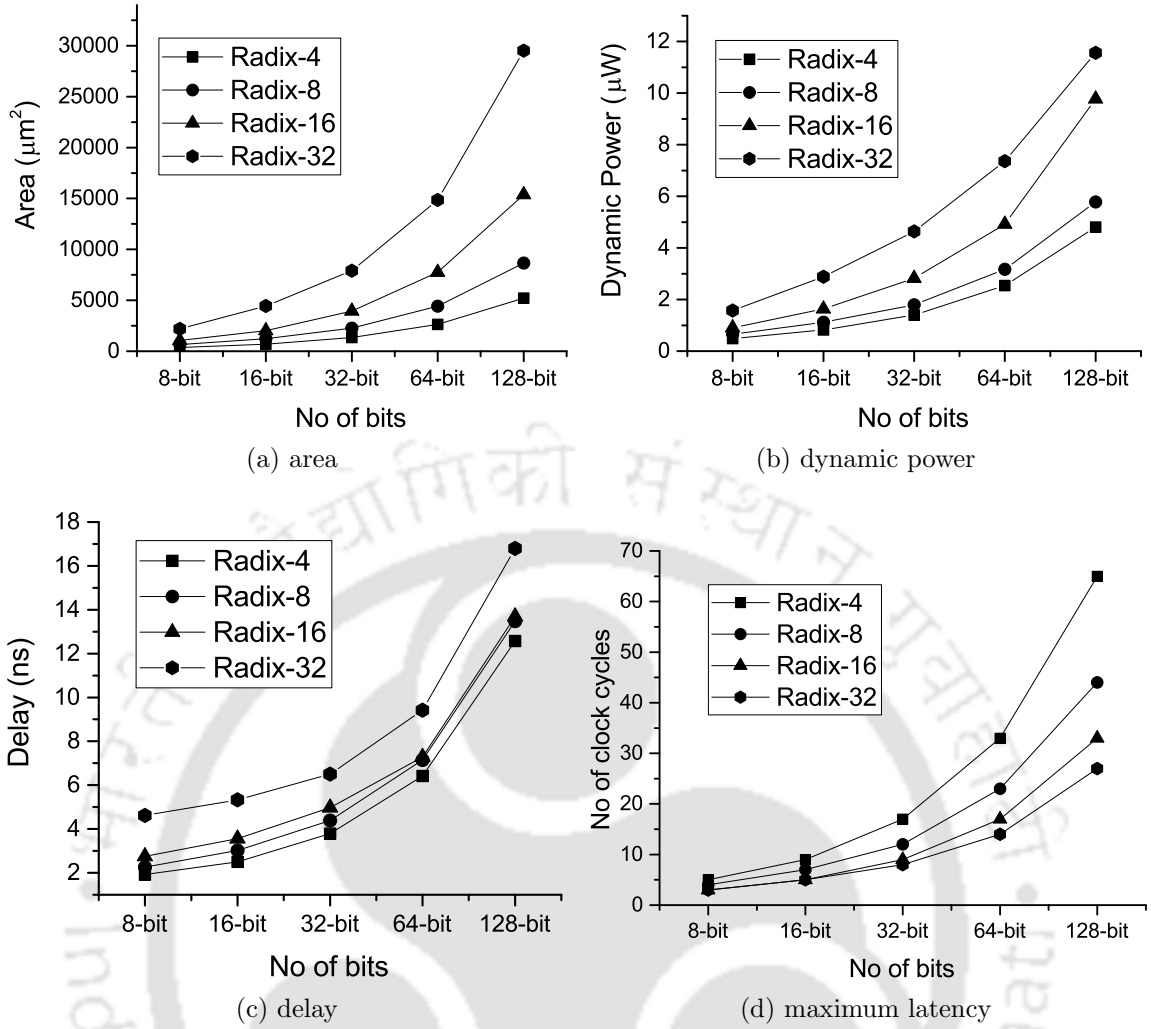
algorithm	**data width**	**technology node**	**area (μm^2)**	**critical path delay (ns)**	**latency (clock cycles)**	**total delay (ns)**
SAADI-EC [79]	16	45	6105	1.57	1(min) 15(max)	1.57(min) 23.55(max)
SAADI [79]	16	45	4872	1.60	1(min) 15(max)	1.6(min) 24.0(max)
[56]	32	45	50132	85.71	1	85.71
[58]	32	45	72945	88.29	1	88.29
[80]	32	130	11534	3.69	33	121.77
Proposed	16	40 nm	2022	3.56	2(min) 5(max)	7.12(min) 17.8(max)
	32	40 nm	3938	4.97	2(min) 9(max)	9.94(min) 44.73(max)
	32	130 nm	28929	5.21	2(min) 9(max)	10.42(min) 46.89(max)

better latency and delay. [80] takes almost 3× more time than the maximum delay of the proposed design. In [79], few designs are reported with latency varying from 1(min) to 15(max). Upon comparing with our 16-bit design, it is clear that the proposed design is much faster than the existing ones. The proposed design has lower latency which varies from 2(min) to 5(max). Also, the maximum delay of the proposed design is 17.8ns which is 24% lesser than SAADI-EC [79] and 26% lesser than SAADI [79]. So, from Table 4.2, it is clear that the proposed architecture is faster than many existing architectures. Also, because of the lower critical path delay, the circuit can work up to a maximum frequency of 201MHz (for 4.97ns). The proposed floating-point divider of Section 4.5 is also synthesised at 40 nm to analyse its performance. The circuit requires 3353μm^2 area and 2.76$\mu W/MHz$ power. The delay of the design is 4.83ns, and the latency is 8 clock cycles.

Table 4.3: Power, area, delay and latency of the proposed divider circuit for different data width and radix values at 40 nm.

No. of bits	Radix	Area (μm^2)	Power (μW/MHz)	Delay (ns)	Min Latency	Max Latency
8-bit	Radix-4	363.21	0.49	1.92	2	5
	Radix-8	654.27	0.65	2.26	2	4
	Radix-16	1,064.57	0.90	2.75	2	3
	Radix-32	2,194.59	1.57	4.62	2	3
16-bit	Radix-4	687.25	0.82	2.50	2	9
	Radix-8	1,230.74	1.11	3.02	2	7
	Radix-16	2,022.96	1.63	3.56	2	5
	Radix-32	4,441.75	2.88	5.33	2	5
32-bit	Radix-4	1,341.52	1.40	3.79	2	17
	Radix-8	2,262.51	1.79	4.39	2	12
	Radix-16	3,938.66	2.82	4.97	2	9
	Radix-32	7,914.19	4.64	6.50	2	8
64-bit	Radix-4	2,628.71	2.54	6.42	2	33
	Radix-8	4,426.05	3.17	7.15	2	23
	Radix-16	7,745.02	4.92	7.28	2	17
	Radix-32	14,855.00	7.36	9.42	2	14
128-bit	Radix-4	5,212.97	4.80	12.57	2	65
	Radix-8	8,651.01	5.78	13.48	2	44
	Radix-16	15,376.44	9.77	13.69	2	33
	Radix-32	29,509.96	11.55	16.80	2	27

The work is further extended to implementation for different data widths and analysis of the same at 40 nm technology node. The algorithm is also analysed for different radix implementations. Table 4.3 shows the area, power, delay and latency of these implementations at UMC 40 nm technology node. The data widths used for this analysis are 8, 16, 32, 64, and 128 bit. Each of these data widths is further analysed for radix values 4, 8, 16, and 32. The designs are implemented using Verilog HDL and synthesised in Synopsys DC. Figure 4.6 shows the graphical representation of the synthesis results. It can be observed that the area, power, delay, and latency increase exponentially for a higher number of data



(a) area (b) dynamic power

(c) delay (d) maximum latency

Figure 4.6: Area, power, delay and latency of the proposed architecture vs. data width for different radix values.

bits. One another observation from the Figure 4.6c is that the rate of increase of delay is almost equal for radix-4, radix-8, radix-16 and radix-32. However, the rate of increase in power and area is significant for a higher radix implementation. On the other hand, the rate of increase in latency decreases as we go for a higher radix implementation. So, from all the graphs of Figure 4.6, it can be concluded that for large data widths, the latency of the circuit can be reduced by a considerable amount by increasing the radix value. This leads to an increase in area and power. However, the delay increases by a minimal amount, as shown in Figure 4.6c.

4.8 Core Integration

The proposed divider circuits are integrated into a RISC-V core (Chapter 3) to support floating-point numbers. The core is first verified on a Xilinx Virtex-7 FPGA board. The FPGA module has a memory of 128 KB for data as well as for instructions. This memory is implemented in Block RAM of FPGA, and because of device limitation, it is limited to 128 KB. The processor has instruction start address at 0x00000200. The core is also synthesised

using Synopsys Design Compiler (DC) at UMC 40 nm technology node for better estimation of area, delay and power.

FPGA result

The proposed core is implemented on a Xilinx Verte7 FPGA board at 100MHz, and the functionality of the core is verified for different using C programs. These C programs include arithmetic function, loops, case, array, string, pointer, function etc. In the functional verification process, C codes are first compiled using riscv-gnu-toolchain, a RISC-V cross-compiler and the binary data is uploaded to block ram of FPGA. To display the results of these C codes, UART is used. Only the transmitter module of UART is implemented and used in this processor. The receiver module for UART is not implemented as it is not needed for the verification process. But, later, it can be added to the processor. The FPGA module of the proposed micro-architecture has the following specifications:

- 128 KB of Cache Memory
- Instruction start address at 0x00000200
- UART read address at 0x0001FFF0

For comparison, the FPGA result of the proposed micro-architecture is compared with some open-source cores in Table 4.4. From the table, it can be seen that area of the proposed core is much lesser compared to Shakti[1], pulpino[2], rocket[36]. If we look at the performance of these cores, the proposed core outperforms ShaktiC32 in terms of Dhrystone score as well as CoreMark score. CoreMark score of the proposed micro-architecture, i.e. 4.13 CoreMark/MHz, is much higher than the CoreMark score of PULPino [2] i.e. 2.8 CoreMark/MHz. The Dhrystone score, i.e. 1.71 DMIPS/MHz, is also higher than pulpino[2] and VexRiscv[16].

Table 4.4: FPGA synthesis result of the proposed designs after core integration.

processor	**LUTs**	**FFs**	**power (mW/MHz)**	**DMIPS per MHz**	**CoreMark per MHz**
ShaktiC32 [1]*	18613	7722	12.2	0.52	1.85
ShaktiE32 [1]*	10651	1652			
PULPino [2]*	15406	9756	18.37	1.21	2.8
Rocket [36]*	30773	14424		1.72	
River*	28720	10563	8.13	0.33	
VexRiscv max perf [16]*	1813	1424		1.44	
VexRiscv with MMU [16]*	2070	1913		1.26	
proposed without FP	7617	2319	3.4	1.71	4.13
proposed with FP	8379	2557		1.71	4.13

* These results are not claimed by the core developers. These results are obtained by downloading the cores from repositories and verifying them in the required environments.

ASIC result

The proposed micro-architecture is synthesised using Synopsys DC at UMC 40 nm technology nodes. The place and route of the synthesised design is done in Cadence Innovus, and Synopsys PrimeTime tool is used for the post layout estimation of power and delay. At UMC 40 nm technology node, the core occupies an area of 0.036 mm^2 which consists of 20 K cells and 2.7 K buffers which is almost half of the area occupied by [11]. The critical path delay of the proposed micro-architecture is 5.05 ns which means the core could achieve a maximum frequency of 198 MHz. The core consumes 173.6 µW dynamic power while operating at a clock frequency of 10 MHz. This dynamic power consumption increases as the clock frequency of the system increases. So, to make a fair comparison among the cores, dynamic power consumption per MHz is calculated. The normalised value of dynamic power of the proposed micro-architecture is 17.36 µW/MHz. A comparison of the proposed core results with existing works is shown in Table 4.5.

Table 4.5: ASIC implementation results of the proposed designs after core integrating.

Processor	**Technology**	**Area without cache**	**Max Frequency**	**Dynamic Power**	**CoreMark per MHz**	**DMIPS per MHz**
Shakti-F[12] (RV32I, mul)	55nm	30K	333MHz(3ns)	35.25µW/MHz (11.74 mW)		
RISC-V Basic[11]	65 nm	46.9KGE 0.068mm^2	357MHz	26.28µW/MHz	2.43	
RISC-V Extended[11]	65 nm	53.5KGE 0.077mm^2	357MHz	26.68µW/MHz	2.84	
Open RISC[49]	65 nm	44.5KGE 0.064mm^2	362MHz	33.8µW/MHz	2.66	
Rocket[8] 64-bit	TSMC40PLUS	0.14mm^2	>1GHz	34µW/MHz		1.72
RISC-V Vector Processor[7]	28nm FDSOI	1.19mm^2	93-961MHz	8-173 mW		
Cortex-M3[50]	90 nm	0.09mm^2		31µW/MHz	3.34	1.25
Cortex-M4[23]	90 nm	0.119mm^2	216MHz	32.8µW/MHz	3.4	1.25
Cortex-A5[81]	TSMC40PLUS	0.27mm^2	>1GHz	<80µW/MHz		1.57
Proposed without FP	UMC40	20.2K cells 0.036mm^2	5.05ns 198.0MHz	17.36µW/MHz	4.13	1.71
Proposed with FP	UMC40	31.8K cells 0.058mm^2	5.42ns 184.5MHz	33.2µW/MHz	4.13	1.71

4.9 Conclusion

Although divider is one of the most important parts of digital systems, there are very few papers reported in the literature on its hardware implementation. This work presents a hardware architecture for division in various digital systems. The work also explains the implementation of a single-precision floating-point divider using the proposed architecture. The main problem with a divider circuit is propagation delay. So, the proposed work is focused on optimising the delay of the circuit. The work also presents an analysis of the

proposed method for different input data widths (8, 16, 32, 64, and 128). The analysis is further extended to different radix values, 4, 8 and 16. From the analysis, it can be concluded that higher radix implementation of the proposed method is found more suitable for larger data widths. It reduces the latency by a significant amount with a small increment in delay. The proposed divider design is also integrated with a RISC-V core, and the core performance is measured in terms of power, area and delay. Overall, the work presents a novel binary division method that can be used in any digital system and achieve better performance.

CHAPTER

5

DESIGN AND IMPLEMENTATION OF RISC-V CORE FOR DATA-CENTRIC IOT APPLICATIONS

Contents

5.1 Introduction 54
5.2 ISA and its Extension 55
5.3 Micro-architecture 56
5.3.1 Pipeline 57
5.3.2 ALU 57
5.3.3 MAC 58
5.4 Results and Discussion 60
5.4.1 ASIC 60
5.4.2 FPGA 60
5.4.3 Performance Measurement 61
5.5 Conclusion 64

5.1 Introduction

In the last few years, we have seen a lot of small, portable, and battery-powered Internet-of-Things (IoT) endpoint devices. These IoT devices pushed us to a world where all these devices can communicate among them and other devices to provide valuable functionalities. In other words, an IoT endpoint device can sense, listen and see the real world and collects information for further processing and transmission. Such devices have a vast range of applications such as control processing, DSP, machine learning, security, etc. These devices are usually controlled by a microcontroller unit (MCU). Therefore, it is expected that the demand for small, low-power and efficient sensors and processing platforms in the IoT segment will skyrocket in the near future [82].

The IoT endpoint devices are battery-powered, and demand optimised implementation for maximum battery life. These endpoint devices are built around an MCU and allow the integration of multiple sensors. The main use of MCU is controlling and light-load processing. In general, the IoT endpoint devices are portable, and therefore they should be very cheap to maintain and operate. This requires efficient low-power implementation. In such devices, the major contributor to the overall power of the system is the wireless (or wired) transmission of data from the endpoint devices to the cloud [83]. So, it is necessary to contract the amount of data that needs to be transmitted. The sensor data can be reduced by extracting important information from the raw data with the help of more complex algorithms, such as recognition, classifications, feature extraction [84]. The execution of these algorithms demands a high-performance MCU with energy efficiency.

The majority of IoT endpoint devices use single-issue in-order cores for controlling and light-load processing. A single-core processor is typically more energy-efficient and provides higher Instructions per clock (IPC) [85]. Current commercial IoT products often use Cortex-M series ARM processors. A simple single-core processor is very efficient for controlling and light-load processing, but it is neither powerful nor efficient enough to execute complex data processing algorithms [86]. Therefore, many efforts have been put to improve the efficiency and performance of the processor in IoT devices. In [87], additional functional units are added in the core pipeline to enhance the performance. The additional functional units are selected by considering the need of modern IoT devices. A crypto-primitive generation engine is also incorporated for cryptographic applications. The use of additional functional units enables parallel processing via dynamic matching of resources to demand. In recent years, convolutional neural network (CNN) has become a popular algorithm in many applications. A System on Chip (SoC) consists of a RISC-V processor, and CNN accelerator is proposed in [88]. The execution time of standard multiply-accumulate calculations is greatly reduced using a convolutional array of activation functions, which are also used to accelerate the CNN process greatly. One approach to achieve higher performance and energy efficiency is to incorporate digital signal processing (DSP) functionalities into the processor core. In biomedical applications, the transmission costs can be reduced by extracting the heart rate of an ECG signal and transmitting only the extracted features over the communication channel [89]. Higher efficiency in performance can also be achieved with dedicated hardware accelerators. In [90], the speed of Fast Fourier transformation (FFT) is increased using a dedicated hardware accelerator to improve the overall seizure detection process. The dedicated hardware accelerator is controlled by an MCU. The combination of MCU with hardware accelerator is good for achieving higher performance, but such a system is very specialised and hence not very efficient for other applications. As the modern world devices are becoming smarter and smarter, security is also becoming a significant concern. Therefore, IoT devices are

now adapting cryptographic algorithms to secure their data. In [91], a micro-architecture is proposed with in-memory and near-memory computing to support large vector calculation. The core is capable of executing cryptographic algorithms more efficiently than many other existing cores.

In this work, a core micro-architecture is designed with high performance and energy efficiency. We investigated and found that it is possible to build a fast and flexible processing platform that consumes only a few milliwatts. The base ISA used for the hardware implementation is RISC-V ISA [6]. Further, we have optimised the ISA and micro-architecture to achieve better performance and code density than other proprietary ISA based MCUs, such as ARM Cortex-M series cores. The proposed core is specifically designed to give better performance in IoT applications. The significant contributions of this work can be summarised as follows: 1) Design of a four-stage pipelined micro-architecture that can execute the instructions efficiently and provide IPC as high as possible. 2) Integration of MAC (Multiply and Accumulate) instructions to enhance the performance of DSP algorithms. Modification of binary multiplier circuit to avoid any further hardware requirement due to the addition of MAC execution unit. 3) Enhancement of the performance of finite field multiplication so that it may be used to hardware security algorithms. 4) Modification of ALU to increase the performance of algorithms related to data processing.

The following sections will first explain the ISA and its modification in Section 5.2. The core micro-architecture is discussed in Section 5.3. The hardware resource requirement and benchmark results of the design are discussed in Section 5.4. Finally, Section 5.5 concludes the work.

5.2 ISA and its Extension

The starting point of a core design is the selection of ISA (Instruction Set Architecture). There are mainly two ISAs; ARM and x86, that are dominating the current commercial industry. But both of them are very complex and have proprietary issues. So, it is almost impossible for an individual researcher to implement core based on these ISAs. An open-source ISA can solve these issues and allows freedom to a researcher for designing cores and boosting performance in specific applications by adding user-defined instructions. RISC-V is one of such ISAs widely accepted due to its compatibility with direct native hardware implementation. The proposed core is designed to support the base integer instruction set, RV32I, along with the standard extension 'M' and a few new application-specific instructions. The new instructions are introduced to increase the data processing capabilities of the core and to improve the performance in data-centric applications such as IoT, DSP, hardware security, etc.

Figure 5.1 shows the newly added custom instructions. The first instruction, *mac*, performs 32×32 multiplication of *rs1* and *rs2*. The result of this multiplication is added to an accumulation register, and the lower 32-bits of the result are stored in the *rd* register. The next three instructions, *mach, machu, machsu* perform the same for signed×signed, unsigned×unsigned, signed×unsigned multiplication, and the upper 32-bits of the accumulation register are returned to the *rd* register. For initialising the accumulation register, *mul* instruction is used. A *mul* instruction performs 32×32 multiplication and initialises the accumulation register with the result. The result is also returned to the *rd* register. This eliminates the requirement of a separate multiplier unit, and all the multiplications are executed using the MAC unit.

31 25	24 20	19 15	14 12	11 7	6 0	
0000000	rs2	rs1	000	rd	0001011	mac
0000000	rs2	rs1	001	rd	0001011	mach
0000000	rs2	rs1	010	rd	0001011	machsu
0000000	rs2	rs1	011	rd	0001011	machu
0000001	rs2	rs1	000	rd	0001011	ex.ffm
0000001	rs2	rs1	001	rd	0001011	ex.add
0000001	rs2	rs1	011	rd	0001011	ex.xor
0000001	rs2	rs1	100	rd	0001011	ex.or
0000001	rs2	rs1	101	rd	0001011	ex.and
0000010	rs2	rs1	000	rd	0001011	ex.csr
0000010	rs2	rs1	001	rd	0001011	ex.csl

Figure 5.1: Extended custom instructions.

Similar to MAC, an accumulation register is also used in the ALU unit. This allows us to perform two logical/arithmetic operations in a single instruction, which is useful in many data-intensive applications. However, unlike the MAC unit, standard logical/arithmetic instructions are not used for initialising the ALU accumulation register, as these are very frequently used instructions and repeatedly resetting the ALU accumulation register is not a good programming practice. The accumulation register can be initialised to zero by executing *ex.and* instruction with one of its input registers value zero. The standard instruction set, RV32I, does not support the circular shift of the registers. It only supports logical and arithmetic shift of registers. Therefore, two instructions, *ex.csr* and *ex.csl*, are added to performs circular left/right shift of register values. *Ex.csr* performs circular shift right of *rs1*, and *ex.csl* performs circular shift left of *rs1*. These two instructions are encoded in I-type format. The operand to be shifted is in *rs1*, and the shift amount is encoded in the lower 5-bits of the I-immediate field. There is a special function, *ex.ffm*, introduced to the instruction set to enhance the finite field arithmetic performance. The *ex.ffm* uses characteristic 2 finite field with 256 elements, which is also called Galois Field GF(2^8). It performs modulo multiplication of *rs1* and *rs2* in Rijndaels Galois Field, considering individual bytes separately as shown in 5.1.

$$R = x[3] \cdot y[3] \oplus x[2] \cdot y[2] \oplus x[1] \cdot y[1] \oplus x[0] \cdot y[0] \tag{5.1}$$

Where *x[i]* and *y[i]* are individual bytes of *rs1* and *rs2*. With the addition of these extra instructions, the computational capability and throughput of the proposed core is increased by a significant amount. The details of the performance improvement is discussed in section 5.4.

5.3 Micro-architecture

In this section, we will discuss the microarchitectural optimisations of the proposed core aimed to increase the performance and efficiency of the core. The pipelining of the micro-architecture is described first, and then the individual components of the core.

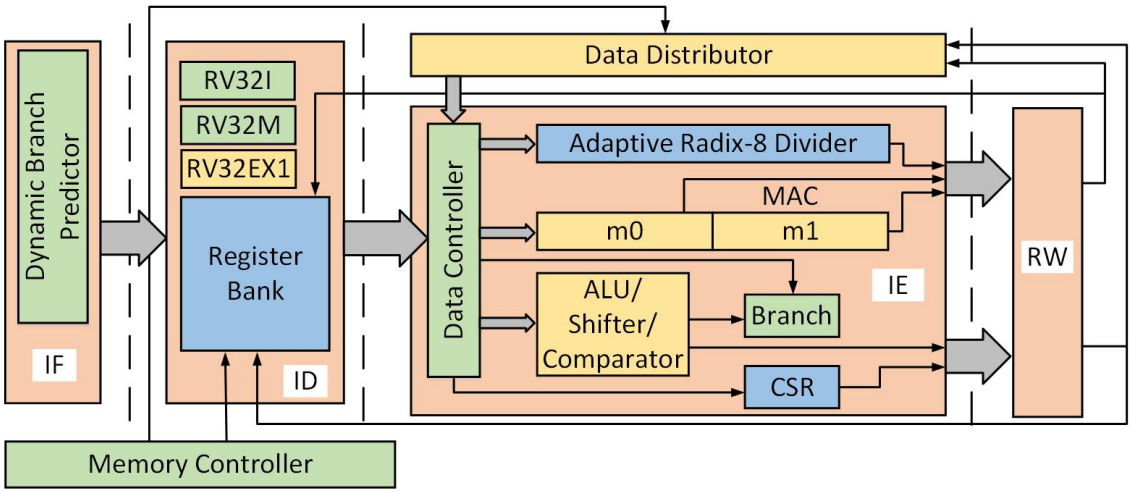


Figure 5.2: 4-stage pipeline of the proposed core

5.3.1 Pipeline

The most significant design aspect of processor implementation is the number of pipeline stages of the core. A shorter pipeline is good for simple design and achieving higher IPC (Instructions per Cycle). But, these designs require a longer clock cycle and usually operates at low clock frequency. The operating frequency can be increased by increasing the number of pipeline stages, but it also increases the probability of data and control hazards, which in turn reduces the IPC. In high-performance applications, the processor is designed with modern optimisation techniques, such as prefetch buffers and speculation. However, the implementation of these optimisation techniques increases the hardware resource of the design, which also increases the overall power consumption. Therefore, a processor with a short pipeline and higher IPC is preferred for low-power applications. The number of pipeline stages typically varies from one to five. For example, Cortex-M4 is a popular ARM core that has three pipeline stages, namely fetch, decode, execute. The micro-architecture has a single write-back port in the LSU (Load and Store Unit). The absence of a separate write-back port in the LSU and a fourth pipeline stage leads to stalls while executing load operations.

In the proposed core, the pipeline of the micro-architecture is organised in 4-stages. The basic goal is to keep the IPC as high as possible and reduce stalls as much as possible. These four stages are IF (instruction Fetch), ID (Instruction Decode), IE (Instruction Execute) and RW (Register Write). Compared to the 5-stage pipeline, the proposed 4-stage pipeline does not have the MEM stage to handle the delay of memory read/write instructions. This may lead to data/control hazards. To eliminate these hazards, a separate write-port is used in RB (Register Bank). The separate write-port increases the size of the register bank, but it also eliminates MUXes and buffers from the pipeline.

5.3.2 ALU

To enhance the performance in applications like IoT that use a lot of sensor data, the ALU is custom made to perform addition, AND, OR, XOR of three numbers. The incorporation of an accumulation register enables a user to combine multiple arithmetic operations into a single instruction. Typically, an ALU is capable of performing the addition of two numbers, so we have to execute 9 instructions in order to add 10 numbers. Whereas, using the proposed ALU micro-architecture, the same operation can be completed in 6 instructions. This is

Table 5.1: Summation of 10 numbers at instruction level with and without the use of custom instructions.

Without extension	Wth extension
add reg1, d1, d2	ex.and 0, 0, 0
add reg1, d3, reg1	ex.add 0, d1, d2
add reg1, d4, reg1	ex.add 0, d3, d4
add reg1, d5, reg1	ex.add 0, d5, d6
add reg1, d6, reg1	ex.add 0, d7, d8
add reg1, d7, reg1	ex.add reg1, d9, d10
add reg1, d8, reg1	
add reg1, d9, reg1	
add reg1,d10, reg1	

illustrated in Table 5.1. The advantages are significant when the size of the input data set increases. For example, for a set of 1000 data entries, the summation can be performed in 501 clock cycles compared to the 999 clock cycles in a traditional two-input ALU.

The first two numbers that the proposed ALU design uses are register values, and the third number is the accumulation register. The accumulation register stores the previous results of ALU. The accumulation register can be updated only by executing *ex.add, ex.and, ex.or,* and *ex.xor* instructions. In Table 5.1, *ex.and* instruction is used at the beginning of the summation to reset the accumulation register. The same can also be achieved by adding another instruction that initialises the accumulation register with the addition of two input register values. This extra instruction would also increase the decoder complexity. Therefore, we decided not to increase the instructions set.

5.3.3 MAC

The multiplier is an essential part of any core micro-architecture. In our design, it occupies 0.013 µm^2 area, which is 21% of the total area (0.061 µm^2) at UMC 65 nm technology node. So, an efficient multiplier circuit can enhance the power, performance and area efficiency of the core. In the proposed core micro-architecture, the multiplier is combined with the MAC unit. MAC is an essential part of any DSP-based core micro-architecture, and it can be used to boost the performance of many signal-processing algorithms. If we had incorporated a separate MAC unit, it would have increased the size of our core by an additional 20-25% area. The proposed MAC integrated multiplier circuit is implemented using only 3086 cells. The resource requirement of the same unit without MAC support is 2862 cells.

Figure 5.3 shows the register level implementation of the MAC unit. It has a register named as *Data_Reg* that accumulates the multiplication result of the inputs. To reduce the critical path delay of the micro-architecture, the partial product addition is performed in three parts, as shown in Figure 5.3. The lower 32-bits of the result has a latency of one clock cycle, and the higher 32-bits has 2 clock cycle latency.

Similar to the multiplier, a divider is also an essential part of core micro-architecture. The RV32M RISC-V ISA extension has four division instructions. Depending on the behaviour of the other circuit components, a radix-8 divider circuit is implemented to execute these instructions. The divider unit has a latency of 2-12 clock cycles and a critical path delay of

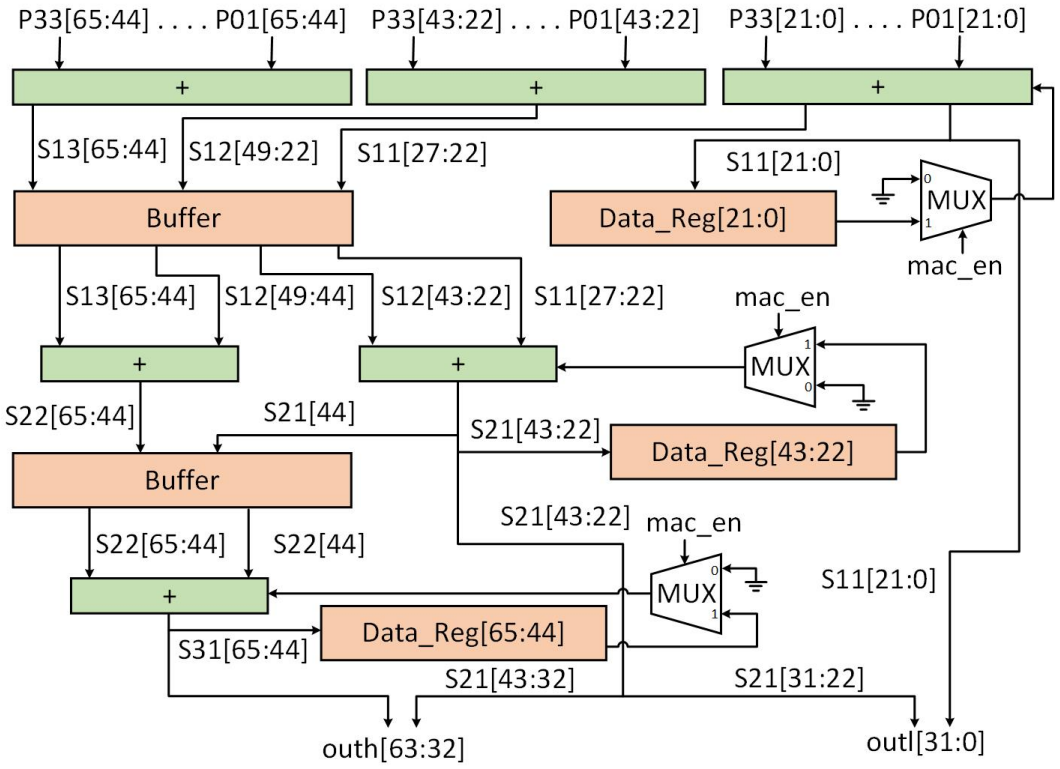


Figure 5.3: Block diagram of MAC unit

2.29 ns at UMC 65 nm technology node. A higher radix divider can provide better latency, but it also increases the delay of the circuit. So depending on the delay, radix-8 is found to be the most suitable one for the proposed micro-architecture. Figure 5.4 shows the block level representation of the divider unit. In the first step of division, the divisor is multiplied with integer values 0 to 7. These results are then compared to the dividend, and depending on the outcome of this comparison; subtraction and shift are performed on the dividend. This process continues for 12 cycles (maximum), and at the end of the 12th clock cycle, the circuit gives the required quotient and remainder.

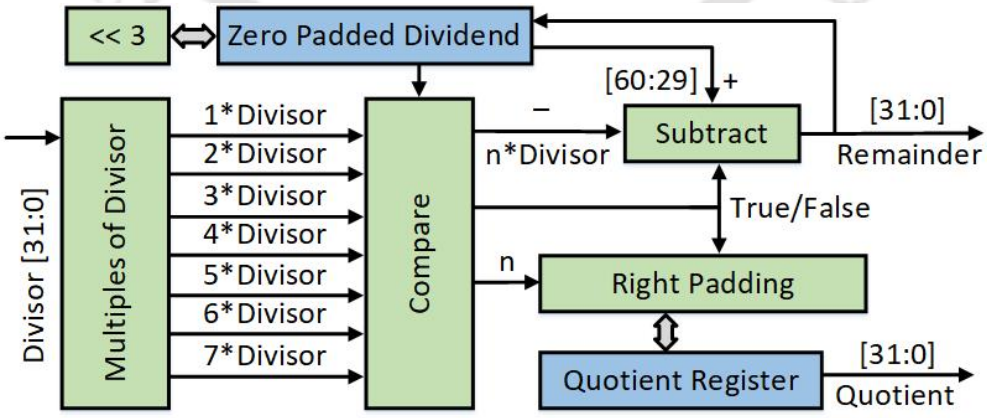


Figure 5.4: Block diagram of 32-bit radix-8 divider

5.4 Results and Discussion

The proposed design supports RISC-V base integer instruction set, RV32I, with M standard extension for integer multiplication and division. On top of these instructions, 11 new user-defined instructions have been added to the instruction set to enhance the performance in IoT applications. The proposed design is implemented using Verilog HDL and verified using Xilinx Vivado simulator. The design is synthesised in Synopsys DC at UMC 65 nm technology node for the estimation of area, power and delay. The layout of the design is implemented using Cadence Innovus, and post-layout estimation of power and delay is done in Synopsys PrimeTime. For performance measurement, the core is tested for applications such as cryptographic kernels, DSP, linear algebra, etc. The design is also made compatible with FPGAs and verified on Xilinx Virtex-7 FPGA board.

5.4.1 ASIC

The post-layout result of the proposed core is compared with some state-of-the-art cores in Table 5.2. At UMC 65 nm technology node, the core is implemented using 16.8 K cells and 1.6 K buffers, which results in a total area of 0.061 mm^2. The critical path delay of the proposed micro-architecture is 3.05 ns which means, the core is capable of working up to a maximum frequency of 328 MHz. While running at 100 MHz clock frequency, the core consumes 2.644 mW total power. About 99.6% of this power is dynamic power, and the leakage is only 10µW (0.4%). The dynamic power consumption is normalised to the clock frequency to make the comparison fair among the cores. The normalised value of dynamic power consumption of the proposed design is 26.44µW/MHz dynamic power. The proposed core outperforms many cores of Table 5.2 in area, maximum frequency, and power. Two benchmark programs, Dhrystone and CoreMark, are run on the proposed micro-architecture to see its performance in general purpose applications. The core performs 1.73 DMIPS per MHz and 4.04 COreMark/MHz, which is better than all other reported cores in Table 5.2.

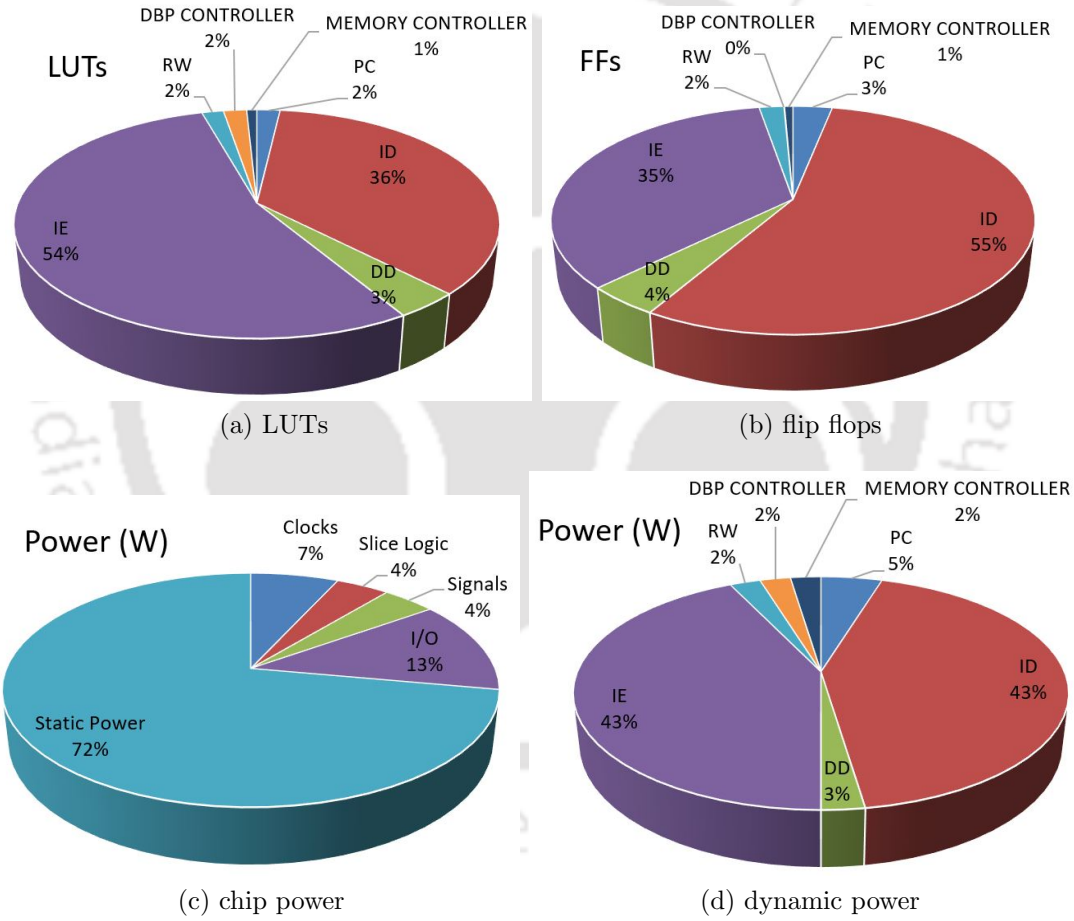
Table 5.2: ASIC post-layout result of the proposed IoT core at UMC 65 nm technology node.

Core	**Proposed**	**[11]**	**[49]**	**[23]**	**[50]**	**[51]**	**[52]**
ISA	RV32IM+Ex	RISC-V Ext	OpenRISC ISA	ARMT32	ARMT32	icyflex	RV32IMAFC
Technology Node	65 nm	65 nm	65 nm	90 nm	90 nmLP	180 nm	55 nm
Area (mm^2)	0.061	0.077	0.064	0.119	0.09	1.6	0.083
Delay (ns)	3.05	2.80	2.76	4.62	NA	20	4.76
Max Freq (MHz)	328	357	362	216	NA	50	210
Dynamic Power (µW/MHz)	26.4	26.68	33.8	32.8	31	120	67.8
DMIPS/MHz	1.73	NA	NA	1.52	1.50	NA	1.38
CoreMark/MHz	4.04	2.84	2.66	3.40	3.34	NA	3.10

5.4.2 FPGA

The proposed micro-architecture is primarily designed for ASIC, but fabrication and verification of a silicon chip requires a lot of time and cost. Therefore, the proposed design is implemented in such a way that it becomes compatible with FPGA hardware verification.

The core is tested on a Xilinx Virtex-7 FPGA board with an instruction cache and data cache of 64 KB each. These memories are implemented in the Block RAM of FPGA, and depending on the available resources, their size can be increased. The core has instruction start address at 0x00000200, and it is verified at a clock frequency of 100 MHz. The synthesis result shows that the core can be implemented with 7260LUTs and 2631FFs. The timing report shows that the design has a maximum delay path of 5.030 ns, which means it can achieve a maximum frequency of 198MHz on Virtex-7. For the functional verification of the design, the core is first verified with some basic C programs like addition, multiplication, factorial, loops, string, etc. Once the functionality of the core is verified, various algorithms related to IoT applications such as AES, SHA-1, DCT, convolution etc., are run on the proposed core to measure its performance. A popular benchmark program, Dhrystone, is also run on the core. All of these algorithms and benchmarks are compiled using a modified version of RISC-V GCC compiler to generate the binary of the programs. The binary data is uploaded to the memory of the core on FPGA to execute.



(a) LUTs (b) flip flops

(c) chip power (d) dynamic power

Figure 5.5: Distribution of FPGA power and hardware resources for the proposed IoT core.

5.4.3 Performance Measurement

In this section, a comparison of the performance is performed between basic RISC-V ISA (RV32IM) and the extension proposed (RV32IM+EX1) in this work. To evaluate the performance of the proposed design, various benchmark applications are executed on the core.

The set of benchmark applications includes DSP algorithms (convolution, FIR (finite impulse response), cryptographic algorithms (SHA-1 (Secure Hash Algorithm), AES (Advanced Encryption Standard)), and more data-intensive linear algebra (matrix multiplication). The compiler used to compile these C based benchmarks is derived from the original RISC-V GCC compiler, which is itself a derived version of MIPS GCC. The compiler is integrated with some user-defined libraries, which use the proposed extended instructions to execute complex algorithms.

Table 5.3: Performance improvement with the use of proposed ISA extension.

C-Program	RISC ISA	Extended ISA	% improvement
Convolution	8512	5343	37.2
FIR filter	7971	5398	32.2
DCT	63543	37803	40.5
AES	54265	19258	64.5
SHA-1	10790	10274	4.7
Matrix multiplication	2115	1178	44.3
Sum	30333	25324	16.5

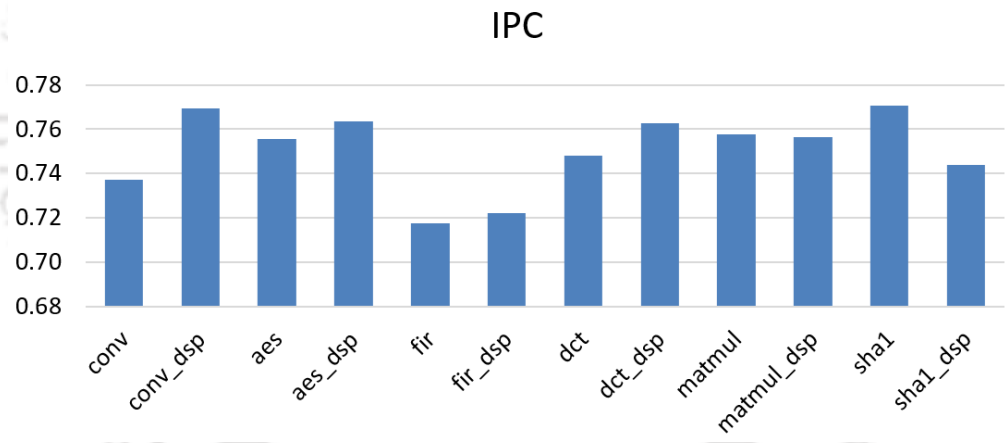


Figure 5.6: Instructions per cycle (IPC) while executing various algorithms.

Convolution is a common and widely used operation in signal processing. To measure the capabilities of the proposed core, convolution is performed on two inputs, having 10 samples each. The result shows that the core takes 8512 clock cycles to complete the convolution. However, with the use of the proposed extended instructions, the same convolution can be executed in 5343 clock cycles, which is around 37% better than the previous result. For a 10-tap FIR filter, the proposed core takes 5398 clock cycles. This is 32% better than the core performance without the proposed extension. The core is also tested for a 2D filter, DCT. The DCT of an 8x8 image requires 63543 clock cycles. However, the same operation can be executed in 37803 clock cycles using the proposed extension. A data-intensive linear algebra kernel, matrix multiplication also shows better performance. For the multiplication of two matrices of size 3x3 each, the proposed core takes 1178 clock cycles compared to 2115 clock cycles without the use of the proposed extension. This is around 1.8×performance improvement. The core is also tested for some security-related algorithms, such as AES,

SHA-1, etc. The core performance is improved by 64.5% in AES, whereas SHA-1 shows 4.7% improvement. Figure 5.6 shows the instructions per cycle (IPC) of the proposed core while executing the above-mentioned algorithms. The proposed core is capable of maintaining the IPC between 0.72 and 0.77.

In the proposed work, the performance improvement in the convolution, FIR filter, DCT is mainly due to the MAC unit. The MAC instructions multiply two register values and accumulate the result in an accumulation register. To explain the performance improvement in convolution, an example of 1D convolution is shown in Figure 5.7 and Figure 5.8. Figure 5.7 shows the c and assembly code without using the proposed extension, and Figure 5.8 shows the same using the MAC operations. The assembly code is shown for a small portion of the c code, which is indicated by a red box in the figures. From the example, it can be clearly seen that the same c operation requires 8 instructions in RISC-V ISA and 7 instructions in the proposed core ISA. Therefore, there is an improvement of one instruction, which further increases as the loop continues for a larger number of iterations. For a convolution of two signals having 10 samples each, the execution takes 8512 clock cycles in the first case and 5343 clock cycles in the proposed core with ISA extension. The experiment is performed only for 1D convolution. However, it will perform better 2D convolution also, because that also involves similar operations.

```
/* convolution operation */
    Begin_clk = rdcyc();
    for(i=0;i<m+n-1;i++) {
        y[i] = 0;
        for(j=0;j<=i;j++) {
            y[i]=y[i]+(x[j]*h[i-j]);
        }
    }
    End_clk = rdcyc();
    No_of_clk = End_clk - Begin_clk;
```

(a)

```
ac8:   0006a703          lw  a4,0(a3)
acc:   0007a803          lw  a6,0(a5)
ad0:   00078593          mv  a1,a5
ad4:   00468693          addi    a3,a3,4
ad8:   03070733          mul a4,a4,a6
adc:   ffc78793          addi    a5,a5,-4
ae0:   00e60633          add a2,a2,a4
ae4:   feb412e3          bne s0,a1,ac8 <main+0x1d8>
```

(b)

Figure 5.7: (a) C and (b) assembly code of convolution without using the proposed extension.

```
/* convolution operation */
    Begin_clk = rdcyc();
    for(i=0;i<m+n-1;i++) {
        asm volatile ("mul zero, %[b], %[c] ;\n": : [b] "r" (x[0]), [c] "r" (h[i]) );
        for(j=1;j<i;j++) {
            asm volatile ("mac zero, %[b], %[c] ;\n": : [b] "r" (x[j]), [c] "r" (h[i-j]) );
        }
        asm volatile ("mac %[a], %[b], %[c] ;\n": [a] "=r" (y[i]) : [b] "r" (x[j]), [c] "r" (h[i-j]) );
    }
    End_clk = rdcyc();
    No_of_clk = End_clk - Begin_clk;
```

(a)

```
b34:   ffc72683          lw  a3,-4(a4)
b38:   0007a603          lw  a2,0(a5)
b3c:   00c6800b          mac zero,a3,a2
b40:   ffc78793          addi    a5,a5,-4
b44:   00072683          lw  a3,0(a4)
b48:   00470713          addi    a4,a4,4
b4c:   fe8794e3          bne a5,s0,b34 <main+0x244>
```

(b)

Figure 5.8: (a) C and (b) assembly code of convolution using the proposed extension.

Similar to MAC unit, the accumulation register of ALU also helps in reducing the execution time applications. This register enhances the performance of the core whenever we have to use the ALU operations repeatedly on a large data set. For example, if we have to find the sum of 10000 numbers, then we have to run a loop of 5000 iterations, adding two numbers in each iteration. This addition requires 6 instructions of RISC-V ISA, as shown in Figure 5.9. However, the same operation requires 5 instructions in the proposed core, as shown in Figure 5.10. Overall, the complete summation of 10000 numbers requires 30333 clock cycles without using the proposed extension and 25324 clock cycles using the proposed ISA extension. This is around 16% improvement in performance.

```
int result=0,i;
for(i=0; i<=m-2; i=i+2) {
    result = result + x[i] + x[i+1] ;
}
return result;
```

(a)

```
5b0:   0007a603          lw  a2,0(a5)
5b4:   0047a683          lw  a3,4(a5)
5b8:   00878793          addi    a5,a5,8
5bc:   00c50533          add a0,a0,a2
5c0:   00d50533          add a0,a0,a3
5c4:   fef716e3          bne a4,a5,5b0 <sum_normal+0x48>
```

(b)

Figure 5.9: (a) C and (b) assembly code of sum without using the proposed extension.

```
int result,i;
asm volatile ("ex.and zero, zero, zero ;\n");
for(i=0; i<=m-4; i=i+2) {
    asm volatile ("ex.add zero, %[b], %[c] ;\n": : [b] "r" (x[i]), [c] "r" (x[i+1]) );
}
asm volatile ("ex.add %[a], %[b], %[c] ;\n": [a] "=r" (result) : [b] "r" (x[i]), [c] "r" (x[i+1]) );
return result;
```

(a)

```
510:   0007a703          lw  a4,0(a5)
514:   0047a683          lw  a3,4(a5)
518:   00878793          addi    a5,a5,8
51c:   02d7100b          ex.add  zero,a4,a3
520:   fef618e3          bne a2,a5,510 <sum_dsp+0x48>
```

(b)

Figure 5.10: (a) C and (b) assembly code of sum using the proposed extension.

5.5 Conclusion

This chapter presents a 4-stage pipelined high-performance 32-bit core. It is fully based on the RV32IM instruction set with custom extensions to enhance the performance in some real-world applications. The performance of the core is measured by computing matrix multiplication, FIR filter, DCT, etc. With the use of proposed custom instructions, the core performance can be improved by 1.59×in convolution, 1.68×in DCT, and 2.81×in AES encryption, respectively. For instruction accelerators, data can be moved at a relatively faster speed because it is tightly coupled with the core. On the other hand, independent accelerators are more autonomous. Therefore, it is necessary to combine the characteristics of the algorithm to select the appropriate coupling mode for a core to obtain optimum acceleration in performance.

The core results verify that the proposed core can achieve higher performance in many practical applications with a very small requirement of resources. Hence it can be considered as a suitable candidate for IoT endpoint devices.

CHAPTER

6

DESIGN AND IMPLEMENTATION OF RISC-V CORE FOR HIGH-SECURITY APPLICATION

Contents

6.1 Introduction 66
6.2 ISA and its Extension 66
6.3 Micro-architecture 67
6.3.1 Instruction Fetch 68
6.3.2 Instruction Decode 68
6.3.3 Instruction Execute 68
6.4 Results and Discussion 74
6.4.1 ASIC 75
6.4.2 FPGA 76
6.5 Comparison of Proposed IoT and High-security ISA Extensions 78
6.6 Conclusion 79

6.1 Introduction

The advancement in loT is not only making our life more convenient but also leading to some serious problems. Amongst them, the potential risk of sensitive data leakage is one of the most concerned problems. This is especially important in the case of critical infrastructure, which determines the life and health of people. The conventional solution is to utilise public-key cryptography (PKC) for secure key exchange along with symmetric-key cryptography (SKC) like AES for secure data transfer. Cryptography is one of the main approaches to secure information during communication.

IoT devices can be perceived as interconnected Embedded Systems, and their design usually includes light load processors. Due to the strict power and resources constraints, these processors usually offer very limited computing power, and they have relatively little ROM and RAM memory. Therefore, the cryptographic algorithms implemented on embedded systems must be efficient (use minimum resources) and be resistant to a wide range of attacks, including side-channel attacks. The main challenge of the implementation of cryptographic algorithms for embedded systems is the fulfilment of seemingly conflicting requirements of the level of security, performance, and price of the final device.

Almost all cryptographic algorithms have been designed to be implemented in computer systems that use universal processors providing sufficient CPU power. Therefore, the software implementation of encryption or hashing on 8/16/32-bit MCUs is slow and energy-intensive and requires quite a lot of memory. The search for new and adaptation of existing algorithms on platforms with limited computing power is covered by lightweight cryptography. Another approach to solving the problem of the complexity of encryption algorithms assumes the use of cryptographic accelerators to speed up calculations hundreds of times. The importance of security and the limitations in existing solutions motivate a new architecture in favour of the requirements for IoT devices. In this chapter, we propose a novel design of a cryptographic processor.

6.2 ISA and its Extension

The starting point of a core design is the selection of ISA (Instruction Set Architecture). As mentioned earlier, there are mainly two ISAs, ARM and x86, that are dominating the current commercial industry. But both of them are very complex and have proprietary issues. So, it is almost impossible for an individual researcher to implement core based on these ISAs. An open-source ISA can solve these issues and allows freedom to a researcher for designing cores and boosting performance in specific applications by adding user-defined instructions. RISC-V is one of such ISAs widely accepted due to its compatibility with direct native hardware implementation. The proposed core is designed to support the base integer instruction set, RV32I, along with the standard extension 'M' and a few new application-specific instructions. The new instructions include six instructions for AES 128-bit encryption and decryption. Two of these instructions are for modifying the key, and the rest are for AES encryption/decryption.

Figure 6.1 shows the newly added custom instructions. The instruction, AES.KEY can update the KEYREG of the AES hardware encryption unit. The KEYREG is a 128-bit register that contains the key of AES encryption. The instruction shifts the lower 64-bits of the KEYREG to the higher 64-bits and loads the lower 64 bits with *rs2* and *rs1*. IAES.KEY does the same for AES decryption unit.

31 25	24 20	19 15	14 12	11 7	6 0	
0000011	rs2	rs1	000	rd	0001011	AES.KEY
0000011	rs2	rs1	001	rd	0001011	AES.PT
0000011	rs2	rs1	010	rd	0001011	AES.CT
0000011	rs2	rs1	011	rd	0001011	IAES.KEY
0000011	rs2	rs1	100	rd	0001011	IAES.CT
0000011	rs2	rs1	101	rd	0001011	IAES.PT

Figure 6.1: Extended custom instructions for AES encryption and decryption.

AES.PT provides the plaintext to the AES encryption unit. If *rd* is *zero*, *rs1* and *rs2* provides the lower 64-bits of the plaintext but do not start the encryption process. A non *zero* *rd* starts the encryption process with the upper 64-bits of plaintext in *rs2* and *rs1*. AES.CT writes 32-bits of the ciphertext in *rd* register starting from the lower side. Therefore, it needs to be executed four times to get the complete ciphertext. IAES.CT and IAES.PT do the same for AES decryption unit. IAES.CT provides the ciphertext and IAES.PT writes the decrypted plaintext in register bank.

6.3 Micro-architecture

The proposed core micro-architecture is designed in four pipeline stages: IF (instruction Fetch), ID (Instruction Decode), IE (Instruction Execute) and RW (Register Write). The micro-architecture is similar to the low-power core. However, the IE stage is incorporated with a dedicated AES encryption/decryption unit to enhance the performance in data encryption/decryption. Figure 6.2 shows the pipeline stages of the proposed core. The main focus of this work is on optimising the core power, area, delay, and performance. The design is optimised to achieve a maximum frequency close to 200 MHz. Pipeline stages are organised in such a way that minimum data, control, and structural hazards can occur. These points are discussed in more detail in the following part of this section.

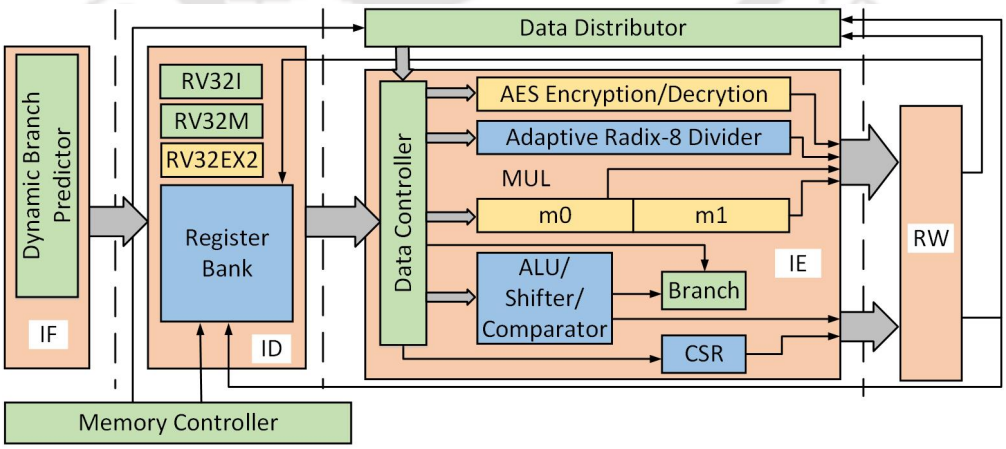


Figure 6.2: 4-stage pipeline of the proposed core.

6.3.1 Instruction Fetch

The IF stage generates a memory read request for instructions. It requests the next instruction as soon as it receives the previously requested instruction. The instruction fetching process takes one clock cycle, as every memory read process requires a minimum of one clock cycle. Therefore, the IF stage has two 32-bit registers, one to hold the address of the current instruction and the other to hold the address of the next instruction. This stage does not introduce any new delay to the circuit except the delay due to the memory read operation.

The micro-architecture includes a DBP that fetches instructions depending on the past result of branch instructions. The DBP maintains a history table with the help of program counter, branch address, and 2-bit prediction counter. The value of the counter decides the prediction. The four states of the 2-bit counter are assigned as strongly not taken, weakly not taken, weakly taken and strongly taken. For the first two states, the branch is predicted as false, and for the subsequent two states, the branch is predicted as true. If a wrong prediction occurs, the whole pipeline needs to be flushed, and correct instruction has to be fetched again. The number of bits for prediction counter can be one, two or more than two. But, every option has its own advantages and disadvantages [39]. A 1-bit counter is too simple, and prediction accuracy is not very good. A 2-bit predictor works well when branches predominantly go in one direction. A second check is made to make sure that a short and temporary change of direction does not change the prediction away from the dominant direction. Increasing the counter size further does not affect much the accuracy of prediction [40], but increases the complexity of the design. So, a 2-bit saturating counter based DBP is used for the proposed design.

6.3.2 Instruction Decode

In this stage, the fetched instructions are decoded to generate different control signals and added for the IE stage. All the instructions have to go through this stage of operation. All control transfer (jump & branch) instructions complete their operation in IE stage itself except the unconditional jump instruction, JAL. JAL instruction does not involve any register read/write operation; hence it is separated from other control transfer instructions and is forced to complete its operation in ID stage. As a result, the flushing of IE stage is not required for JAL. This leads to overall performance improvement of the core.

6.3.3 Instruction Execute

In IE stage, register values are first checked. If they are ready to read, execution continues; otherwise, values are fetched from IE/RW stage, depending on previous instructions. These register values are then fed into different Functional Units (FU) of IE stage for execution. These FUs execute the register values, and the execution output is written back to the register bank. The IE stage has six FUs. These FUs with their operations and clock cycles are mentioned below:

1. ALU - add, sub, and, or, xor (1 clock)
2. Comparator - signed/unsigned comparison (1 clock)
3. Shifter - logical/arithmetic shift (1 clock)
4. Multiplier - signed/unsigned multiplication (1-2 clocks)

5. Divider - signed/unsigned division (2-12 clocks)

6. AES - encryption/decryption (40 clocks)

ALU can perform five operations on two 32-bit inputs, which are the values from registers or program counter or zero-padded immediate. Comparator performs six comparisons, namely equal, not equal, signed/unsigned lesser than and signed/unsigned greater than on 32-bit inputs. Comparator unit gives an output of 1-bit, which is further zero-padded to form a 32-bit value. Shifter performs logical left/right and arithmetic right shift operation on 32-bit values. Depending on the type of instruction, the output of one of these three functional units is sent as register write signal.

IE stage also generates the memory read/write signals and branch signals. Memory read/write address is obtained from the ALU unit, and memory read/write data is obtained from the second source register, *rs2*. Memory read/write enable signal comes from the ID stage. These three signals are then combined to control the memory read/write operations. The branch consists of two signals; branch enable and branch address. Branch address is assigned from ALU output, and comparator output triggers the branch enable signal depending on the type of instruction.

Multiplier and Divider

Multiplier and Divider circuits are vital parts of any core micro-architecture, and their hardware complexity is very high compared to other functional units of IE stage. So, the hardware circuits of multiplication and division have a vital role in the overall area, power, and performance of the core.

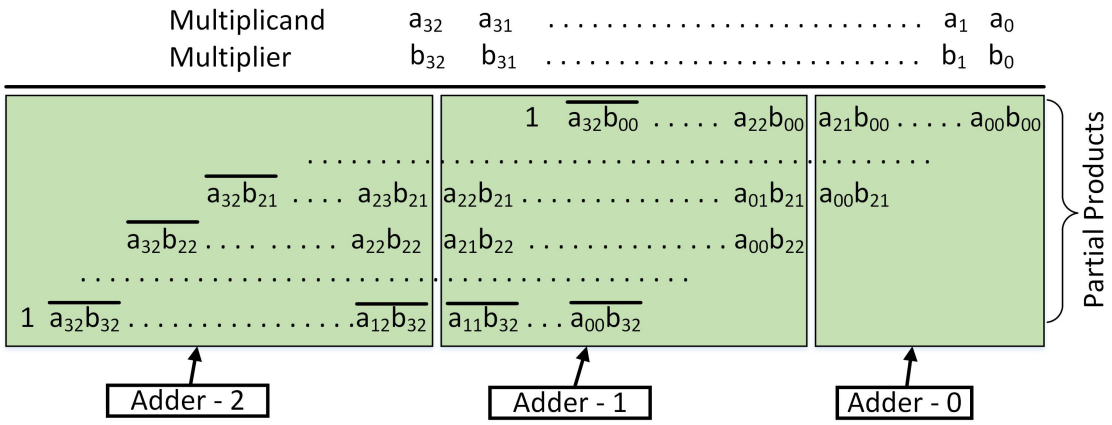


Figure 6.3: Addition of partial products of 33×33 signed multiplier.

In the core architecture, the requirement is to multiply two 32-bit signed/unsigned numbers. A 32-bit signed multiplier is capable of multiplying two 32-bit signed numbers but can only multiply up to 31-bit unsigned numbers as MSB is the sign bit. A 33-bit signed multiplier instead of a 32-bit signed multiplier is used to solve this issue. In the proposed multiplier circuit, the 32-bit signed/unsigned inputs are converted to 33-bit signed numbers. This 33-bit signed conversion is implemented in hardware by padding MSB on the left if the 32-bit input is signed and '0' otherwise. The 33×33 signed multiplier has 33 partial products as shown in Figure 6.3. These partial products are grouped in three parts as shown in Figure 6.3. These sets of partial products are added and buffered to produce the 64-bit

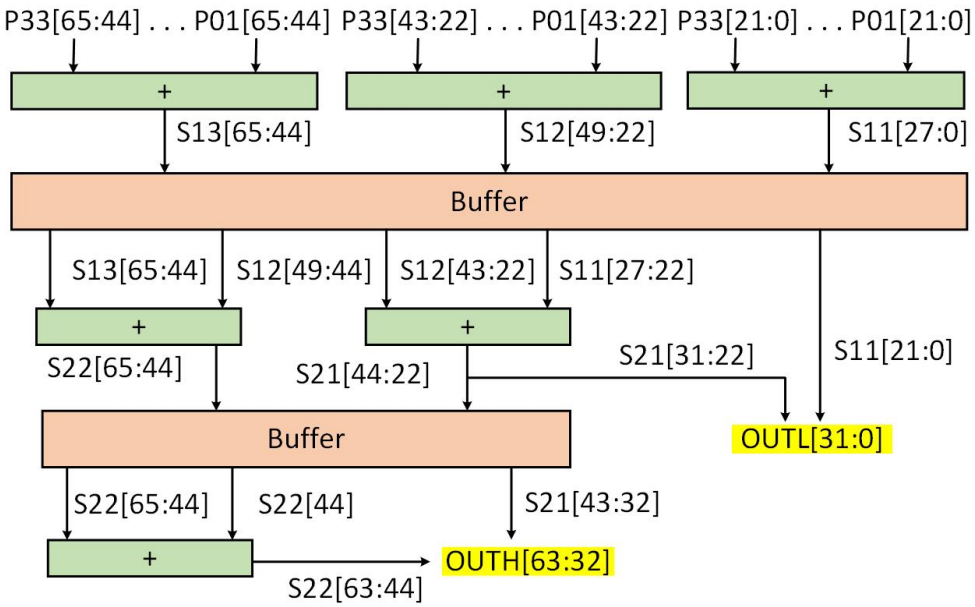


Figure 6.4: Block diagram of multiplier unit.

multiplication output as shown in Figure 6.4. The multiplier has single clock cycle latency for the lower 32-bits of the result and two clock cycle latency for the upper 32-bits of the result.

Similar to the multiplier, a divider is also an essential part of core micro-architecture. The RV32M RISC-V ISA extension has four division instructions. Depending on the behaviour of the other circuit components, a radix-8 divider circuit is implemented to execute these instructions. The divider unit has a latency of 2-12 clock cycles and a critical path delay of 2.29ns at UMC 65 nm technology node. A higher radix divider can provide better latency, but it also increases the delay of the circuit. So, radix-8 is found to be the most suitable one to keep the delay within the limit for the proposed micro-architecture. Figure 6.5 shows the block level representation of the radix-8 divider unit. In the division process, the divisor is first multiplied with integer values 0 to 7, and these results are compared to the dividend. Depending on the outcome of this comparison, subtraction and shift are performed on the dividend. This process continues for 2 (minimum) - 12 (maximum) clock cycles to generate the required quotient and remainder.

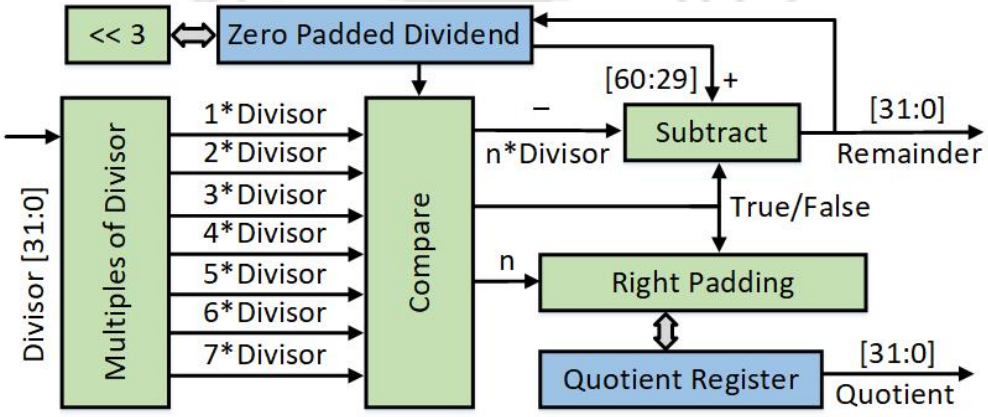


Figure 6.5: Block diagram of 32-bit radix-8 divider.

AES (128-bit key)

AES stands for Advanced Encryption System. It is an encryption algorithm used in IT applications to secure sensitive materials. AES was selected in 2001 as an official government security standard, but over time it also became the de facto encryption standard for the private sector. The AES security standard can be applied to restrict access to both hardware and software. AES encrypts data with a fixed block size of 128-bit and key sizes of 128, 192, or 256 bits. The key size used for an AES cypher specifies the number of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. In this work, the key size used is 128-bit which is good enough for a small processor. In fact, a 'brute force' attack would take billion years to crack AES 128-bit (AES) encryption even using a supercomputer.

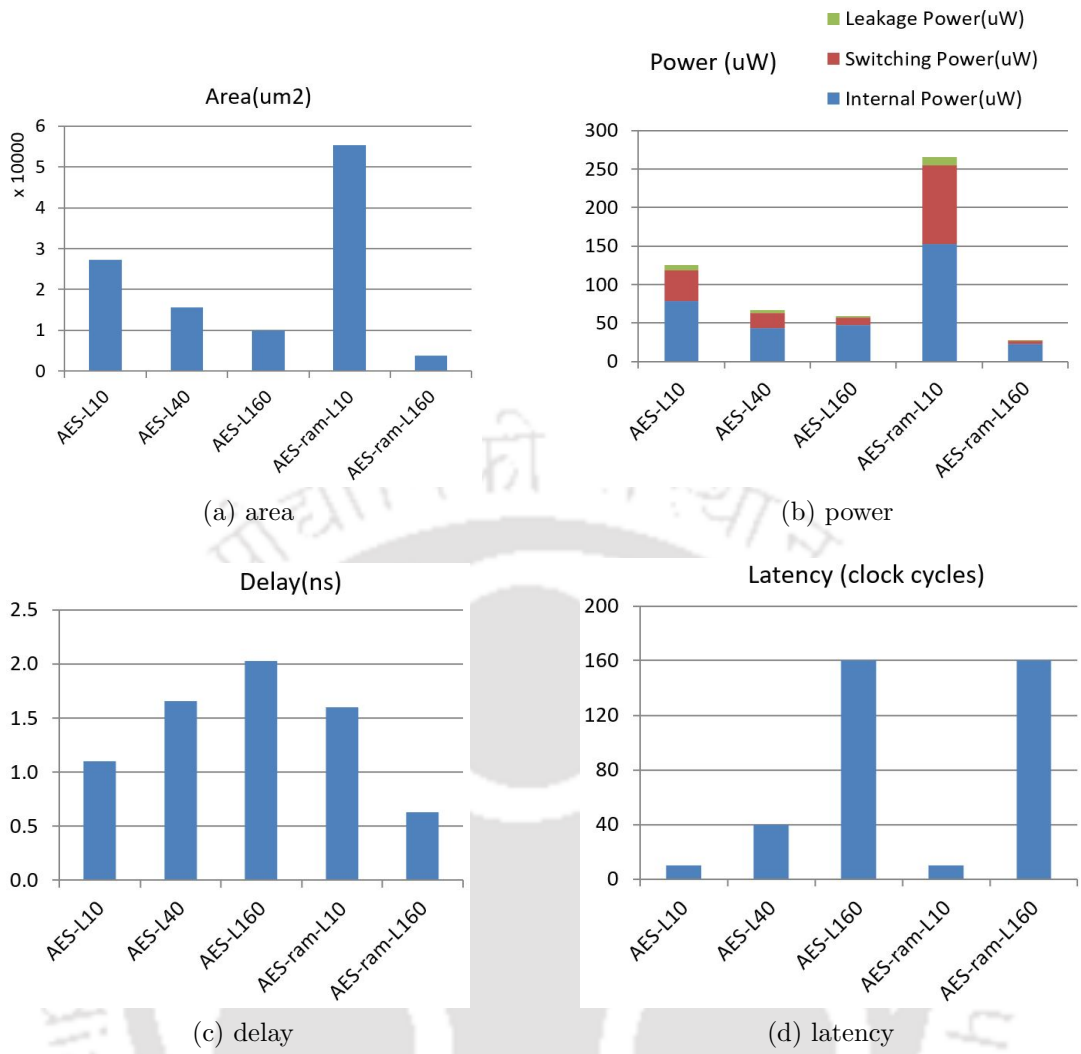
The AES encryption takes 10 rounds to produce ciphertext for the plaintext. It has also an initial stage of *AddRoundKey*. In the next 9 rounds, the data goes through *SubBytes*, *ShiftRows*, *MixColumns* and *AddRoundKey* in every round. In the final round, it performs *SubBytes*, *ShiftRows* and *AddRoundKey* to generate the ciphertext. In this work, we have designed five AES hardware encryption units based on the input data width and the type of s-box implementation.

- AES-L10 : 128-bit input/output data width, s-box using gate
- AES-L40 : 32-bit input/output data width, s-box using gate
- AES-L160 : 8-bit input/output data width, s-box using gate
- AES-ram-L10 : 128-bit input/output data width, s-box using memory
- AES-ram-L160 : 8-bit input/output data width, s-box using memory

For the first three designs, AES-L10, AES-L40, AES-L160, the s-box is implemented using gates. The input/output data widths for these designs are 128-bit, 32-bit and 8-bit, respectively. In the case of AES-L10, the processing takes a single clock cycle for each round of AES encryption. Therefore, the total latency of the design is ten clock cycles for single 128-bit encryption. In the case of AES-L40, every round is divided into four sub-rounds and therefore making the latency of the circuit 40 clock cycles. However, the area and power of the design are improved due to the reuse of hardware resources of *SubBytes*, *MixColumns* and *AddRoundKey*. Similarly, AES-L160 has a data width of 8-bits, and the latency of the

Table 6.1: Synthesis results of AES hardware implementations at UMC 65 nm technology node.

Design	**Area (mm^2)**	**Dynamic power (μW/MHz)**	**Delay (ns)**	**Latency (clock cycles)**	**Throughput (Gbps)**
AES-L10	0.0272	11.8	1.10	10	11.6
AES-L40	0.0155	6.3	1.66	40	1.92
AES-L160	0.0099	5.6	2.03	160	0.39
AES-ram-L10	0.0553	25.4	1.60	10	8
AES-ram-L160	0.0038	2.7	0.63	160	1.26



(a) area (b) power

(c) delay (d) latency

Figure 6.6: Area, power, delay, and latency of the AES encryption hardware designs.

circuit is found to be 160 clock cycles. In the last two designs, the s-box is implemented using memory, and the input/output data widths used are 128-bit and 8-bit. The circuits of these two designs have a latency of 10 and 160 clock cycles. Table 6.1 and Figure 6.6 show the ASIC synthesis results of the AES encryption units. Considering the results, the designs AES-L40 and AES-L10 are used in the implementation of proposed RISC-V cores. One design is targeted for high-throughput, and the other one keeps a balance between throughput and hardware resources.

Figure 6.7 shows the block diagram of the AES encryption unit, which is used in the proposed core micro-architecture. In the initial stage, bitwise XOR is performed between input plaintext and 128-bit key, and the result is stored in a register. For the next 9 rounds, the data goes through *ShiftRows*, *SubBytes*, *MixColumns*, and *AddRoundKey* in every round. In the *ShiftRows* stage, rows of data are shifted cyclically by a certain number of steps. *SubBytes* stage replaces each byte with a *SubBytes* using an 8-bit substitution-box (S-BOX). According to the original algorithm, *SubBytes* is performed prior to *ShiftRows*, but for the ease of hardware implementation, these two stages are interchanged. This interchange does not affect the result of the encryption as the stages are independent of each other. The *MixColumns* multiplies all four bytes of a column in Rijndael Galois field with a given matrix.

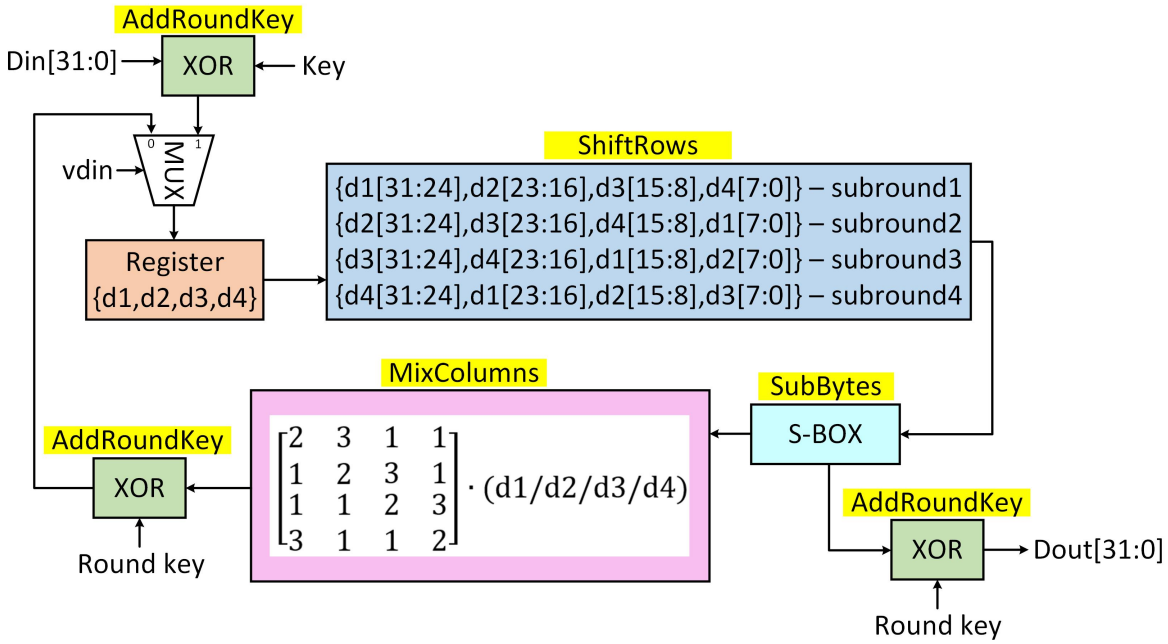


Figure 6.7: Block diagram of AES-L40 encryption unit.

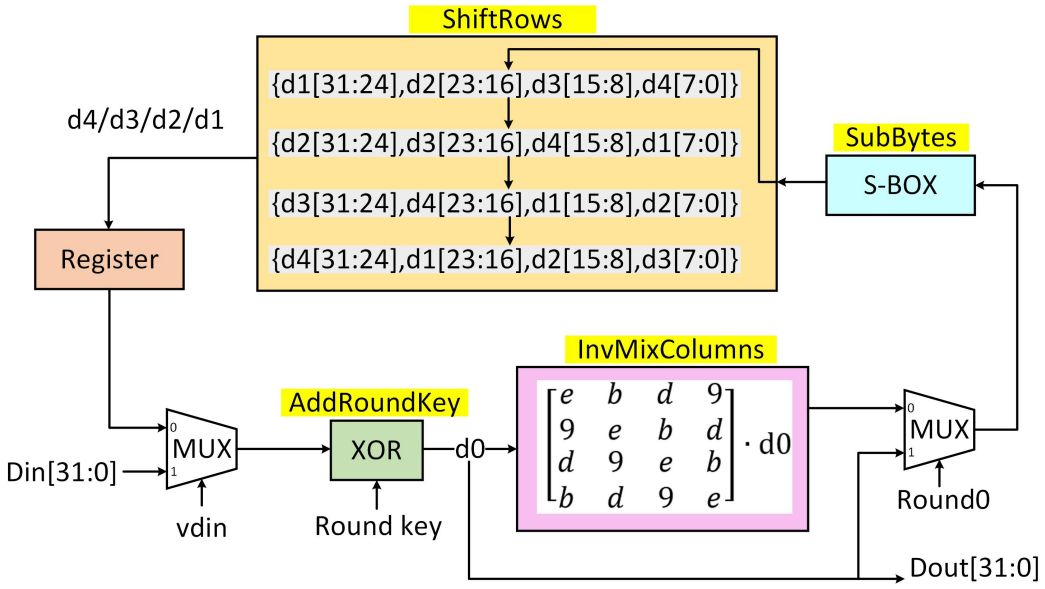


Figure 6.8: Block diagram of AES-L40 decryption unit.

In the last round, the data propagates through *ShiftRows*, *SubBytes*, and *AddRoundKey*, skipping the *MixColumns* stage. The circuit generates the output in the 10th round (40th clock cycle) from the *AddRoundKey* as shown in the block diagram. In the AES decryption unit, shown in Figure 6.8, the whole process executes in the reverse order. The *MixColumns* is replaced with a *InvMixColumns* where the columns are multiplied with a different matrix shown in Figure 6.8. Also, *ShiftRows* does the cyclic shift of row elements in the reverse direction. The AES decryption unit also takes 40 clock cycles for single 128-bit decryption.

In the hardware implementation of AES-L40 design, 32-bits of data are processed at every clock cycle. Initially, the 128-bit plain text is stored in a register, as shown in Figure 6.7. At

every clock cycle, four bytes of data is fetched from this register considering the ShiftRows, as shown in Figure 6.9. For SubBytes stage, each byte of data is processed using an S-Box module. Therefore, the S-Box is instantiated four times inside the SubBytes stage. The MixColumns multiplies all four bytes with a 4x4 matrix in the Rijndael Galois field. This multiplication is implemented using shifters and XOR gates, as shown in Figure 6.9.

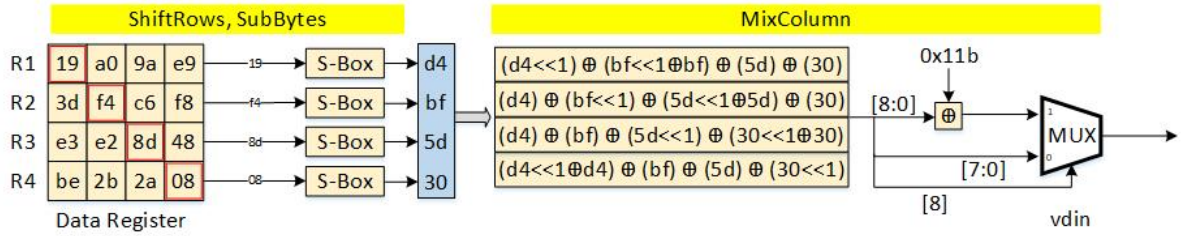


Figure 6.9: ShiftRows, SubBytes, and MixColumn stages.

Table 6.2 shows a distribution of hardware resources among different submodules of the proposed AES-L40 design. The table does not have information about the ShiftRows stage because the shifting of rows is considered while fetching data from the data register, as shown in Figure 6.9. So, in reality, ShiftRows does not require any hardware resource for implementation. Since we are processing 32-bits of data at every clock cycle, the controlling part of the design becomes complex, which is reported as others in Table 6.2.

Table 6.2: Area occupied by different submodules of the AES-L40 design.

FPGA				**ASIC (65nm)**		
Module		**LUT**	**FF**	**Module**		**Area (μm^2)**
AddRound		32	0	AddRound		115
SubBytes		4×40	0	SubBytes		4×1009
MixColumns		32	0	MixColumns		338
Others		98	15	Others		3532
Key Scheduler	ks_xor	32	0	Key Scheduler	ks_xor	195.8
	S-Box	4×40	0		S-Box	4×1009
	S-Box	4×40	0		S-Box	4×1009
Total		**668**	**143**	**Total**		**15906**

6.4 Results and Discussion

In this work, we have proposed two core micro-architectures based on AES encryption/decryption unit. The first core is targeted to high throughput, and AES-L10 is used as encryption/decryption unit. The second core is integrated with the AES-L40 encryption/decryption unit. The second core requires lesser hardware resources compared to the first core at the cost of throughput. Both the cores support base integer instruction set, RV32I of RISC-V ISA with multiplier and divider extension, M. The cores support six extended instructions for AES encryption/decryption. The designs are implemented using

Verilog HDL. The functionality of the cores is verified with various C-programs like addition, subtraction, factorial, loop, string, etc. These codes are compiled using the RISC-V GCC compiler, and Xilinx Vivado 2016.4 is used for the simulation of the Verilog code. The designs are also verified on Xilinx Virtex-7 FPGA platform, and the area, power, and delay of the designs on Virtex-7 are reported in the following part of this section. The proposed core micro-architectures are synthesised using Synopsys DC at 65 nm technology node for the estimation of area, power, and delay on silicon. The place and route of the synthesised designs are done in Cadence Innovus, and Synopsys PrimeTime tool is used for post-layout estimation of power and delay.

6.4.1 ASIC

At UMC 65 nm technology node, the first proposed core is implemented using 41 K cells and 4.1 K buffers, which results in a total area of 0.120 mm^2. The critical path delay of the proposed micro-architecture is 2.84 ns which means, the core is capable of working up to a maximum frequency of 352 MHz. The dynamic power requirement of the proposed design is 65.94 µW/MHz. The second core is implemented using 27.9 K cells and 2.7 K buffers, which results in a total area of 0.093 mm^2. The critical path delay of the proposed micro-architecture is 2.81 ns which means, the core is capable of working up to a maximum frequency of 356 MHz. While running at 100 MHz clock frequency, the core consumes 4.733 mW total power. About 99.6% of this power is dynamic power, and the leakage is only 17.8 µW (0.4%). The normalised value of dynamic power consumption of the proposed design is 47.1 µW/MHz. These results are reported in Table 6.3 and compared with some commercial/RISC-V cores. It can be seen that the core results are better than many existing micro-architectures. Arm CryptoCell-312 [92] is a security platform which is integrated to SSE-200, SIE-200, Cortex-M23 and Cortex-M33. CryptoCell-312 can process 3.07 bits per clock cycle with a maximum clock frequency of 200 MHz (at TSMC CLN40ULP). This results in a maximum throughput of 0.61 Gbps. Our high-throughput proposed core (proposed 1) is capable of working up to 0.71 Gbps. This value is 16% higher than CryptoCell-312 even at 65 nm, which may further increase at lower technology nodes.

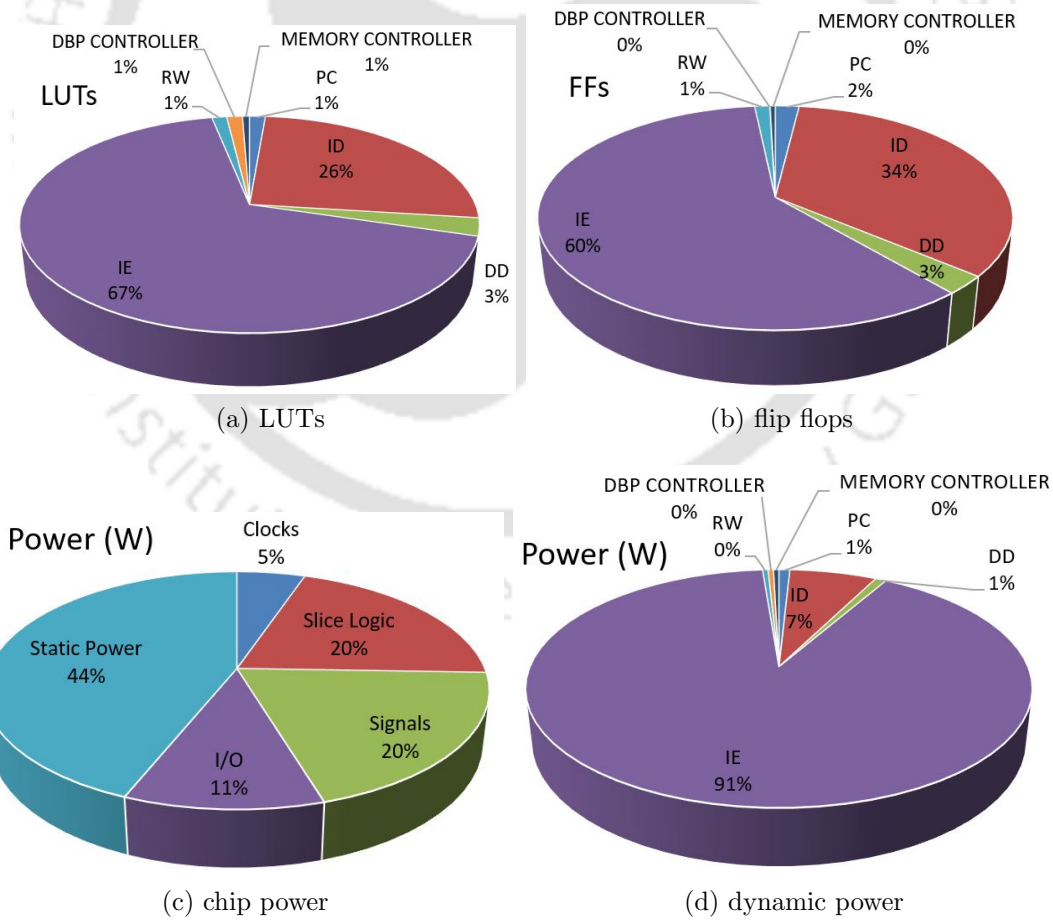
Table 6.3: ASIC post-layout result of the proposed high-security cores.

Core	Proposed 1 AES-L10	Proposed 2 AES-L40	PULPino [11]	Arm Coertex-M4 [23]	SiFive [52]	Arm CryptoCell-312 [92]
ISA	RV32IM+Ex2	RV32IM+Ex2	RISC-V Ext	ARMT32	RV32IMAFC	NA
Technology node	65 nm	65 nm	65 nm	90 nm	55 nm	TSMC CLN40ULP
Area (mm^2)	0.120	0.093	0.077	0.119	0.083	NA
Critical path delay (ns)	2.84	2.81	2.80	4.62	4.76	5
Max. freq (MHz)	352	356	357	216	210	200
Dynamic power (µW/MHz)	65.94	47.1	26.68	32.8	67.8	NA
DMIPS/MHz	1.73		NA	1.52	1.38	NA
CoreMark/MHz	4.04		2.84	3.40	3.10	NA
AES Throughput (Gbps)	0.71	0.49	NA	NA	NA	0.61

6.4.2 FPGA

The proposed micro-architectures are tested on a Xilinx Virtex-7 FPGA board with an instruction cache and data cache of 64 KB each. These memories are implemented in the Block RAM of FPGA, and depending on the availability of resources, their size can be increased/decreased. The cores have instructions starting address at 0x00000200, and they are verified at a clock frequency of 100 MHz. The synthesis results show that the first core (high-throughput) core has a resource requirement of 9335 LUTs, 3780 FFs and 10 RAMB18. The timing report shows that the design has a maximum delay path of 5.03ns, which means it can achieve a maximum frequency of 198.8MHz on Virtex-7. The design has a dynamic power requirement of 225mW while running at 100MHz clock. The second core can be implemented with 8345 LUTs and 3836 FFs. It can work up to a maximum frequency of 198.8 MHz with 5.03 ns critical path delay. The design has a dynamic power requirement of 158 mW while running at 100MHz clock. Figure 6.10 and Figure 6.11 show pie chart representation of the resource and power distribution of the proposed cores on XC7VX485T-2FFG1761C (Virtex-7) FPGA chip. Table 6.4 shows the FPGA synthesis results of the proposed cores.

The functionality of the proposed designs is verified on the FPGA hardware platform, Virtex-7. The cores are first verified with some basic C programs like addition, multiplication, factorial, loops, string, etc. Once the functionality of the core is verified, AES is



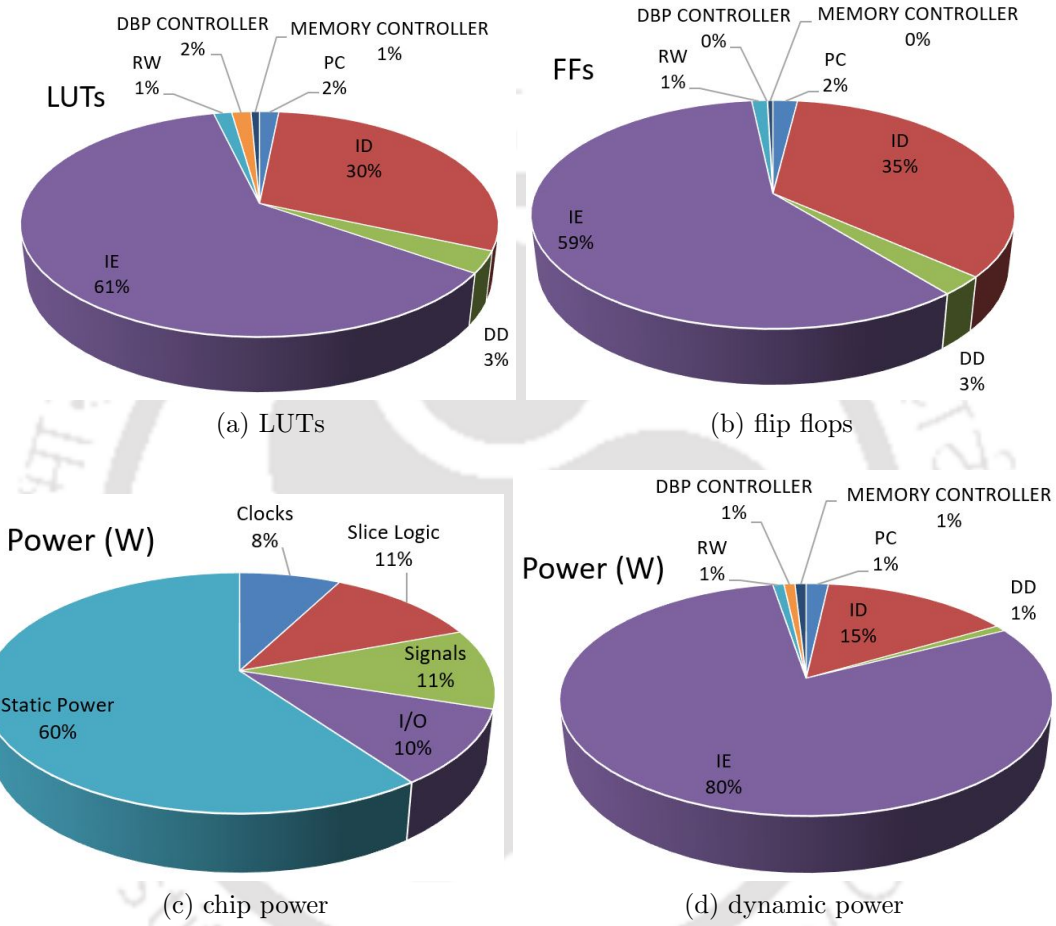
(a) LUTs (b) flip flops

(c) chip power (d) dynamic power

Figure 6.10: Distribution of FPGA power and hardware resources for the proposed core with AES-L10.

Table 6.4: FPGA synthesis results of the proposed high-security cores.

Core	LUTs	FFs	RAMB18	Delay (ns)	Max. freq (MHz)	Dynamic power (μW/MHz)	AES Throughput (Gbps)
Proposed 1 AES-L10	9335	3780	10	5.03	198.8	0.225	0.40
Proposed 2 AES-L40	8345	3836	4	5.03	198.8	0.158	0.27



(a) LUTs (b) flip flops

(c) chip power (d) dynamic power

Figure 6.11: Distribution of FPGA power and hardware resources for the proposed core with AES-L40.

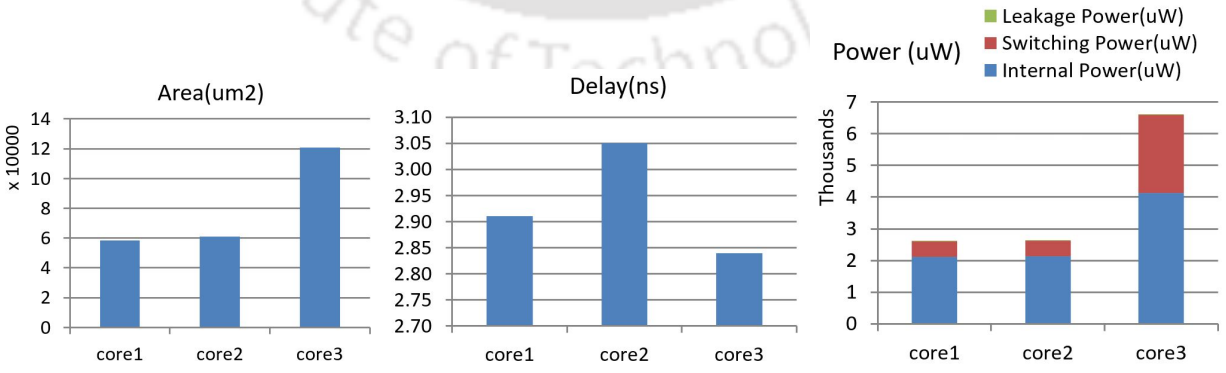
run on the core to measure its performance. The compiler used is a modified version of RISC-V GCC compiler, which uses the proposed instructions to enhance the performance of AES encryption/decryption. The compiled binary is uploaded to the cache memory of the processor on FPGA to execute. The proposed high-throughput core requires 63 clock cycles for AES encryption/decryption, which results in a throughput of 0.40 Gbps on FPGA. The proposed second core takes 93 clock cycles for single AES encryption and 97 clock cycles for single AES decryption. The throughput of the design is 0.27 Gbps during encryption, and the same for decryption is 0.26 Gbps.

6.5 Comparison of Proposed IoT and High-security ISA Extensions

In this section, a comparison is shown among three different core designs based on proposed ISA extensions. The first core, core1, supports only the standard instructions RV32IM of RISC-V. This core micro-architecture is different from the low-power core discussed in chpater 3. Major differences are the multiplier and the divider units. The multiplier and the divider designs used in the micro-architecture are the same as discussed in section 6.3 of this chapter. The second core, core2, supports RV32IM and the IoT extension (EX1) proposed in chapter 5. The third core, core2, supports RV32IM and the AES extension (EX2) proposed in the current chapter. Table 6.5 compares the ASIC implementation results (post-layout) of the three cores at UMC 65 nm technology node. Figure 6.12 shows a graphical comparison of area, delay and power. From the results, it can be seen that the AES integration is a very resource-intensive process. The AES functional unit occupies around 34% of the total area. However, the integration significantly boosts the AES encryption/decryption. The throughput of the AES integrated core, core3, is 0.49 Gbps, which is very high compared to the other two cores. Due to the integration of IoT extension (EX1), core2 also shows a significant improvement in AES encryption than core1, but not as high as core3. Table 6.6 shows the synthesis results of the three cores on Xilinx Virtex-7 FPGA, and Figure 6.13 shows the graphical representation of the same.

Table 6.5: ASIC post-layout result of core1 (RV32IM), core2 (RV32IM+EX1), and core3 (RV32IM+EX2) at UMC 65 nm technology node.

Core		core1	core2	core3
ISA		RV32IM	RV32IM+EX1	RV32IM+EX2
Area (mm^2)		0.058	0.061	0.093
Delay (ns)		2.91	3.05	2.81
Max. freq (MHz)		344	328	356
Dynamic power (µW/MHz)		26.0	26.3	47.1
AES encryption	Latency (clock cycles)	54265	19258	93
	Throughput (Gbps)	$8.1x10^{-4}$	$2.18x10^{-3}$	0.49

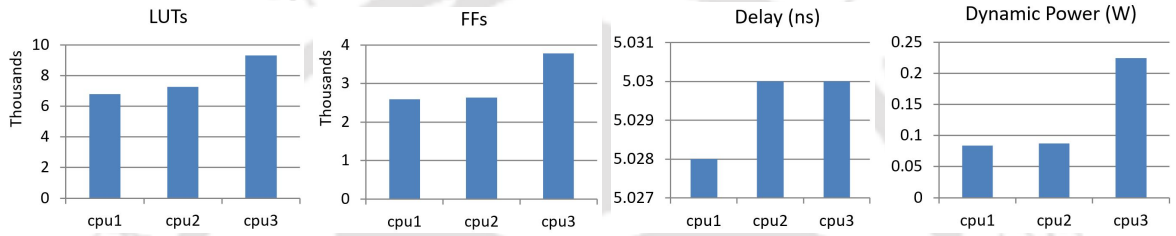


(a) area (b) delay (c) power

Figure 6.12: Comparison of area, delay, and power of core1 (RV32IM), core2 (RV32IM+EX1), and core3 (RV32IM+EX2).

Table 6.6: FPGA synthesis result of core1 (RV32IM), core2 (RV32IM+EX1), and core3 (RV32IM+EX2) for Xilinx Virtex-7.

Core		core1	core2	core3
ISA		RV32IM	RV32IM+EX1	RV32IM+EX2
LUTs		6783	7260	8345
FFs		2589	2631	3836
Delay (ns)		5.03	5.03	5.03
Max. freq (MHz)		198.8	198.8	198.8
Dynamic power (mW/MHz)		0.84	0.87	1.58
AES encryption	Latency (clock cycles)	54265	19258	93
	Throughput (Gbps)	4.68×10^{-4}	1.32×10^{-3}	0.273



(a) LUTs (b) FFs (c) Delay (d) Dynamic power

Figure 6.13: Comparison of area, delay, and power of core1 (RV32IM), core2 (RV32IM+EX1), and core3 (RV32IM+EX2) on Xilinx Virtex-7 FPGA.

6.6 Conclusion

This chapter presented implementations of a cryptographic processor by integrating AES encryption/decryption unit to a 32-bit RISC-V processor. The main goal of the proposed method was to achieve the maximum throughput, as well as to ensure a low requirement of area, power and delay. The results show that the proposed design is capable of gaining high throughput in AES encryption and decryption without using many hardware resources. From the comparison with existing designs, it is seen that the proposed design outperforms many of the existing architectures in terms of area, power, delay, and performance.

CHAPTER

7

CONCLUSION

Contents

7.1 Conclusion 81
7.2 Summary of Contributions 81
7.3 Future Directions 82

7.1 Conclusion

ISA and micro-architecture are two major aspects of processor design. In this work, we have developed different core micro-architectures based on these two aspects. The micro-architectures are designed to get maximum performance with a minimum requirement of area and power. The base ISA used for the designs is RISC-V (RV32IM). Two ISA extensions, EX1 and EX2, are also proposed for data-centric IoT and high-security applications. In this work, we have proposed three 32-bit core micro-architectures targeted to applications: low-power, data-centric IoT devices and high-security. The summary of contributions is given below.

7.2 Summary of Contributions

The low-power core is a four-stage pipelined micro-architecture. All the integer instructions (RV32I), except load and store, are executed in four stages. Memory instructions, load, and store are separated from the mainstream pipeline with an extra stage of memory read/write. Multiplication and division operations are executed in multiple cycles to reduce the critical path delay of the design. The design has a critical path delay of 5.05 ns which results in a maximum frequency of 198 MHz. The work also presents an analysis of eight different multiplier circuits. The analysis helped in selecting Baugh Wooley multiplier for the proposed core. Similarly, another analysis of seven divider circuits is also carried out, and radix-16 divider is chosen. The core is also incorporated with a dynamic branch prediction (DBP) unit with a 2-bit counter. The results show that the core outperforms many existing commercial and open-source cores.

The IoT core is based on RISC-V ISA (RV32IM) with the support of proposed ISA extension, EX1. The EX1 consists of eleven instructions those help in enhancing the core performance in DSP algorithms, various filters, cryptographic kernels, etc. The IoT core micro-architecture is also organised in four pipeline stages and provide an average IPC of 0.74. A MAC (Multiply and Accumulate) is incorporated to the core to enhance its performance in DSP applications. The MAC hardware unit is designed in such a way that the same unit is used for integer multiplication, and therefore reduces the requirement of any further hardware resource for a separate multiply unit. A finite field multiply unit is also incorporated to the core, which enhances the performance in hardware security algorithms. The ALU of the proposed IoT core has an accumulation register, which can store the result of ALU operation and use that in future operations. There is a 7% increment in the hardware resources of the core due to the incorporation of the proposed ISA extension. However, it helps in achieving an average of 1.74×performance improvement.

The high-security has a dedicated hardware unit for AES encryption/decryption. An ISA extension of six instructions, EX2, is proposed to use this AES hardware unit. The core can perform an AES encryption in 63 clock cycles, whereas the same encryption requires 19258 and 54265 clock cycles in the proposed IoT core and low-power core, respectively. The throughput of the core is 0.71 Gbps for AES encryption/decryption. At UMC 65nm technology node, the core can work up to a maximum frequency of 352 MHz. Table 7.1 presents some important results of the proposed low-power, IoT and high-security RISC-V cores.

This work also presents a design and implementation of a novel binary divider. The work includes an analysis of the proposed divider for different data widths (8, 16, 32, 64, 128) and

Table 7.1: Summary of the proposed low-power, IoT and high-security RISC-V cores.

Core	Low-power	IoT	High-security
ISA	RV32IM	RV32IM+EX1	RV32IM+EX2
Technology node	90nm	65nm	65nm
Area (mm^2)	0.173	0.061	0.093
Delay (ns)	5.39	3.05	2.81
Max. freq (MHz)	166.94	328	356
Dynamic power (μW/MHz)	19.75	26.3	47.1
DMIPS/MHz	1.71	1.73	1.73
CoreMark/MHz	4.13	4.04	4.04
AES Throughput (Gbps)	$4.1x10^{-4}$	$2.18x10^{-3}$	0.49

different radix values (4, 8, 16). The area, power, delay of the designs are estimated using Synopsys DC at 40 nm technology node. From the analysis, it has been observed that higher radix implementation of the proposed method is found more suitable for larger data widths. It reduces the latency by a significant amount with a small increment in delay.

7.3 Future Directions

In this work, we focused only on the core micro-architecture. However, memory management is also an important part of processor design. In future, one can integrate the proposed core micro-architectures into advanced memory management systems to get the most performance out of it. A multi-core system can also be a future direction to work upon. Modern multi-core systems are performing better than single-core high-performance systems. Multi-core systems provide flexibility to integrate different cores into a single chip. This allows the processor to perform better in multiple applications. This type of system development requires optimisation in hardware as well as in software.

RISC-V is a small and open-source instruction set. This allows individual researchers to develop ideas related to processor design. The ISA also allows novel extensions. In this work, we proposed two ISA extensions targeted to IoT and high-security applications. In future, researchers may come out with more ISA extensions depending on the demand of modern world devices. For example, ARM has Cortex-A, Cortex-R, Cortex-M, Ethos, Neoverse etc. series of processors depending on applications. Similarly, researchers may build RISC-V processors target to applications such as machine learning, Graphics processors, time-critical systems, high-performance computing etc.

We have integrated UART external peripheral to the proposed cores for communication from the external world. In future, VGA, HDMI, etc., may also be integrated for better display of program results. USB may also be integrated to connect various external world devices. In the proposed designs, we have used instruction and data cache of 64 KB each. These memories are good enough to fit small programs but not enough to fit larger programs. Integration of multilevel cache and DDR memory can solve this issue. This may also be another direction of work.

PUBLICATIONS

Published

1. **S. Bora** and R. Paily, "Design and Implementation of Adaptive Binary Divider for Fixed-point and Floating-point Numbers", *Circuits, Systems, and Signal Processing* (2021).

2. **S. Bora** and R. Paily, "A High-Performance Core Micro-Architecture Based on RISC-V ISA for Low Power Applications," in *IEEE Transactions on Circuits and Systems II*: Express Briefs, vol. 68, no. 6, pp. 2132-2136, June 2021, doi: 10.1109/TCSII.2020.3043204.

3. Talukder, S., Singh, R., **Bora, S.** et al., "An Efficient Architecture for QRS Detection in FPGA Using Integer Haar Wavelet Transform", *Circuits, Systems, and Signal Processing* 39, 36103625 (2020). https://doi.org/10.1007/s00034-019-01328-2.

4. V. C. Sekhar, **S. Bora**, M. Das, P. K. Manchi, S. Josephine and R. Paily, "Design and Implementation of Blind Assistance System Using Real Time Stereo Vision Algorithms," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), 2016*, pp. 421-426, doi: 10.1109/VLSID.2016.11.

Under Review

1. S. Bora and R. Paily, "Design and Implementation of a Four-Stage Pipelined Micro-Architecture based on RISC-V for IoT Devices," in *IEEE Internet of Things Journal.*

2. S. Bora and R. Paily, "A Four-Stage Pipelined Cryptographic Core Microarchitecture Based on RISC-V," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*

BIBLIOGRAPHY

[1] "Shakti repository." `https://bitbucket.org/casl/shakti_public`.

[2] "Pulpino." `https://github.com/pulp-platform/pulpino`.

[3] "Rocket core overview." `http://www.lowrisc.org/docs/tagged-memory-v0.1/rocket-core/`.

[4] "Dhrystone scores." `http://www.roylongbottom.org.uk/dhrystone%20results.htm`.

[5] "Coremark scores." `https://www.eembc.org/coremark/scores.php`.

[6] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0," Tech. Rep. UCB/EECS-2014-54, EECS Department, University of California, Berkeley, May 2014.

[7] B. Zimmer, Y. Lee, A. Puggelli, J. Kwak, R. Jevtic, B. Keller, S. Bailey, M. Blagojevic, P. F. Chiu, H. P. Le, P. H. Chen, N. Sutardja, R. Avizienis, A. Waterman, B. Richards, P. Flatresse, E. Alon, K. Asanovic, and B. Nikolic, "A RISC-V Vector Processor With Simultaneous-Switching Switched-Capacitor DC DC Converters in 28 nm FDSOI," *IEEE Journal of Solid-State Circuits*, vol. 51, pp. 930–942, April 2016.

[8] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanovic, and K. Asanovic, "A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators," in *ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC)*, pp. 199–202, Sep 2014.

[9] M. Gautschi, M. Schaffner, F. K. Gurkaynak, and L. Benini, "An Extended Shared Logarithmic Unit for Nonlinear Function Kernel Acceleration in a 65-nm CMOS Multicore Cluster," *IEEE Journal of Solid-State Circuits*, vol. 52, pp. 98–112, Jan 2017.

[10] F. Conti, R. Schilling, P. D. Schiavone, A. Pullini, D. Rossi, F. K. Grkaynak, M. Muehlberghuber, M. Gautschi, I. Loi, G. Haugou, S. Mangard, and L. Benini, "An IoT Endpoint System-on-Chip for Secure and Energy-Efficient Near-Sensor Analytics," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–14, 2017.

[11] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gurkaynak, and L. Benini, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, no. 99, pp. 1–14, 2017.

[12] S. Gupta, N. Gala, G. S. Madhusudan, and V. Kamakoti, "SHAKTI-F: A Fault Tolerant Microprocessor Architecture," in *2015 IEEE 24th Asian Test Symposium (ATS)*, pp. 163–168, Nov 2015.

[13] "Shakti processor program." `http://shakti.org.in/`.

[14] "Orca rv32im." `https://github.com/vectorblox/orca`.

[15] "mriscv." `https://github.com/onchipuis/mriscv`.

[16] "Vexriscv." `https://github.com/SpinalHDL/VexRiscv`.

[17] "Lowrisc." `https://github.com/lowRISC/lowrisc-chip`.

[18] "Ri5cy a small 4-stage risc-v vore." `https://github.com/pulp-platform/riscv`.

[19] A. Waterman, "Design of the RISC-V Instruction Set Architecture," in *Technical Report No. UCB/EECS-2016-1, EECS Department, University of California, Berkeley*, January 2016.

[20] A. Waterman, Y. Lee, R. Avizienis, H. Cook, D. Patterson, and K. Asanovic, "The RISC-V Instruction Set," in *Poster at the Symposium on High Performance Chips (HotChips-25)*, Stanford, CA, August 2013.

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

[22] D. Rossi, A. Pullini, M. Gautschi, I. Loi, F. K. Gurkaynak, P. Flatresse, and L. Benini, "A 1.8v to 0.9v body bias, 60 gops/w 4-core cluster in low-power 28nm utbb fd-soi technology," in *2015 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pp. 1–3, 2015.

[23] "The arm cortex-m4 processor, arms high performance embedded processor.." `https://developer.arm.com/products/processors/cortex-m/cortex-m4`.

[24] D. Rossi, I. Loi, A. Pullini, and B. L., *Ultra-Low-Power Digital Architectures for the Internet of Things*. Alioto M. (eds) Enabling the Internet of Things. Springer, Cham., 2017.

[25] N. Gala, A. Menon, R. Bodduna, G. S. Madhusudan, and V. Kamakoti, "SHAKTI Processors: An Open-Source Hardware Initiative," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, pp. 7–8, Jan 2016.

[26] J. Gray, "GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 17–20, May 2016.

[27] Y. Lee, A. Waterman, H. Cook, B. Zimmer, B. Keller, A. Puggelli, J. Kwak, R. Jevtic, S. Bailey, M. Blagojevic, P. F. Chiu, R. Avizienis, B. Richards, J. Bachrach, D. Patterson, E. Alon, B. Nikolic, and K. Asanovic, "An Agile Approach to Building RISC-V Microprocessors," *IEEE Micro*, vol. 36, pp. 8–20, Mar 2016.

[28] B. Zimmer, Y. Lee, A. Puggelli, J. Kwak, R. Jevtic, B. Keller, S. Bailey, M. Blagojevic, P. F. Chiu, H. P. Le, P. H. Chen, N. Sutardja, R. Avizienis, A. Waterman, B. Richards, P. Flatresse, E. Alon, K. Asanovic, and B. Nikolic, "A RISC-V vector processor with tightly-integrated switched-capacitor DC-DC converters in 28nm FDSOI," in *2015 Symposium on VLSI Circuits (VLSI Circuits)*, pp. C316–C317, June 2015.

[29] V. Patil, A. Raveendran, P. M. Sobha, A. D. Selvakumar, and D. Vivian, "Out of order floating point coprocessor for RISC V ISA," in *2015 19th International Symposium on VLSI Design and Test*, pp. 1–7, June 2015.

[30] A. Pullini, F. Conti, D. Rossi, I. Loi, M. Gautschi, and L. Benini, "A Heterogeneous Multi-Core System-on-Chip for Energy Efficient Brain Inspired Computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. PP, no. 99, pp. 1–1, 2017.

[31] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Grkaynak, A. Bartolini, P. Flatresse, and L. Benini, "A 60 gops/w, -1.8v to 0.9v body bias ulp cluster in 28nm utbb fd-soi technology," *Solid-State Electronics*, vol. 117, pp. 170–184, 2016. PLANAR FULLY-DEPLETED SOI TECHNOLOGY.

[32] Y. Lee, D. Sheffield, A. Waterman, M. Anderson, K. Keutzer, and K. Asanovic, "Measuring the gap between programmable and fixed-function accelerators: A case study on speech recognition," in *2013 IEEE Hot Chips 25 Symposium (HCS)*, pp. 1–2, Aug 2013.

[33] H. Vo, Y. Lee, A. Waterman, and K. Asanovic, "A Case for OS-Friendly Hardware Accelerators," in *7th Annual Workshop on the Interaction between Operating System and Computer Architecture (WIVOSCA-2013), at the 40th International Symposium on Computer Architecture (ISCA-2013)*, Tel Aviv, Israel, June 2013.

[34] "Comparing amba ahb to axi bus using system modeling." `https://www.design-reuse.com/articles/24123/amba-ahb-to-axi-bus-comparison.html`.

[35] "Pulpino." `http://www.pulp-platform.org/`.

[36] "Rocket chip." `https://github.com/freechipsproject/rocket-chip`.

[37] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avizienis, J. Wawrzynek, and K. Asanovic, "Chisel: Constructing hardware in a Scala embedded language," in *DAC Design Automation Conference 2012*, pp. 1212–1221, 6 2012.

[38] "riscv-rocket-chip-generator-tutorial-hpca2015." `https://riscv.org/wp-content/uploads/2015/02/riscv-rocket-chip-generator-tutorial-hpca2015.pdf`.

[39] H. Arora, S. Kotecha, and R. Samyal, "Dynamic branch prediction modeller for risc architecture," in *2013 International Conference on Machine Intelligence and Research Advancement*, pp. 397–401, 12 2013.

[40] Y. J. Ahn, D. Y. Hwang, Y. S. Lee, J.-Y. Choi, and G. Lee, "Saturating counter design for meta predictor in hybrid branch prediction," in *Proceedings of the 8th WSEAS International Conference on Circuits, Systems, Electronics, Control & Signal Processing*, CSECS'09, (Stevens Point, Wisconsin, USA), pp. 217–221, World Scientific and Engineering Academy and Society (WSEAS), 2009.

[41] N. Petra, D. D. Caro, V. Garofalo, E. Napoli, and A. G. M. Strollo, "Design of fixed-width multipliers with linear compensation function," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, pp. 947–960, May 2011.

[42] S. Kuang, J. Wang, and C. Guo, "Modified booth multipliers with a regular partial product array," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, pp. 404–408, May 2009.

[43] E. Antelo, P. Montuschi, and A. Nannarelli, "Improved 64-bit radix-16 booth multiplier based on partial product array height reduction," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, pp. 409–418, 2 2017.

[44] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Transactions on Computers*, vol. C-22, pp. 1045–1047, 12 1973.

[45] R. Muralidharan and C. Chang, "Radix-4 and radix-8 booth encoded multi-modulus multipliers," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, pp. 2940–2952, Nov 2013.

[46] Y. Bansal, C. Madhu, and P. Kaur, "High speed vedic multiplier designs-a review," in *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, pp. 1–6, 3 2014.

[47] L. Dadda, "On serial-input multipliers for two's complement numbers," *IEEE Transactions on Computers*, vol. 38, pp. 1341–1345, 10 1989.

[48] L. Chen, J. Han, W. Liu, P. Montuschi, and F. Lombardi, "Design, evaluation and application of approximate high-radix dividers," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, pp. 299–312, 7 2018.

[49] M. Gautschi, A. Traber, A. Pullini, L. Benini, M. Scandale, A. D. Federico, M. Beretta, and G. Agosta, "Tailoring instruction-set extensions for an ultra-low power tightly-coupled cluster of openrisc cores," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 25–30, Oct 2015.

[50] "The arm cortex-m3 processor, the industry-leading 32-bit processor for highly deterministic real-time applications." `https://developer.arm.com/products/processors/cortex-m/cortex-m3`.

[51] C. Arm, S. Gyger, J. Masgonty, M. Morgan, *et al.*, "Low-Power 32-bit Dual-MAC 120 μW/MHz 1.0 V icyflex1 DSP/MCU Core," *IEEE Journal of Solid-State Circuits*, vol. 44, pp. 2055–2064, 7 2009.

[52] "SiFive E24, a high-performance microcontroller for motor control, sensor fusion, and IoT applications." `https://www.sifive.com/cores/e24`.

[53] M. Pradhan and R. Panda, "High speed multiplier using nikhilam sutra algorithm of vedic mathematics," *International Journal of Electronics*, vol. 101, no. 3, pp. 300–307, 2014.

[54] K. Kataria and S. Patel, "Design of high performance digital divider," in *2020 IEEE VLSI DEVICE CIRCUIT AND SYSTEM (VLSI DCS)*, pp. 1–6, 2020.

[55] U. S. Patankar and A. Koel, "Review of basic classes of dividers based on division algorithm," *IEEE Access*, vol. 9, pp. 23035–23069, 2021.

[56] K. Jun and E. E. Swartzlander, "Improved non-restoring division algorithm with dual path calculation," in *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1379–1382, Aug 2013.

[57] N. Takagi, S. Kadowaki, and K. Takagi, "A hardware algorithm for integer division," in *17th IEEE Symposium on Computer Arithmetic (ARITH'05)*, pp. 140–146, June 2005.

[58] K. Jun and E. E. Swartzlander, "Modified non-restoring division algorithm with improved delay profile and error correction," in *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pp. 1460–1464, Nov 2012.

[59] N. Sorokin, "Implementation of high-speed fixed-point dividers on fpga," *Journal of Computer Science and Technology*, vol. 6, no. 1, pp. 8–11, 2006.

[60] G. Sutter and J. Deschamps, "High speed fixed point dividers for fpgas," in *2009 International Conference on Field Programmable Logic and Applications*, pp. 448–452, Aug 2009.

[61] D. M. Muñoz, D. F. Sanchez, C. H. Llanos, and M. Ayala-Rincón, "Tradeoff of fpga design of a floating-point library for arithmetic operators," *Journal Integrated Circuits and Systems*, vol. 5, no. 1, pp. 42–52, 2010.

[62] Z. T. Sworna, M. U. Haque, and S. Rahman, "An fpga-based divider circuit using simulated annealing algorithm," in *2018 18th International Symposium on Communications and Information Technologies (ISCIT)*, pp. 1–6, Sep 2018.

[63] D. W. Matula, M. T. Panu, and J. Y. Zhang, "Multiplicative division employing independent factors," *IEEE Transactions on Computers*, vol. 64, pp. 2012–2019, July 2015.

[64] H. Srinivas, K. Parhi, and L. Montalvo, "Radix 2 division with over-redundant quotient selection," *IEEE Transactions on Computers*, vol. 46, no. 1, pp. 85–92, 1997.

[65] L. Montalvo, K. Parhi, and A. Guyot, "New svoboda-tung division," *IEEE Transactions on Computers*, vol. 47, no. 9, pp. 1014–1020, 1998.

[66] M. D. Ercegovac and J. . Muller, "Variable radix real and complex digit-recurrence division," in *2005 IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05)*, pp. 316–321, July 2005.

[67] E. Antelo, T. Lang, P. Montuschi, and A. Nannarelli, "Digit-recurrence dividers with reduced logical depth," *IEEE Transactions on Computers*, vol. 54, pp. 837–851, 7 2005.

[68] J. Ebergen, I. Sutherland, and A. Chakraborty, "New division algorithms by digit recurrence," in *Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, 2004.*, vol. 2, pp. 1849–1855 Vol.2, Nov 2004.

[69] E. Antelo, T. Lang, P. Montuschi, and A. Nannarelli, "Low latency digit-recurrence reciprocal and square-root reciprocal algorithm and architecture," in *17th IEEE Symposium on Computer Arithmetic (ARITH'05)*, pp. 147–154, 6 2005.

[70] D. L. Harris, S. F. Oberman, and M. A. Horowitz, "Srt division architectures and implementations," in *Proceedings 13th IEEE Sympsoium on Computer Arithmetic*, pp. 18–25, July 1997.

[71] J. Verbeke and R. Cools, "The newtonraphson method," *International Journal of Mathematical Education in Science and Technology*, vol. 26, no. 2, pp. 177–193, 1995.

[72] R. Goldschmidt, *Applications of division by convergence*. Master's thesis, M.I.T., 1964.

[73] A. Parashar, G. Aggarwal, R. Dang, P. Dalmia, and N. Pandey, "Fast combinational architecture for a vedic divider," in *2017 14th IEEE India Council International Conference (INDICON)*, pp. 1–5, 2017.

[74] Y. Fu, M. Ikebe, T. Shimada, T. Asai, and M. Motomura, "Low latency divider using ensemble of moving average curves," in *2017 18th International Symposium on Quality Electronic Design (ISQED)*, pp. 397–402, 2017.

[75] N. Singh and T. N. Sasamal, "Design and synthesis of goldschmidt algorithm based floating point divider on fpga," in *2016 International Conference on Communication and Signal Processing (ICCSP)*, pp. 1286–1289, 2016.

[76] M. K. Jaiswal and H. K.-H. So, "Area-efficient architecture for dual-mode double precision floating point division," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 2, pp. 386–398, 2017.

[77] "IEEE standard for floating-point arithmetic," 2008.

[78] S. Obermann and M. Flynn, "Division algorithms and implementations," *IEEE Transactions on Computers*, vol. 46, no. 8, pp. 833–854, 1997.

[79] J. Melchert, S. Behroozi, J. Li, and Y. Kim, "SAADI-EC: A quality-configurable approximate divider for energy efficiency," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2680–2692, 2019.

[80] Y. Park, J. Kwon, and Y. Lee, "Area-efficient and high-speed binary divider architecture for bit-serial interfaces," in *2016 International SoC Design Conference (ISOCC)*, pp. 303–304, 2016.

[81] "Arm cortex-a5 processor performance and specification." `http://www.arm.com/products/processors/cortex-a/cortex-a5.php`.

[82] G. Lammel, "The future of mems sensors in our connected world," in *2015 28th IEEE International Conference on Micro Electro Mechanical Systems (MEMS)*, pp. 61–64, 2015.

[83] V. Shnayder, M. Hempstead, B.-r. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, (New York, NY, USA), p. 188200, Association for Computing Machinery, 2004.

[84] E. F. Nakamura, A. A. F. Loureiro, and A. C. Frery, "Information fusion for wireless sensor networks: Methods, models, and classifications," *ACM Comput. Surv.*, vol. 39, p. 9es, sep 2007.

[85] O. Azizi, A. Mahesri, B. C. Lee, S. J. Patel, and M. Horowitz, "Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis," *SIGARCH Comput. Archit. News*, vol. 38, p. 2636, 6 2010.

[86] M. Konijnenburg, S. Stanzione, L. Yan, D. W. Jee, J. Pettine, R. van Wegberg, H. Kim, C. van Liempd, R. Fish, J. Schluessler, H. de Groot, C. van Hoof, R. F. Yazicioglu, and N. van Helleputte, "28.4 a battery-powered efficient multi-sensor acquisition system with simultaneous ecg, bio-z, gsr, and ppg," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 480–481, 2016.

[87] P. G. Flikkema and B. Cambou, "Adapting processor architectures for the periphery of the iot nervous system," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 615–620, 2016.

[88] D. Li, H. Gong, and Y. Chang, "Implementing riscv system-on-chip for acceleration of convolution operation and activation function based on fpga," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 1–3, 2018.

[89] Y. Zhang, F. Zhang, Y. Shakhsheer, J. D. Silver, A. Klinefelter, M. Nagaraju, J. Boley, J. Pandey, A. Shrivastava, E. J. Carlson, A. Wood, B. H. Calhoun, and B. P. Otis, "A Batteryless 19 *mu*W MICS/ISM-Band Energy Harvesting Body Sensor Node SoC for ExG Applications," *IEEE Journal of Solid-State Circuits*, vol. 48, pp. 199–213, Jan 2013.

[90] S. R. Sridhara, M. DiRenzo, S. Lingam, S. J. Lee, R. Blazquez, J. Maxey, S. Ghanem, Y. H. Lee, R. Abdallah, P. Singh, and M. Goel, "Microwatt Embedded Processor Platform for Medical System-on-Chip Applications," *IEEE Journal of Solid-State Circuits*, vol. 46, pp. 721–730, April 2011.

[91] Y. Zhang, L. Xu, Q. Dong, J. Wang, D. Blaauw, and D. Sylvester, "Recryptor: A reconfigurable cryptographic cortex-m0 processor with in-memory and near-memory computing for iot security," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4, pp. 995–1005, 2018.

[92] "Arm cryptocell-312." `https://developer.arm.com/ip-products/security-ip/cryptocell-300-family/cryptocell-312`.