ENHANCING RISC-V ISA WITH POSIT ARITHMETIC AND ASSOCIATED HARDWARE INNOVATION

Thesis

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF TECHNOLOGY (RESEARCH)

in

VLSI DESIGN

by

ASHLEY KURIAN
(Reg. No. 203009)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,
SURATHKAL, MANGALORE - 575025

FEB 2023

DECLARATION

I hereby ***declare*** that the Research Thesis entitled **ENHANCING RISC-V ISA WITH POSIT ARITHMETIC AND ASSOCIATED HARDWARE INNOVATION** which is being submitted to the ***National Institute of Technology Karnataka, Surathkal*** in partial fulfillment of the requirement for the award of the Degree of ***Master of Technology (Research)*** in ***VLSI Design*** in ***Department of Electronics and Communication Engineering*** is a ***bonafide report of the research work carried out by me***. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

[illegible]

Place : NITK - Surathkal
Date : 06/02/2023

ASHLEY KURIAN
(Reg. No. 203009)
Department of Electronics &
Communication Engineering

CERTIFICATE

This is to certify that the Research Thesis entitled **ENHANCING RISC-V ISA WITH POSIT ARITHMETIC AND ASSOCIATED HARDWARE INNOVATION** submitted by **ASHLEY KURIAN** (Register Number: 203009) as the record of the research work carried out by her, is accepted as the Research Thesis submission in partial fulfillment of the requirements for the award of degree of **Master of Technology (Research)** in **VLSI Design**.

Dr. Ramesh Kini M.
Research Guide
Associate Professor
Department of Electronics &
Communication Engineering
NITK Surathkal - 575025

Chairman-DRPC
(Signature with Date and Seal)

Acknowledgment

I would like to convey my heartfelt thanks and regards to my research guide, Dr. Ramesh Kini M., Associate Professor, Department of Electronics and Communication Engineering, NITK, for his constant mentorship and counseling provided during the course of this research work. His immense knowledge and motivation has helped to bring this thesis in its present form. I could not have imagined having a better advisor for my research work.

Moreover, I would like to extend my sincere gratitude to the RPAC members Dr. Laxminidhi Tonse, Professor, Department of Electronics and Communication Engineering and Dr. Basavaraj Talawar, Assistant Professor, Dept. of Computer Science and Engineering for their valuable suggestions and constant support during the progress of the research.

Moreover, I am immensely grateful to the faculty and staff of Department of Electronics and Communication Engineering, NITK. I am thankful to Dr. Sumam David S., Professor, Department of Electronics and Communication Engineering for her efforts in familiarizing me with FPGA and the Vivado tool. To my friends and family, who stood by me and gave me unconditional love and support, I am beyond blessed to have you in my life.

Ashley Kurian

Abstract

The Posit extended RISC-V processor has gained tremendous attention as an alternative to its floating-point counterpart as it offers extensive customization in terms of dynamic range and precision. As a result, domain specific application developers have started to embrace it as a viable solution in emerging fields such as the big data analytics, machine learning and audio processing. However, the Posit compliant RISC-V processor needs further enhancements to augment the acceptability. In this thesis, the shortcomings of the existing Posit integration approaches are discussed and a novel approach is put forth. Unlike the current approaches of extending the RISC-V ISA with Posit, the thesis provides insights on how Posit can be incorporated along with the floating point arithmetic within the core. We also present a comparative study of various Posit integration approaches in terms of the resource utilization and timing requirements. From the analysis, it is found that the proposed approach performs better. The proposed RISC-V processor instruction set supports three data types, namely, the integer, floating-point and the Posit arithmetic. This processor also supports data type casting. This hardware unit helps in re-use or porting of existing code written using integer/ floating-point data types to be converted to Posit data type. Also it helps in using mixed data types in the code. Two different data type casting approaches such as the mixed operand type and the data type converter approach are suggested. Comparison of these approaches are carried out in terms of speed and the area occupied. Moreover, the Posit unit is modified to support two different exponent sizes, thereby enabling dynamic switching between higher precision and dynamic range at run-time with minimal overheads. The enhanced Posit compliant RISC-V processor is implemented on an Artix-7 Xilinx FPGA.
Keywords: arithmetic, data type casting, floating-point, IEEE-754, Posit, RISC-V.

Contents

List of Tables

List of Figures

List of Abbreviations

CSR	Control and Status Register (CSR)
DTC	Data Type Casting
DNN	Deep Neural Network
DZ	Division by Zero
es	Exponent Size
F	Floating point extension
FPU	Floating-Point Unit
HDL	Hardware Description Language
I	Integer extension
IR	Instruction Register
ISA	Instruction Set Architecture
M	integer Multiplication and division
MOT	Mixed Operand Type
N	Posit size
NaN	Not a Number
NaR	Not a Real
P	Posit extension

PPU	Posit Processing Unit
rd	Register Destination
rs	regime bits
rs1	Register Source 1
RoCC	Rocket Custom Co-processor
RV32IMF	RISC-V 32 bit IMF extensions
S	Sign bit
SoC	Silicon on Chip

CHAPTER 1

INTRODUCTION

Burgeoning scientific and technological domain applications demand varied precision and dynamic range arithmetic. Owing to the paucity of a suitable supplant, computer architects still restrict themselves to the currently used IEEE 754-2008 complaint floating-point arithmetic. Use of floating-point arithmetic can result in errors due to rounding, underflow and overflow. This has resulted in the proposal of Posit arithmetic as an alternative. It provides variable dynamic range, eliminates NaNs (not a number), mathematically incorrect multiple representation of zero and avoids underflow and overflow through rounding.

Moreover, the variable size of the exponent and fraction fields in Posit arithmetic enables customization. A m-bit Posit substantially reduces the overall resource utilization compared to n-bit floating-point arithmetic (where $m < n$) [3]. RISC-V, an open source Instruction Set Architecture (ISA) has been designed to support extensive customization and specialization. The base integer ISA (*RV32I*) can be further extended with standard, reserved and custom instructions. The custom extension feature in RISC-V ISA can be utilized to develop customized applications. Over the years, RISC-V has gained acceptance leading towards its development in terms of both hardware and software.

1.1 Motivation

The RISC-V ISA, today supports only two floating-point standard extensions namely F and D, which stands for single and double precision floating-point extensions respectively. The merits of novel Posit arithmetic over the IEEE-754 floating-point arithmetic gives ample motivation to incorporate a Posit compliant RISC-V ISA. A Posit enabled RISC-V core can be developed through two approaches:

1. Posit integration as an accelerator by utilizing the custom opcode space of the RISC-V ISA

2. Posit integration as a tightly coupled unit by replacing the F extension

The feasibility of these two approaches and the co-existence of the Posit and floating-point arithmetic needs to be investigated. Implementation of a Posit and IEEE-754 floating-point compliant RISC-V core will escalate the acceptance of the processor in a wide range of domains. The run-length encoding feature of Posit gives it an edge in comparison to the floating-point arithmetic especially in applications concerning a stringent requirement of higher dynamic range or precision. The outgrowth that the unit under consideration is likely to bring about in fields such as the machine learning, artificial intelligence, big-data analytics etc., motivates me to work on the same.

1.2 Objectives

The objectives of the research are as listed below,

1. Design and implementation of the Posit unit
2. Incorporation of the Posit unit within the RISC-V core, thereby facilitating the co-existence of the floating-point and Posit arithmetic
3. A comparative study of Posit integration within the core pipeline and as an accelerator in terms of resource utilization and timing requirements
4. Develop hardware to handle mixed operand types namely Posit, integer and floats
5. Extend the RISC-V instruction set to add instructions meant to handle data type casting
6. A comparative study of the data type casting approaches put forth
7. Develop reconfigurable Posit unit capable of switching between the maximum exponent size (es) at runtime

1.3 Thesis organization

This thesis explores the development of a Posit compliant RISC-V processor and its enhancement that can augment its acceptability in numerous applications. The thesis organization is as follows,
Chapter 2 provides a brief background about the Posit arithmetic and the RISC-V ISA. Chapter 3 deals with the literature survey and subsequent discussion on the research gaps identified. Chapter 4 elaborates on the research work that includes the development of Posit arithmetic unit, the discussion on the analysis of optimum Posit format $< N, es >$, approaches to incorporate Posit arithmetic into the RISC-V ISA, implementation of data type casting, and the development of reconfigurable Posit unit. Chapter 5 discusses the implementation results followed by conclusion of the present work and the possibilities of future directions in Chapter 6.

CHAPTER 2

BACKGROUND

This chapter discusses about the novel Posit arithmetic and the RISC-V open source ISA in detail.

2.1 Posit format

A Posit number is denoted by the representation $< N, es >$, where N, es represents the Posit word size and maximum exponent size respectively. The Posit arithmetic format $< N, es >$ is as shown in Figure 2.1. Similar to the floating-point arithmetic, a 2's complement notation is used for negative number representation.

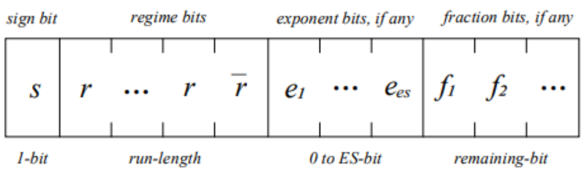


Figure 2.1: $< N, es >$ Posit arithmetic format

2.1.1 Posit fields

Sign bit (S): The MSB bit of the Posit number denotes the sign bit. $S = 0, 1$ represents a positive and a negative number respectively.

Regime bits (rs) : The number of identical bits following the sign bit forms the regime field. It may be a run of zeros followed by a one or vice versa. The regime bits decide the variable K where,

$$K = \begin{cases} rs\text{-}1; \text{ if regime bits begin with one} \\ \text{-}rs; \text{ if regime bits begin with zero.} \end{cases} \tag{2.1}$$

The K value contribute to the formation of total exponent value of a Posit number.

Exponent bits (es) : The maximum number of exponent bits allowed is represented by 'es'. The variable size of the regime along with the constraint on the maximum exponent size enables variability in the size of the exponent and fraction bits too. This flexibility allows a Posit number to have even zero exponent bits. It is to be noted that the regime and exponent bits together are used to evaluate the exponent value in Posit as compared to floating-point arithmetic, where regime bits are totally absent.

Fraction bits : The bits left after the exponent bits if any, forms the fraction field. The fraction field is always preceded by an implied hidden bit, which is always one. The variable 'f' denotes the fraction value.

Unlike the floating-point arithmetic, Posit has only one representation of zero and NaR (Not a Real) number each. This helps in the development of simpler, smaller and faster circuits than by using IEEE-754 arithmetic.

The value of a Posit number X with the format $< N, es >$ can be computed as follows,

$$X = \begin{cases} 0; \text{ if P=00...0} \\ \text{NaR; if P=10...0} \\ (-1)^s \; 2^{2^{es^K}} \; 2^e \; (1{+}f); \text{ otherwise} \end{cases} \tag{2.2}$$

The following examples demonstrate the value computation of a Posit number.
Eg.1: $A < N, es >=< 8, 2 >$ Posit number 00001111 with fields distinguished by underscore is 0_0001_11_1, has $Sign = 0$, $es = 2$,
$K = -rs = -3$ (since regime begins with zero, $K = -rs$), $e = 3$ and $f = 1/2$
Posit Value $= (-1)^0 2^{2^{2^{-3}}} 2^3 (1 + 1/2)$

Eg.2: A Posit number 10001111 having $< N, es >=< 8, 3 >$ with fields distinguished is 1_0001_111. Since the *sign bit* $= 1$, 2's complement needs to be taken. The result 0_1110_001 has $Sign = 0$, $es = 3$, $K = rs - 1 = 2$ (regime begins with 1), $e = 1$ and $f = 0$
Posit Value $= -(-1)^0 2^{2^{3^2}} 2^1 (1 + 0)$

It can be concluded from equation 2.2 that,

1. Smaller es results in higher precision and
2. Larger es results in higher dynamic range

Hence, it is possible to vary precision and dynamic range as per the requirements of the user by manipulating the maximum exponent bits (es). This feature of the Posit arithmetic gives it better acceptance compared to floats.

2.2 RISC-V ISA

The RISC-V ISA mandates only the base extension (*RV32I*). The base RISC-V is a 32-bit processor architecture with 31 general-purpose registers. All instructions are 32 bits long. The other optional standard extensions include M (Multiplication and division), A (Atomic instructions), C (Compressed instruction), F (Single precision floating point instructions), etc.The F extension adds support for single-precision floating-point operations to the base RISC-V ISA. The F extension defines a set of floating-point instructions and registers, as well as the behavior of the floating-point unit (FPU). With energy efficiency as a concern, RISC-V ISA endeavors to maintain the base as well as each standard extension constant over time and new instructions are layered as further optional extensions, ie. regardless of the addition of non-standard extension, the base integer extension continues to be fully supported. RISC-V ISA with 32-bit instruction format supports four major custom opcodes. These custom opcodes have been reserved to develop user specific applications. This feature of the ISA enables easy integration of customized unit to the core.

CHAPTER 3

LITERATURE SURVEY AND RESEARCH GAP

A study about the novel *Posit* arithmetic and the open-source *RISC-V* ISA is carried out. The existing approaches for replacing the currently used IEEE-754 arithmetic with Posit in RISC-V ISA is explored. Moreover, the feasibility and significance of Posit arithmetic is evaluated through a comparative performance analysis of Posit and floating-point arithmetic in relevant applications such as the self driving cars, weather forecasting etc.

Manish Kumar Jaiswal and Hayden K. H. So (2019) [1] proposed algorithmic flows, pipelined architecture and open source parametrized Verilog HDL generator for Posit arithmetic architectures pertaining to addition/subtraction, multiplication and division.

Souradip Sarkar et al. (2019) [2] focused on performance analysis using a novel reconfigurable hardware accelerator of Posit number format for signal processing algorithms and performed comparison of Posit with IEEE-754 standard. They came to a conclusion that the Posit number representation has a much larger dynamic range and precision compared to the IEEE-754 standard for the same bit width. Additionally, Posit is capable of providing significant performance gain for algorithms in the physical layer of communication systems.

Rohit Chaurasiya et al. (2018) [3] designed a completely parameterized area and energy efficient Posit arithmetic unit generator. They have experimentally demonstrated that a n-bit IEEE-754 compliant adder/multiplier can be safely replaced with a suitable m-bit Posit, where $m < n$. Additionally, the Posit arithmetic unit designed is shown to consume less energy and area compared to the state-of-art realizations. Both, synthesis for FPGA and ASIC are performed. The proposed posit arithmetic adder/subtracter is implemented on a Zedboard with a Zynq-7000 SoC.

Sugantha Tiwari et al. (2019) [4] provided insights on the integration of Posit arithmetic with RISC-V core. Two separate approaches were discussed for the Posit integration to the RISC-V core. The first approach leverages the standard 'F' extension, thereby enabling a quick bring up of the design. The second approach is by leveraging the custom opcode space of RISC-V ISA. The implementation and analysis of both the approaches culminated in the inference that leveraging custom opcode space approach needs to be undertaken to facilitate the co-existence of the floating-point and Posit arithmetic. The Posit unit has been enhanced to support two *es* values, thereby enabling dynamic switching between higher precision and dynamic range at run time. They have also presented analysis of various software applications running on the core that provides better performance in terms of quality as compared to IEEE-754 standard. However the paper does not reveal the details of the implementation of reconfigurability.

Arunkumar M. V. et al. (2020) [5] implemented and integrated a Posit Processing Unit (PPU) into rocket chip SoC generator. A discussion on using the standard RISC-V ISA 'F' and 'D' extensions for Posit arithmetic or utilizing the custom opcode space for replacing IEEE-754 FPU with the Posit processing unit is done.

Junjie Hou et al. (2018) [6] researched on the FPGA implementation of Posit arithmetic for extending floating-point IP cores for FPGA based scientific data analytics. Their work on $< 32, 3 >$ type Posit and floating-point single precision IP core lead to the conclusion that Posit exhibits better superiority in representation and dynamic range than IEEE-754 format. Furthermore, larger word size formats such as that with 64 and 128 bits were found to enhance the precision.

Riya Jain et al.(2020) [7] presented a consolidated general purpose processor based framework consisting of melodica and clarinet. The melodica is a Posit arithmetic core that implements parametric fused-multiply-accumulate and supports quire data type, while clarinet is a melodica enabled processor based on RISC-V ISA. Their work claims to be the first ever quire enabled RISC-V CPU.

Riaz-ul-haque Mian (2020) [8] developed an evolution framework for software-hardware co-design solutions of decimal computation using the RISC-V ecosystem.

Macro Cococcioni et al. (2018) [9] discussed the introduction of an integrated Posit processing unit (PPU) as an alternative to FPU for deep neural networks in automotive applications. The study lead to the conclusion that implementing a PPU with the co-processor is a promising way to speed up the DNN inference space.

Zachariah Carmichael et al. (2019) [10] proposed an exact multiply and accumulate algorithm namely, deep Positron for accelerating ultra-low precision ($<= 8-bit$) DNNs with Posit numerical format. They conducted experiments on deep Positron architecture for multiple low-dimensional datasets and showed that 8-bit Posits achieve better performance compared to their fixed and floating-point counterparts.

Macro Cococcioni et al. (2021) [11] focused on reducing the number of bits needed to represent the weights of DNNs using Posit number system. Moreover, the paper exploits RISC-V vectorization to expedite the format encoding /decoding evaluation of activation functions and performs the computation of core DNN matrix vector operations. It was inferred that the open source hardware platform like RISC-V along with the open source DNN software implementations may enable a new class of completely open DNN computing environments.

Milan Klower et al. (2019) [12] worked on trying to minimize word size of the weather forecast data from 32-bit to 16-bit. Research was carried out to compare the rounding error between half precision floats and 16-bit Posit format. Using the shallow water (a standard forecast model) they concluded that $< 16,1 >/< 16,2 >$ Posit had more accuracy than 32-bit floats. Further, they recommended $< 16,2 >$ Posit as the standard format for weather and climate models.

The literature survey is summarized in Table 3.1.

Paper	**Contribution**
Manish Kumar Jaiswal and Hayden K. H. So (2019)	Proposed Posit arithmetic architecture for addition/subtraction, multiplication, division
Souradip Sarkar et al. (2019)	Concluded that Posit has larger dynamic range and precision compared to IEEE-754 standard
Rohit Chaurasiya et al. (2018)	Designed a parameterized Posit arithmetic unit and stated that n-bit IEEE-754 compliant adder/multiplier can be replaced with a m-bit Posit, where $m < n$
Sugantha Tiwari et al. (2019)	Proposed two approaches for integration of Posit arithmetic with the RISC- V core and implemented dynamic switching of es at run time
Arunkumar M. V. et al. (2020)	Integrated a Posit Processing Unit into rocket chip SoC generator
Junjie Hou et al. (2018)	Concluded that larger word size Posit formats enhance the precision
Riya Jain et al.(2020)	Presented Posit arithmetic core that implements parametric fused-multiply-accumulate and supports quire data type
Riaz-ul-haque Mian (2020)	Developed evolution framework for software-hardware co-design solutions of decimal computation using the RISC-V ecosystem
Macro Cococcioni et al. (2018)	Concluded that PPU with the co-processor speeds up the DNN inference space
Zachariah Carmichael et al. (2019)	Concluded that 8-bit Posits achieve better performance compared to their fixed and floating-point counterparts
Macro Cococcioni et al. (2021)	Reduced the number of bits needed to represent the weights of DNNs using Posit number system and expedited the format encoding /decoding evaluation of activation functions
Milan Klower et al. (2019)	Concluded that $< 16,1 >/< 16,2 >$ Posit has more accuracy than 32-bit floats and recommended $< 16,2 >$ Posit as the standard format for weather and climate models

Table 3.1: Literature summary

3.1 Research gap

From the above literature survey, the relevance and benefits of Posit over floating-point arithmetic is evident especially in applications demanding higher dynamic range, precision and minimal error. Also, various approaches have been proposed for the integration of Posit with the open source RISC-V ISA. But in all the previous works, the Posit functional unit is implemented as an accelerator and not within the core pipeline, which enables co-existence of the floating and Posit arithmetic. Methods intended to enable the co-existence of these arithmetic (floating-point and Posit) within the core needs to be investigated as this may contribute to a reduction in timing requirements in contrast to the accelerator approach. The formulation of new instructions and development of corresponding hardware to handle the mixed operand type (Integer, Float and Posit) operations are highly desirable. Furthermore, the 32-bit Posit unit is enhanced to support two different exponent sizes with minimum overheads compared to the existing method. This enables run-time switching between higher dynamic range and precision in accordance with the application being dealt with.

CHAPTER 4

POSIT ENHANCED RISC-V ISA

The work presents the efforts undertaken for the realization of Posit enhanced RISC-V ISA. Additionally, hardware innovations capable of bolstering the efficacy of the unit (Posit enhanced RISC-V core) developed is explored.

4.1 Development of posit arithmetic unit

This section discusses about the development of Posit unit capable of performing elementary arithmetic operations such as addition/subtraction, multiplication and division. A parameterized unit that handles any Posit size 'N' and exponent size 'es' is as shown in Figure 4.1. The block diagram of basic Posit arithmetic flow is depicted in Figure 4.2.

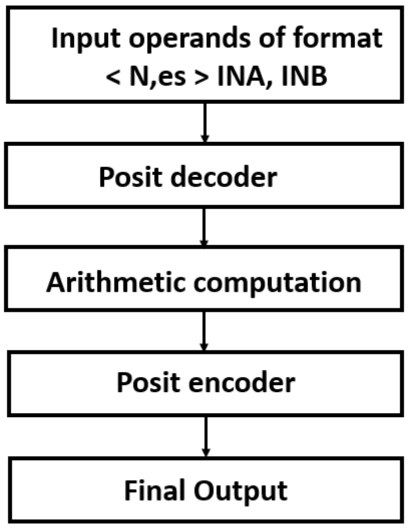


Figure 4.1: Parametrized data flow for <N,es> Posit format

4.1.1 Posit decoder

A Verilog HDL code with parametrized inputs as Posit word size (N), Posit exponent size (es), regime value storage size (rs) and two N-bit operands (A, B) is developed. Initially, the algorithm checks for two exceptional cases (0 and infinity). A Posit input

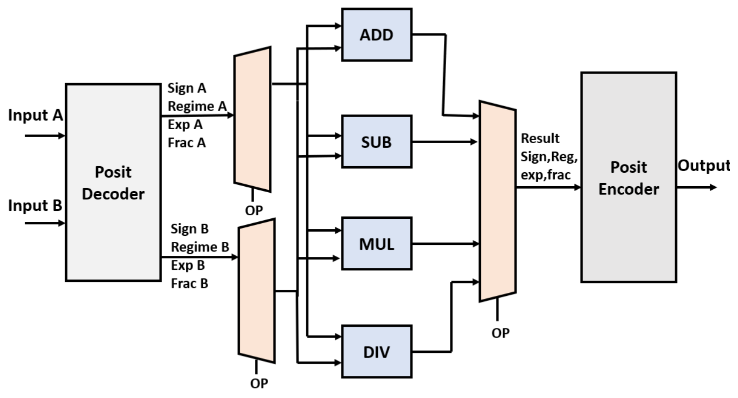


Figure 4.2: Posit arithmetic block diagram

with all N-bit zero's depicts an exception of zero while, all bits zero except the MSB bit forms Posit infinity representation. The exceptions are fed to the subsequent blocks to ensure that the arithmetic operations takes care of these exceptions during arithmetic computation. The exception check is followed by the extraction of four fields in the Posit format namely sign, regime, exponent and fraction. The decoder block then outputs the four fields of the inputs A and B. The algorithm of the Posit extraction block is as shown in the Figure 4.3.

```
Algorithm 1 Algorithm for Posit Decoder
  Given
  N: Posit Word Size
  ES: Posit Exponent Size
  RS: $\log_2 N$ (Posit Regime Value Storage Size)
  Input Operands: A[N-1:0], B[N-1:0]
  Data Extraction
  Sign, Regime Check bit, Regime value, Exponent, Fraction, Exceptions
  Check for Exceptions
  $Z1 \leftarrow \sim A$
  $Z2 \leftarrow \sim B$
  $Z \leftarrow Z1 \& Z2$
  $Inf1 \leftarrow A[N-1] \& \sim |A[N-2:0]$
  $Inf2 \leftarrow B[N-1] \& \sim |B[N-2:0]$
  $Inf \leftarrow Inf1 | Inf2$
  Sign Bit Extraction from Input A
  $A\_sign \leftarrow A[N-1]$
  if $A\_sign$ is 1 then
    $A[N-2:0] \leftarrow -A[N-2:0]$
  else
    $A[N-2:0] \leftarrow A[N-2:0]$
  end if
  Regime Extraction from A
  $A\_reg\_S \leftarrow A[N-2]$
  if $A\_reg\_S$ is 1 then
    $A\_tmp \leftarrow -A$
  else
    $A\_tmp \leftarrow A$
  end if
  $A\_reg\_N \leftarrow LeadingOneDetectorof(A\_tmp[N-2:0])$
  Exponent Extraction from A
  $A\_flush[N-4:0] \leftarrow A[N-4:0] << (A\_reg\_N - 1)$
  $A\_exp[es-1:0] \leftarrow A\_flush[N-4:N-4-es]$
  Fraction Extraction from A
  $A\_frac[N-es-3:0] \leftarrow 1'b1, A\_flush[N-es-4:0]$
  Field Extraction from Input B
  B_sign, B_reg_S, B_reg_N, B_exp, B_frac
```

Figure 4.3: Posit decoder algorithm

4.1.2 Posit arithmetic processing

The sign, regime, exponent and fraction fields so obtained are fed to a demux intended to route the data into appropriate computational blocks (adder/subtractor, multiplier and divider) based on the arithmetic operation signal *OP*. The operational flow of various arithmetic blocks are discussed.

Posit adder/subtractor

The adder identifies the large and the small operands among the inputs, *A* and *B*. After comparison, the operands are swapped if necessary to ensure that *A* and *B* denotes the large and the small operands respectively. This is followed by the computation of sign, regime, exponent and fraction of the result which are then fed to the encoder for final processing. The algorithm of Posit adder is presented in Figure 4.4. The Posit subtractor block follows exactly the same flow as that of Posit adder except that 2's complement of the operand *B* is performed before the arithmetic computation.

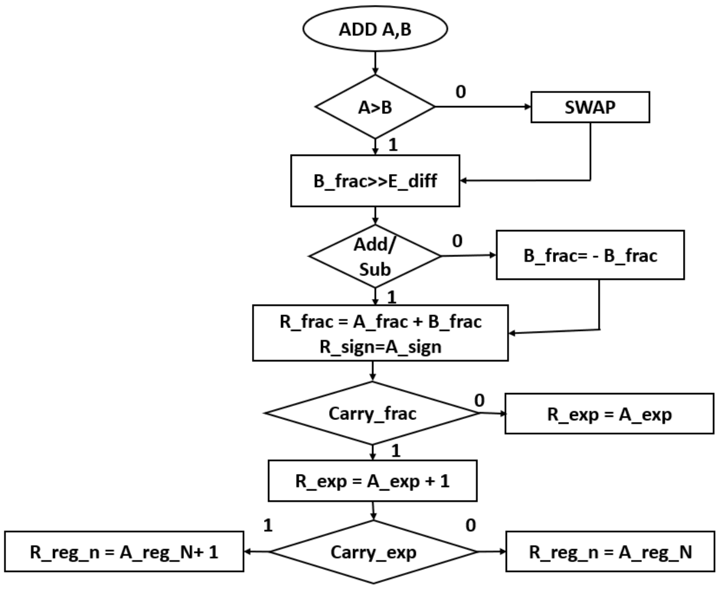


Figure 4.4: Algorithm of Posit adder/subtractor

Posit multiplier

The Posit multiplication is implemented in accordance to the algorithm developed and implemented by Manish Kumar Jaiswal and Hayden K. H. So (2019) [1]. The fraction and exponent fields of the result are obtained by direct multiplication and addition of the fraction and exponent of the operands respectively. The normalization of result fraction and exponent is done based on the carry obtained in the result fraction and exponent computation. The MSB of the regime field (*Result_reg_S*) and the number of bit repetition before the first complement bit (*Result_reg_N*) after *Result_reg_S* are also routed to the Posit encoder. Finally, sign of the result is obtained by *XOR*ing the operand signs. The algorithm of Posit multiplier is shown in Figure 4.5 .

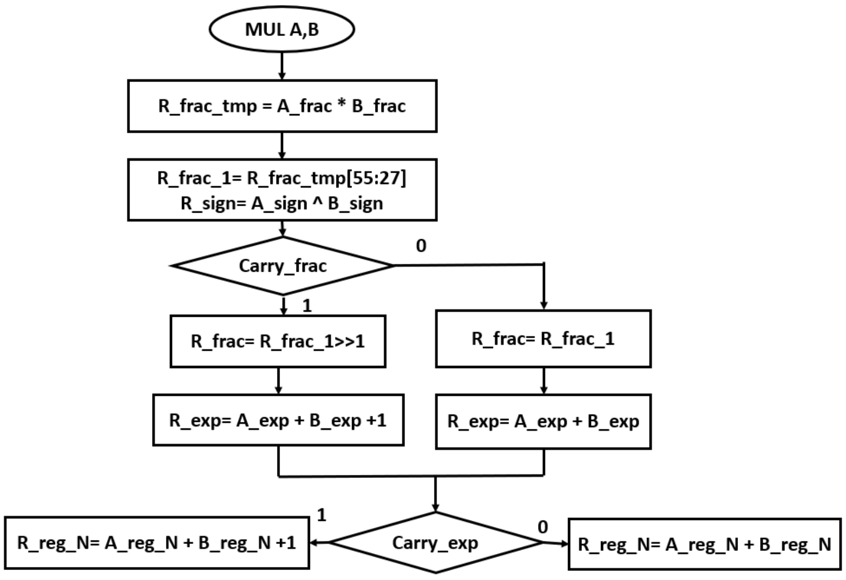


Figure 4.5: Algorithm of Posit multiplier

Posit division block

The Posit division is implemented in accordance to the algorithm developed and implemented by Manish Kumar Jaiswal and Hayden K. H. So (2019) [1]. The algorithm of Posit division operation is presented in Figure 4.6. The fraction of the result is obtained by dividing the fraction fields of the operands. Likewise, the result exponent is computed by subtracting the operand exponents. The *Result_reg_S*, *Result_reg_N*, and *Result_sign* are also computed.

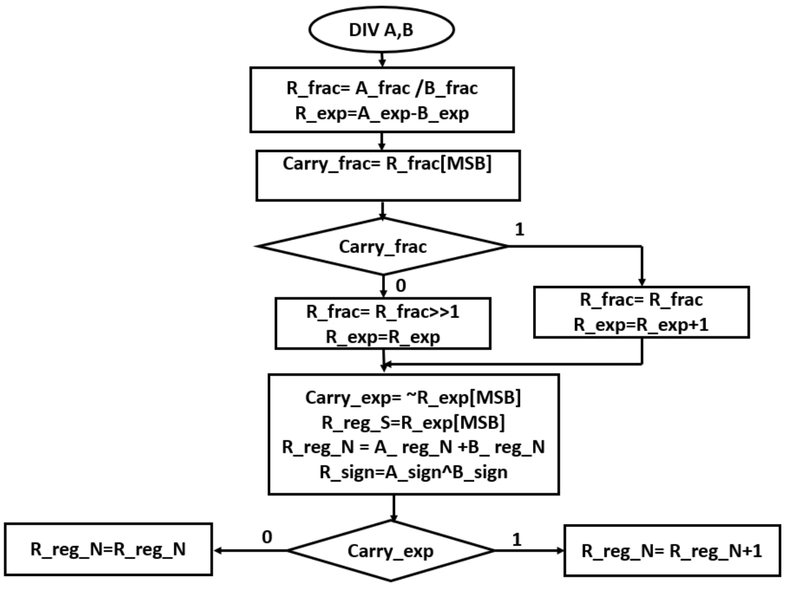


Figure 4.6: Algorithm of Posit division block

Finally, the result fields such as result sign, regime sign and number, exponent and fraction from each of the three arithmetic computational block are fed to a multiplexer to be appropriately routed to the Posit encoder based on the operation (OP) undertaken.

4.1.3 Posit encoder

The Posit encoder algorithm is concerned with the construction, rounding and final processing of the Posit result as presented in Figure 4.7. The regime, exponent and fraction obtained as input is packed into a variable named *REF*. Additionally, three zeros are also appended to the *REF* to serve as guard, round and sticky bit. The *REF* thus obtained is right shifted *Result_ reg_ N* times and vacant part of MSB is filled with *Result_ reg_ S*. Further, rounding is carried out depending on the value of *Result_ reg_ N*. Finally, the result sign is appended to *REF* and the MSB N-bits form the final output, after taking care of the exceptions (zero and infinity). This meets the first objective of the design and implementation of the Posit unit.

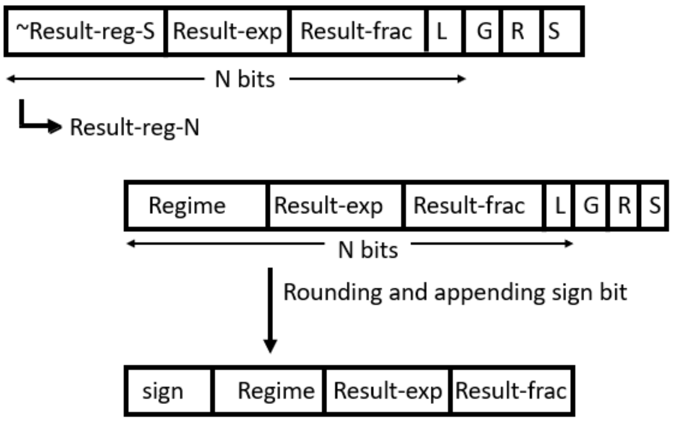


Figure 4.7: Algorithm of Posit encoder

4.2 Application specific discussion on analysis of optimum value of parameters $< N, es >$

The requirement of the dynamic range and precision varies with application. The Posit arithmetic caters to this varying demand and therefore beats floats. Hence the optimal $< N, es >$ is exclusively dependent on the application concerned. This section discusses a few relevant applications based on Gustafson J. and Yonemoto I. (2017) [13] and generalizes their optimal $< N, es >$.

4.2.1 Big data analytics

The big data analytics deals with the real time processing of data strictly within certain time constrains. The requirement of large data size and the speed or throughput optimization are the major parameters of concern. Therefore, big data analytics demands large dynamic range and precision too. The constraints on the $< N, es >$ is depicted in Table 4.1. It can be inferred from the table that, the optimal Posit format for big data analytics is $< 32, 3 >$.

Constraints	Reason
$N < 32$	Lower dynamic range
$N > 32$	Memory word size constraint
$es < 3$	Lower dynamic range
$es > 3$	Lower precision

Table 4.1: Constraints on $< N, es >$ Posit format for big data analytics

4.2.2 Audio processing

Consider an audio clip with sample rate of 44.1KHz and a buffer size of 4096 frames. It was observed that processing with $< 8, 0 >$ Posit format gives a 5% improvement in precision compared to processing with $< 8, 3 >$ Posit. Hence, $0 < es < 3$ and more closer to zero is preferred for audio processing applications.

4.2.3 Machine learning applications

Machine learning applications are also found to have substantial benefits through Posit integration. Deep neural networks such as autonomous driving applications demonstrated that a Posit unit with $< N, es > = < 8, 0 >$ within the co-processor speeds up the inference phase instead of quantization. Another instance of deep learning is the approximation of sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ using Posits. The approximation is done by simply flipping the MSB followed by a right shift of two times.
Eg.: If Posit Number= $8'b10110000$
then $f(x) = \frac{1}{1+e^{-x}} = 8'b00001100$

Consider a neural network to evaluate the boolean formula on a vector of ten variables. Eg.: $f(v) = v[1]^{(v[3]\& \ v[4])}$. The comparison to the actual results illustrated an accuracy of 98% and 56% for Posit $< 8, 0 >$ and $< 16, 0 >$ respectively. Also $es = 0$ is desired as the application here urges for precision.

4.2.4 Generalization

Generally, the standard Posit configuration used for single precision and double precision instruction format is $< 32, 2 >$ and $< 64, 3 >$ respectively. Hence, the exponent size for single precision format lies between 0 and 2, and that for the double precision format varies between 0 and 3 depending on the regime. Additionally, a study of various $< N, es >$ Posit formats for basic arithmetic operations were undertaken. It was concluded that the latency of a particular arithmetic computation is independent of the $< N, es >$, while the maximum register width depends on both N and es. The results of the analysis are depicted in Table 4.2 and 4.3. For an N-bit Posit, the maximum fraction field size is $(N - es - 3)$. Among the basic Posit arithmetic operations, multiplication demands the maximum fraction field size. The maximum register field size of the fraction field in case of multiplication is $2 * (N - es - 3) - 1$. The maximum register width refers to the combined field size for different Posit fields, ie., *maximum register width* = (*maximum fraction register width*) +1 (*sign bit*) +es (*maximum exponent field size*) + 2 (*minimum regime field width*).

Posit Format $< N, es >$	**Latency**
$< 8, 2 >$	8
$< 8, 3 >$	8
$< 16, 2 >$	8
$< 16, 3 >$	8
$< 32, 2 >$	8
$< 32, 3 >$	8

Table 4.2: Analysis of latency for various $< N, es >$ Posit formats for Posit multiplication

Posit Format $< N, es >$	**Maximum Fraction Field Register Width**	**Maximum Register Width**
$< 8, 2 >$	5	10
$< 8, 3 >$	3	9
$< 16, 2 >$	21	26
$< 16, 3 >$	19	25
$< 32, 2 >$	53	58
$< 32, 3 >$	51	57

Table 4.3: Analysis of maximum register width for various $< N, es >$ Posit formats for Posit multiplication

4.3 Approaches to enhance RISC-V ISA (***RV32IMF***) with posit arithmetic

The integration of Posit arithmetic to the RISC-V ISA can be facilitated through three separate approaches. The first approach deals with Posit integration within the core by replacing the standard F extension. This approach does not support the co-existence of the floating-point and Posit arithmetic. The second approach of Posit incorporation is as an accelerator by utilizing the custom opcode space in RISC-V ISA using a standard co-processor interface. The third approach is similar to that of the first approach in terms of its location, ie. as a tightly coupled unit to the core, but efforts are made to enable the co-existence of the floating-point and the Posit arithmetic. This section describes each of these approaches in detail.

4.3.1 Approach 1: Posit integration as a tightly coupled unit by replacing the standard F extension

This approach enables Posit integration within the core. Here, the Posit instructions are incorporated by replacing the standard F extension of the RISC-V ISA. The instruction encoding remains the same. The Posit unit, similar to the floating-point maintains a separate register bank with 32 ($P0 - P31$) registers along with a Control and Status Register (CSR). The Posit CSR (PCSR) is designed with slight modification to its float counterpart. The float and Posit CSR is juxtaposed in Figure 4.8

and 4.9 respectively to illustrate the modifications brought about. The PCSR differs from FCSR in the requirement of *es*-mode field and absence of flags expect the Division by Zero (DZ) flag. Hence, the PCSR[3] is reserved for DZ flag and five bits of the PCSR (PCSR[5:4,2:0]) can be used for *es* field. Similar to the F extension, we consider 32-bit Posit and expects the requirement of only five bits for *es* in all applications. The Posit number field extractions are in accordance to the *es* value. Unlike floats, that supports multiple rounding modes, Posit supports only round to nearest, ties to even rounding mode depicted with PCSR[7:6] which is hence always set to zero. The reserved field PCSR[31:8] is kept intact.

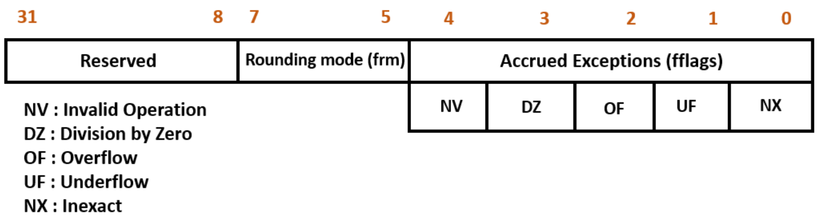


Figure 4.8: Floating-point Control and Status Register (FCSR)

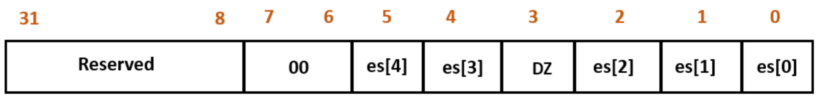


Figure 4.9: Posit Control and Status Register (PCSR)

4.3.2 Approach 2: Posit integration as an accelerator by utilizing the custom opcode space in RISC-V ISA

This approach supports more versatile instructions compared to the previous approach. Here, the Posit unit is integrated as an accelerator and no modifications are needed in the core. An accelerator can be defined as a separate architectural substructure (on a different die or on the same chip) that is designed using a different set of objectives than the base processor, where these objectives are derived from the requirements of a special group of applications. The integration is made possible through utilization of a standard co-processor interface. This approach facilitates the co-existence of Posit and floating-point arithmetic in contrast to approach 1 (Section 4.3.1) in which, Posit stays as a tightly coupled unit to the core. The RISC-V ISA offers custom 0/1/2/3 opcode space exclusively for the development of application specific non-standard extensions. This feature can be leveraged for development of the Posit co-processor unit. A standard co-processor interface namely, Rocket Custom Co-processor (RoCC) is used as an interface here.

RoCC: Rocket Custom Co-processor

The RoCC follows a standard instruction format that enables two source and one destination register values to be passed to and from the accelerator respectively. The

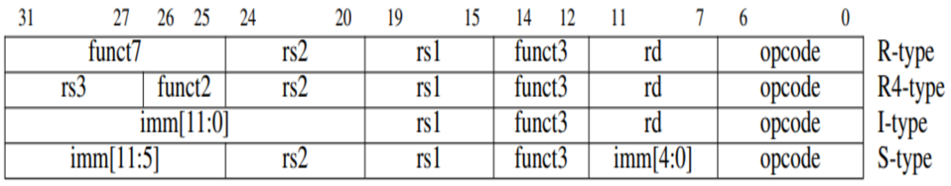
Posit instructions uniquely identified by using the custom opcode assigned to it is passed to the Posit accelerator. The standard RoCC instruction encoding is as shown in Figure 4.10. The *rs1*, *rs2* and *rd* represents the source and destination register values respectively. The *xd*, *xs1* and *xs2* selects between the Posit and the base integer register banks, which is presented in Table 4.4. The RISC-V 32 bit instruction type used in *RV32IF* and its equivalent mappings to RoCC instruction format is presented in Figure 4.11 and 4.12 respectively.

31 25	24 20	19 15	14	13	12	11 7	6 0
funct7	rs2	rs1	xd	xs1	xs2	rd	opcode
7	5	5	1	1	1	5	7

Figure 4.10: Standard RoCC instruction encoding

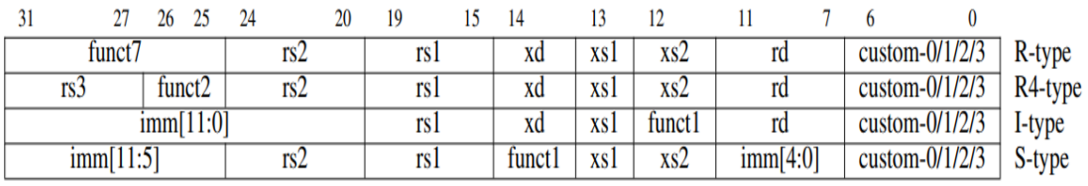
Instruction Handling Processor	xs1	xs2	xd
RISC-V Core Processor	1	1	1
Posit Co-Processor	0	0	0

Table 4.4: RoCC interface selection between integer and posit data



31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
rs3	funct2	rs2	rs1	funct3	rd	opcode	R4-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type

Figure 4.11: *RV32I* instruction mapping to *RV32IF*



31 27	26 25	24 20	19 15	14	13	12	11 7	6 0	
funct7		rs2	rs1	xd	xs1	xs2	rd	custom-0/1/2/3	R-type
rs3	funct2	rs2	rs1	xd	xs1	xs2	rd	custom-0/1/2/3	R4-type
imm[11:0]			rs1	xd	xs1	funct1	rd	custom-0/1/2/3	I-type
imm[11:5]		rs2	rs1	funct1	xs1	xs2	imm[4:0]	custom-0/1/2/3	S-type

Figure 4.12: Mapping of *RV32IF* to RoCC standard instruction format

The Posit accelerator

The Posit accelerator architecture is presented in Figure 4.13. The co-processor is designed by the addition of Posit register bank to the Posit block as described in Section 4.1. The Posit accelerator receives two source registers along with the Instruction Register (IR) as inputs. The inputs to the Posit decoder is selected between the integer register and Posit register values depending on the values of *xs1* and *xs2*. The final Posit result is written back to the destination register of the Posit register bank if $xd = 1$ else, it is written to the integer destination register in the core.

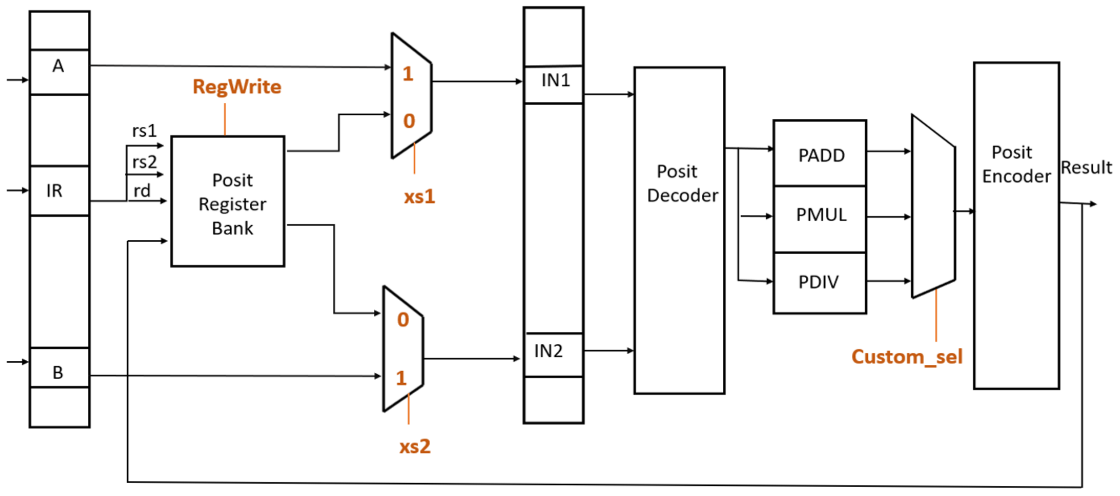
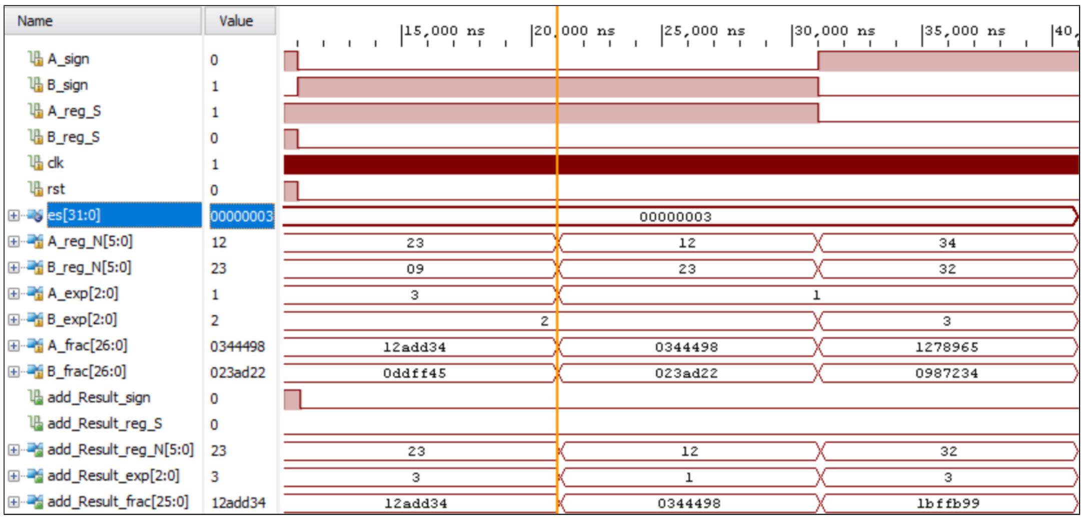


Figure 4.13: Posit accelerator architecture

Simulation results and resource utilization

The code is implemented on Xilinx ARTIX-7 FPGA (xc7a100tcsg324-1). The simulation results of Posit adder, multiplier and the division block within the Posit accelerator for $< 32, 2 >$ is as per Figure 4.14, 4.15 and 4.16 respectively. The number of clock cycles for Posit addition, multiplication and division are as in Table 4.5. The correctness of the arithmetic computation is validated using standard Posit calculator. Resource utilization for basic arithmetic units within the Posit accelerator for $< 32, 2 >$ and $< 32, 3 >$ are presented in Table 4.6 and 4.7.



Name	Value			
A_sign	0			
B_sign	1			
A_reg_S	1			
B_reg_S	0			
clk	1			
rst	0			
es[31:0]	00000003	00000003		
A_reg_N[5:0]	12	23	12	34
B_reg_N[5:0]	23	09	23	32
A_exp[2:0]	1	3	1	
B_exp[2:0]	2	2		3
A_frac[26:0]	0344498	12add34	0344498	1278965
B_frac[26:0]	023ad22	0ddff45	023ad22	0987234
add_Result_sign	0			
add_Result_reg_S	0			
add_Result_reg_N[5:0]	23	23	12	32
add_Result_exp[2:0]	3	3	1	3
add_Result_frac[25:0]	12add34	12add34	0344498	1bffb99

Figure 4.14: Posit adder simulation

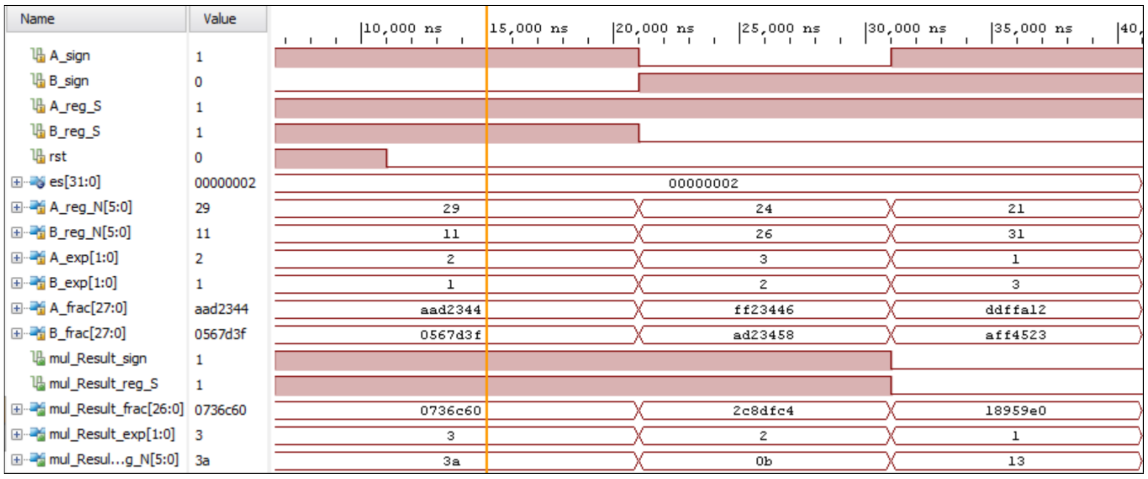


Figure 4.15: Posit multiplier simulation

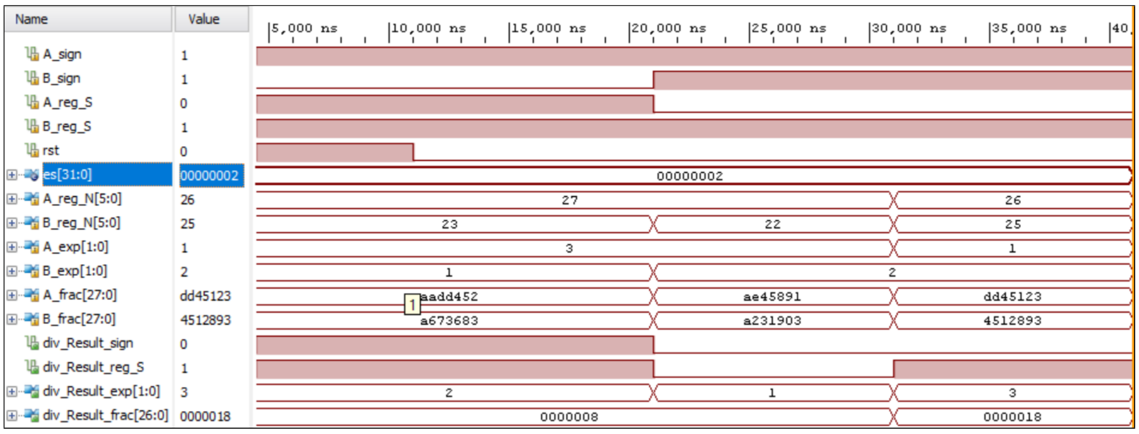


Figure 4.16: Posit division block simulation

Operation	Clock cycles
ADD/SUB	5
MUL	8
DIV	12

Table 4.5: Number of clock cycles for Posit arithmetic operations

Module	$es = 2$	$es = 3$
ADD/SUB	722	704
MUL	798	765
DIV	869	813

Table 4.6: LUT utilization for arithmetic units within Posit accelerator for $N = 32$-bits

Module	$es = 2$	$es = 3$
ADD/SUB	38	35
MUL	46	43
DIV	51	47

Table 4.7: Flip-Flop utilization for arithmetic units within Posit accelerator for $N = 32$-bits

4.3.3 Approach 3: Posit integration as a tightly coupled unit using custom opcode space without replacing the standard F extension

Approach 3 aims to develop a *RV32IMF_XPosit* unit, which incorporates Integer (I), Integer Multiplication and Division (M), Floating-point (F) and Posit extensions within the core. This circumvents the requirement of an accelerator exclusively for Posit arithmetic. Approach 3 is more attractive compared to approach 1 (Section 4.3.1) in terms of co-existence of Posit and floating-point arithmetic. This approach in-fact speeds up the computation as the instruction is fed to the Posit unit at the execute stage instead of the write-back stage as in case when Posit is integrated as an accelerator (approach 2). The advantage in terms of speed for approach 3 can be attributed to the elimination of the stalling of Posit instructions for the last two stages (memory and write-back stage) as in approach 2 (Section 4.3.2). An illustration of the RISC-V core with the Posit and floating-point units are presented in Figure 4.17. The Posit extension is incorporated by replacing the opcode space of the standard F extension with custom opcode as demonstrated in Figure 4.18 and 4.19 respectively. Here, custom-0 opcode is selected to uniquely identify the Posit instructions. The architecture of the Posit and the floating-point unit that supports basic arithmetic operations are presented in Figure 4.20 and 4.21 respectively. This meets the second objective of the incorporation of the Posit unit within the RISC-V core, thereby facilitating the co-existence of the floating-point and Posit arithmetic.

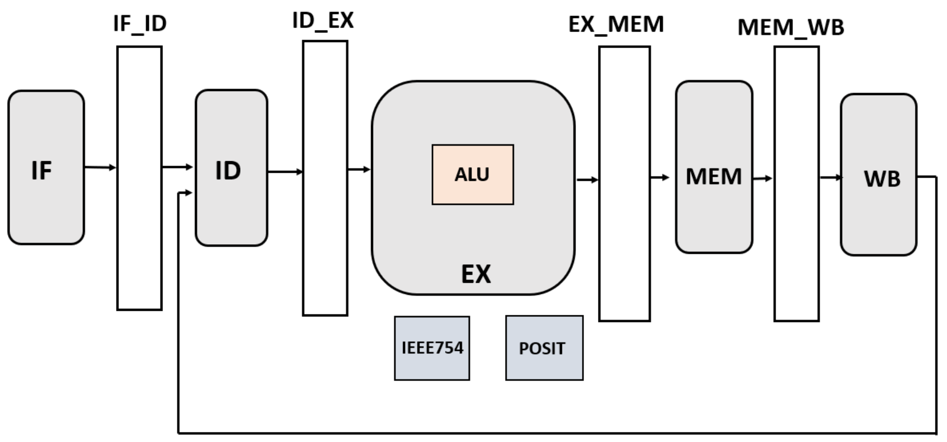
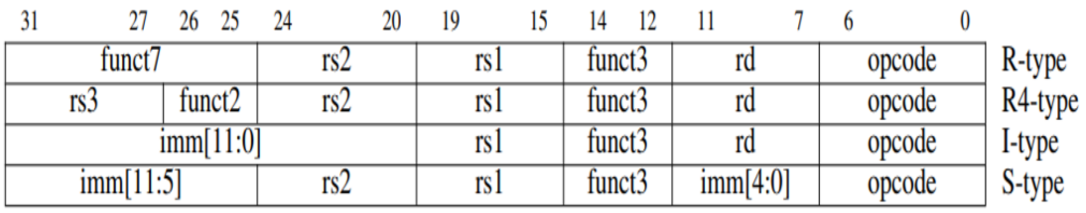
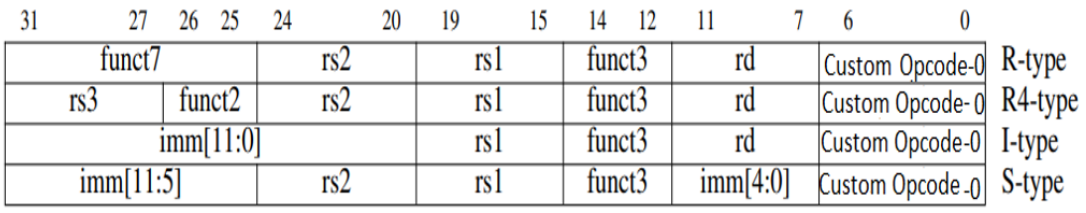


Figure 4.17: RISC-V core with Posit and floating-point blocks



31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
rs3	funct2	rs2	rs1	funct3	rd	opcode	R4-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type

Figure 4.18: RISC-V 32 bit instruction format used for *RV32IF*



31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	Custom Opcode-0	R-type
rs3	funct2	rs2	rs1	funct3	rd	Custom Opcode- 0	R4-type
imm[11:0]			rs1	funct3	rd	Custom Opcode-0	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	Custom Opcode -0	S-type

Figure 4.19: RISC-V 32 bit instruction format used for *RV32IF_XPosit*

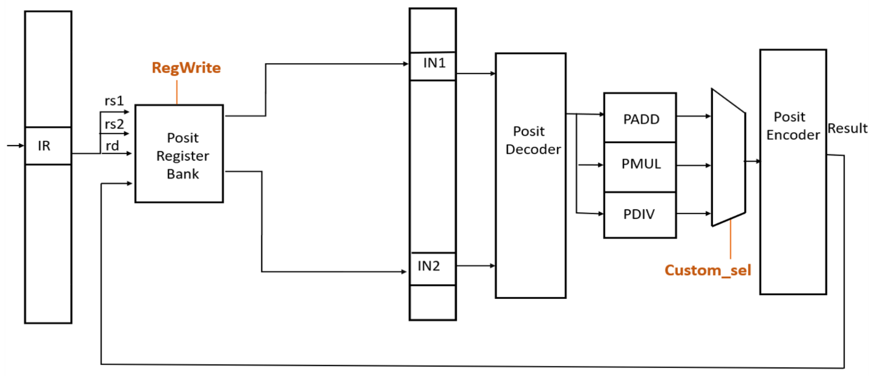


Figure 4.20: Architecture of Posit arithmetic unit within RISC-V core

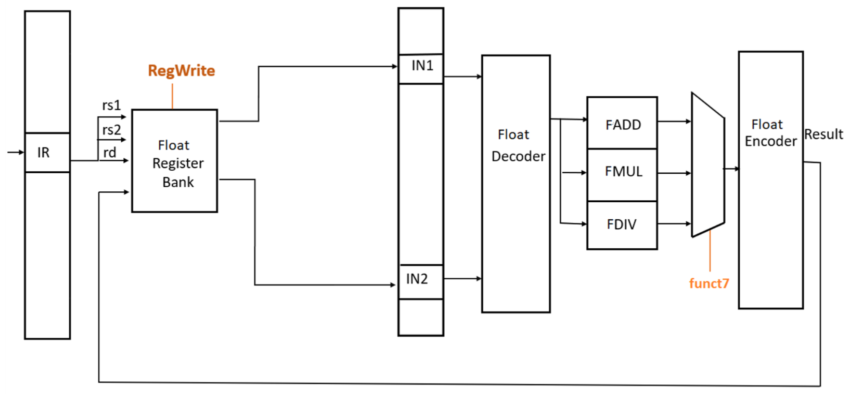


Figure 4.21: Architecture of floating-point arithmetic unit within RISC-V core

4.4 Implementation of data type casting in *RV32IMF_XPosit*

The *RV32IMF_XPosit* processor developed contains a floating-point and a Posit block with separate register banks within the core. Each of the FPU and Posit block can access the integer register bank and perform necessary computations. But no efforts were taken up to deal instructions with mixed operand types. Hence, implementing data type casting in hardware can be an efficient method to address this issue. Data type casting can be implemented through two different approaches. This section describes these approaches in detail.

4.4.1 MOT: Mixed Operand Type block

The MOT block is designed to handle instructions meant to perform arithmetic operations involving mixed operand types. The block converts data type of the source operands to that of the destination operand type and routes it to the appropriate destination functional unit for performing the necessary operation. For instance, the MOT block handles an operation $F6 = P2 + I5$ by converting all the source types (Posit and Integer) to destination type (Float) before addition. The computational flow of the instruction is as follows.
$P2 \to F$ Type; $I5 \to F$ Type; $F6 = (P2 \to F) + (I5 \to F)$
Here, after converting the source operands to float data type, the converted operands are then routed to the FPU for addition. The result of the floating-point addition is then written back to the destination register in the float register bank.

MOT instruction format

The MOT block handles only R-type instructions having two source and a destination register as operands. The MOT instruction format is presented in Figure 4.22. The block gets activated using a custom opcode. The data type of the operand is identified by two bits, specifically '01' for integer, '10' for float and '11' for Posit. From the instruction, the destination type (*xd*) and the two source types (*xs1* and *xs2*) are {IR[31],IR[14]}, {IR[30],IR[13]} and {IR[29],IR[12]} respectively. Similarly, two source and the destination registers are denoted by IR[24:20], IR[19:15] and IR[11:7] respectively. The IR[28:25] field (*funct4*) specifies the type of the operation to be executed.

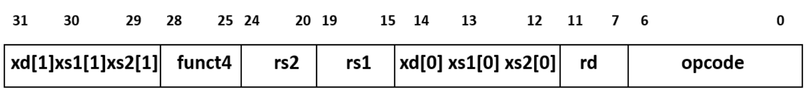


Figure 4.22: MOT instruction format

MOT block architecture

The architecture of the MOT block is depicted in Figure 4.23. The MOT block receives the source operands from all the three register banks, namely Integer, Float and Posit. The multiplexer at the initial stage selects the source operands depending on the value

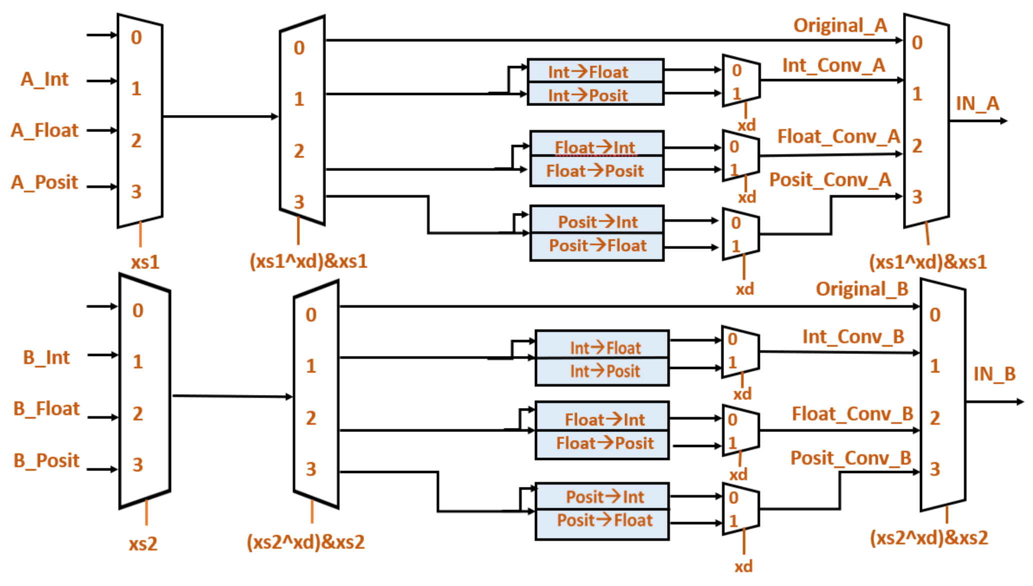


Figure 4.23: MOT block architecture

of *xs1* and *xs2*. The second stage demultiplexer checks if the source and the destination data type are different and decides whether a conversion is required. The operands are then routed to the appropriate conversion block if needed. The converted operands of the destination type is then routed to the appropriate functional block (Integer ALU, Posit or Float block) depending on the *xd* value.

Integration of MOT block to the RISC-V core

The illustration of the MOT block integrated *RV32IMF_XPosit* is presented in Figure 4.24. The separation of the register banks from the respective functional units is the major change in the MOT integrated core. The inputs to the Float, Posit and the Integer arithmetic block is routed from the MOT block in case of a mixed operand type instruction otherwise, it is directly fed from the respective register banks.

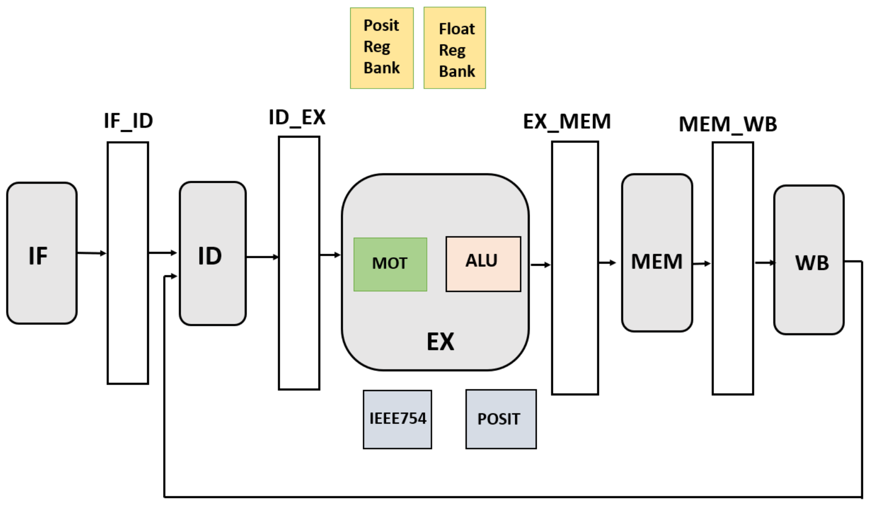


Figure 4.24: MOT block integrated RISC-V core

4.4.2 DTC: Data Type Converter block

Unlike the MOT block, the DTC block do not handle instructions involving arithmetic computation. It is solely designed for data type conversion from one to the other. The DTC block converts the source type operand to that of destination type and writes back the converted value into the destination register bank.

DTC instruction format

The DTC block handles instructions with a single source and a destination as operands. DTC block is also uniquely identified using a custom opcode. The DTC instruction format is presented in Figure 4.25. From the instruction, the destination type (*xd*) and the source type (*xs*) can be obtained form IR[23:22] and IR[21:20] respectively. The introduction of these data type casting instructions satisfies the fifth objective of the thesis.

31 24	23 20	19 15	14 12	11 7	6 0
00000000	xd[1] xd[0] xs[1] xs[0]	rs	000	rd	opcode

Figure 4.25: DTC instruction format

DTC block architecture

The architecture of the DTC block is depicted in Figure 4.26. The initial multiplexer selects the source operand depending on the *xs* value, which is then routed to the appropriate conversion block if needed. The result obtained is written back to the destination register bank decided by *xd*.

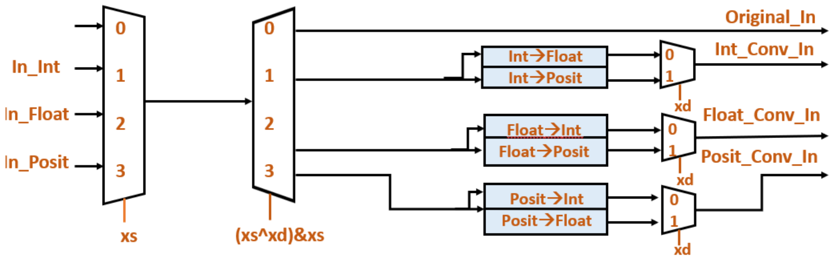


Figure 4.26: DTC block architecture

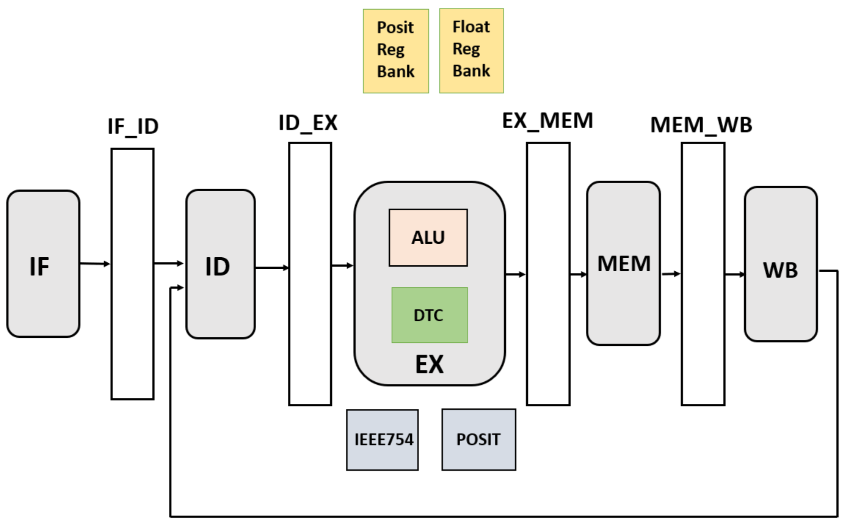


Figure 4.27: DTC block integrated RISC-V core

Integration of DTC block to the RISC-V core

The DTC block integrated *RV32IMF_XPosit* is presented in Figure 4.27. Similar to the MOT block, the DTC also demands the separation of the Posit and float register banks from their respective functional units. This meets the fourth objective of developing hardware to handle mixed operand types namely Posit, integer and floats.

4.5 Reconfigurable posit unit

The Posit unit developed is further enhanced to support multiple es values. This feature in-fact fully leverages the capability of a Posit number to offer better dynamic range and precision for higher and lower es values respectively. The possibility of dynamic switching between different es values at run-time will improve the acceptability of the Posit arithmetic in many relevant applications. Here, a parametrized reconfigurable Posit unit is designed to support $es = 2$ and $es = 3$ for a fixed Posit word size of $N = 32$-bits.

This implementation requires minimal modification in the decoder, encoder and the arithmetic units. The changes in the units within the Posit block are as described. The two es values are declared as parameters as $es\_A = 2$ and $es\_B = 3$. The es is obtained as input, which can be either 2 or 3. The register widths of the exponent and the fraction fields fed as the output of the decoder is designed for maximum width requirement among the possible es values. The exponent register is defined with respect to the maximum es value, ie. es_B. The MSB is sign extended when es is es_A. However, in case of the fraction field, the register width is defined with respect to the smallest exponent value (es_A) since the width of the register is bound as $[word\_size - es - 3 : 0]$. For $N = 32$-bits, the exponent and the fraction registers are depicted as Figure 4.28 and 4.29 respectively. The sign extended bits are represented in blue in the figures.

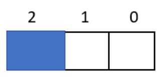


Figure 4.28: Exponent register in reconfigurable Posit unit

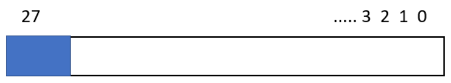


Figure 4.29: Fraction register in reconfigurable Posit unit

Similarly, the exponent and the fraction field registers within the arithmetic units (adder/subtractor, multiplier and the division block) are also designed with maximum register width to accommodate multiple *es* values. Furthermore, corresponding modifications have been incorporated in the encoder stage of the Posit unit. This meets objective seven of developing a reconfigurable Posit unit capable of switching between the maximum exponent size (*es*) at runtime.

4.5.1 Simulation and resource utilization

The simulation of reconfigurable Posit unit is as depicted in Figure 4.30. The simulation represents the Posit result for two separate *es* values (two and three). The operation performed is that of Posit addition between two register operands.

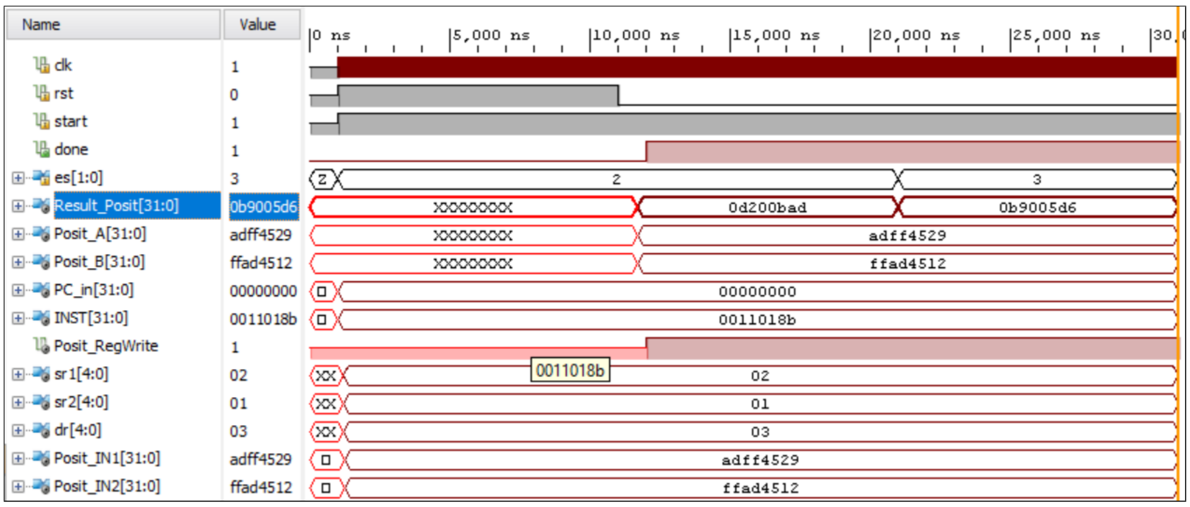


Figure 4.30: Simulation of reconfigurable Posit unit

Posit module	**LUT** ($es = 2$)	**LUT** ($es = 3$)	**LUT** ($es = 2, 3$)
Decoder	399	402	410
Arithmetic Block	1468	1485	1608
Encoder	101	103	104

Table 4.8: LUT comparison between Posit ($es = 2$), Posit ($es = 3$) and Posit ($es = 2, 3$)

Posit module	**FF** ($es = 2$)	**FF** ($es = 3$)	**FF** ($es = 2, 3$)
Decoder	56	56	58
Arithmetic Block	72	75	77
Encoder	32	33	35

Table 4.9: Flip-Flop comparison between Posit ($es = 2$), Posit ($es = 3$) and Posit ($es = 2, 3$)

The resource utilization for the reconfigurable Posit unit ($es = 2, 3$) and that of non-reconfigurable unit ($es = 2$), ($es = 3$) is shown in Table 4.8 and 4.9. It is observed that the reconfigurable unit requires nearly 8% more LUTs and 6% more FFs as opposed to the non-reconfigurable Posit block with $es = 2$. It is to be noted that the previous works [4] of incorporating dynamic switching into the Posit block takes around 15% more LUTs and 8% more FFs. Hence, a saving of 7% and 2% in LUTs and FFs respectively are achieved using the proposed approach of implementing reconfigurability.

CHAPTER 5

IMPLEMENTATION RESULTS

Verilog HDL and RTL coding was used to implement the proposed hardware. The code was implemented on a commercial grade Xilinx ARTIX-7 FPGA (xc7a100tcsg324-1) using the Xilinx Vivado tool while, capable of operating at 100MHz. The results include comparison of the resource utilization and timing requirements of various Posit integration and data type casting approaches. The processor developed works only in user mode. The instructions supported are presented in Appendix A .

5.1 Area and speed comparison of various posit extended RISC-V cores

The Table 5.1 illustrates the clock cycles per instruction for various Posit extended RISC-V cores differing in terms of Posit integration method and co-existence with the IEEE-754 standard. It is found that Posit as an accelerator consumes two additional clock cycles compared to the proposed approach wherein, Posit arithmetic is integrated as a tightly coupled unit along with its floating-point counterpart. Therefore the novel approach is capable of overcoming both the drawbacks of the existing approaches and can therefore be considered as the most effective method of Posit integration to the processor. The Table 5.2 illustrates the resource utilization for the three approaches (4.3.1, 4.3.2 and 4.3.3) of Posit integration as discussed in Section 4.3. It can be noted that though approach 4.3.1 utilizes lesser resources compared to both approaches 4.3.2 and 4.3.3 of Posit integration, F extension is fully replaced in this case. Hence, among the remaining approaches, the proposed approach (4.3.3) is preferred over the accelerator (4.3.2) in terms of the area requirement. This meets the third objective of presenting a comparative study of Posit integration within the core pipeline and as an accelerator in terms of resource utilization and timing requirements.

Posit Integration	**Co-existence with floating-point Arithmetic**	**Clock Cycles/ Instruction**
Tightly Coupled	No	8
Accelerator	Yes	12
Tightly Coupled	Yes	10

Table 5.1: Timing requirement analysis of various Posit integration approaches

Posit Integration	**Co-existance with floating-point Arithmetic**	**LUT Count**	**FF Count**
Tightly Coupled	No	1535	811
Accelerator	Yes	3931	1304
Tightly Coupled	Yes	3642	1103

Table 5.2: Resource utilization analysis of various Posit integration approaches

5.2 Resource utilization and timing requirement analysis of data type casting approaches

The performance analysis of the two data type casting approaches are undertaken. The comparison of the MOT and the DTC block integrated Posit enhanced RISC-V core in terms of area is as presented in Table 5.3. It is found that the DTC block approach occupies 31% and 14% lesser LUTs and FFs respectively compared to the MOT approach.

Data type casting approach	**LUT**	**LUTRAM**	**FF**	**DSP**	**BUFG**
MOT	10448	128	1183	15	12
DTC	7185	110	1009	12	4

Table 5.3: Resource utilization analysis of the data type casting approaches

Operation	**Destination Data Type**	**MOT**	**DTC**
Addition	Float	0.18	0.17
	Posit	0.28	0.18
	Integer	0.2	0.31
Multiplication	Float	0.27	0.26
	Posit	0.2	0.18
	Integer	0.10	0.38
Division	Float	0.28	0.21
	Posit	0.24	0.22
	Integer	0.28	0.38

Table 5.4: Timing requirement analysis of the data type casting approaches (in μs)

The timing requirements (in μs) of the data type casting approaches for basic arithmetic operations involving mixed operand types are as presented in Table 5.4. The table is a subset of all the possible arithmetic operations involving various data types and only includes instructions handling all the three data types. For instance, the first row of the table indicates that the MOT block approach can perform an arithmetic operation such as $F6 = P2 + I5$ in 0.1805μs. However, the DTC block takes only 0.1715μs for the same instruction. The time taken by the DTC block approach is the sum of the time taken for the conversion of all the source data types to that of the destination type and finally to carry out the arithmetic operation. The time taken for conversion between different data types are tabulated in Table 5.5. It is inferred that in terms of the time taken for an operation, the DTC block approach of data type casting is always preferred over the MOT method except when the destination data type is that of an integer. This satisfies the sixth objective of presenting a comparative study of the data type casting approaches put forth.

Source Data Type	**Destination Data Type**	**Conversion Time**
Integer	Float	0.03
Integer	Posit	0.06
Float	Integer	0.1
Float	Posit	0.02
Posit	Integer	0.14
Posit	Float	0.05

Table 5.5: Time taken for conversion between data types (in μs)

5.3 Power summary

The distribution of the power dissipation is as tabulated in Table 5.6. The implementation resulted in a dynamic power dissipation of 0.225W and static power dissipation of 0.098W. The power dissipation is analyzed by loading the set of instructions supported by the Posit enhanced RISC-V processor listed in the appendix section.

Total On-Chip Power (W)	0.323
Dynamic (W)	0.225
Device Static (W)	0.098

Table 5.6: Power analysis

CHAPTER 6

CONCLUSION & FUTURE DIRECTIONS

6.1 Conclusion

Posit is a novel arithmetic that is capable of surmounting the shortcomings of the widely used IEEE-754 standard. A fully parametrized Posit arithmetic core generator for basic arithmetic computations such as addition/subtraction, multiplication and division is developed. Posit arithmetic caters to the varying requirement of dynamic range and precision in significant domains. An analysis of the favorable $< N, es >$ for applications such as machine learning, audio processing and big data analytics are performed. The commendable benefits obtained through Posit utilization in these domains need to be extended to other applications too.

The RISC-V, an open source ISA is enhanced by supporting Posit arithmetic. Various approaches for the Posit implementation in the ISA is proposed. A comparative study of the approaches in terms of timing and resource utilization are also undertaken. The proposed method of Posit integration to the RISC-V core is better in terms of speed compared to the existing accelerator approach. It also allows co-existence of the floating-point and Posit arithmetic unlike when Posit is implemented as an execution unit within the core by replacing the FPU. Furthermore, addition of features that complement compatibility are vital for the acceptability of the Posit extended RISC-V core in various applications. To that extend, hardware unit to implement data type casting is incorporated into the processor. This complements the concurrent use of integer, floating and Posit arithmetic within the processor.

Data type casting block allows the conversion of the floating-point operands into Posit which will in turn allow easy porting to the new arithmetic. Moreover, this feature eliminates the need for rewriting the existing code with the floating-point arithmetic. In addition to that, this hardware innovation supports instructions for carrying out arithmetic operations involving mixed operand types. Different approaches for data type conversion are put forth and compared based on the resource utilization and the timing requirements for few mixed operand instructions. It is inferred that the selection of the data type casting approach depends on the destination data type of the operand. In terms of the timing requirements, the DTC block is preferred to the MOT block except for arithmetic instructions involving integer destination operand. However, the DTC block integrated RISC-V core occupies significantly lesser resources compared to the MOT block integrated core.

Furthermore, the Posit unit is enhanced to facilitate dynamic switching between two different es values with minimal overheads. Since this add-on allows easy transition from higher dynamic range mode to that of higher precision mode, the usage of the Posit arithmetic is likely to be promoted over the floating-point in multiple domains. The enhancement is brought about by configuring the register widths of the exponent and the fraction fields to accommodate both the es values. An increase in area utilization of nearly 8% in LUTs and 6% in FFs are observed in comparison with the non-reconfigurable Posit block with $es = 2$.

6.2 Future directions

The existing design can be further extended in the following ways:

1. The Posit unit be further reconfigured to accommodate many es values, which can improve the acceptability of the arithmetic.
2. The Posit enhanced processor can be used in disciplines such as defence, remote sensing etc. and performance be evaluated.

REFERENCES

[1] Jaiswal, M. K. and So, H. K. (2019). “Pacogen: A hardware Posit arithmetic core generator.” *IEEE Access*, 7, 74586–74601.

[2] Sarkar, S., Velayuthan, P., and Gomony, M. (2019). “A Reconfigurable Architecture for Posit Arithmetic.” *Euromicro Conference on Digital System Design (DSD)*, Kallithea, Greece, 82-87.

[3] Chaurasiya, R., Gustafson, J., Shrestha, R., Neudorfer, J., Nambiar, S., Niyogi, K., Merchant, F. and Leupers, R. (2018). “Parametrized Posit arithmetic hardware generator.” *IEEE International Conference on Computer Design (ICCD)*, Orlando, FL, USA, 334–341.

[4] Tiwari, S., Gala, N., Rebeiro, C. and Kamakoti, V. (2019). “PERI: A Posit Enabled RISC-V Core.” *arXiv:1908.01466v1 [cs.AR]*, `https://doi.org/10.48550/arXiv.1908.01466` (Nov. 24, 2021).

[5] Arunkumar, M. V., Bhairathi, S. G. and Hayatnagarkar, H.G. (2020). “PERC: Posit Enhanced Rocket Chip.” *Workshop on Computer Architecture Research with RISC-V at International Symposium on Computer Architecture*, Valencia, Spain, 1-8.

[6] Hou, J., Zhu, Y., Du, S., and Song, S., (2019). “Enhancing Accuracy and Dynamic Range of Scientific Data Analytics by Implementing Posit Arithmetic on FPGA.” *Journal of Signal Processing Systems*, 91, 1137–1148.

[7] Jain, R., Sharma, N., Merchant, F., Patkar, S. and Leupers, R. (2020). “CLARINET: A RISC-V Based Framework for Posit Arithmetic Empiricism.” *arXiv:2006.00364v3 [cs.AR]*, `https://doi.org/10.48550/arXiv.2006.00364` (Nov. 2, 2021).

[8] Mian, R., Shintani, M. and Inoue, M. (2019). “Cycle-Accurate Evaluation of Software-Hardware Co-Design of Decimal Computation in RISC-V Ecosystem”, *IEEE International System-on-Chip Conference (SOCC)*, Las Vegas, Nevada, USA, 412-417.

[9] Cococcioni, M., Ruffaldi,E. and Saponara, S. (2018). “Exploiting Posit Arithmetic for Deep Neural Networks in Autonomous Driving Applications”, *International Conference of Electrical and Electronic Technologies for Automotive*, Milan, Italy, 1-6.

[10] Carmichael, Z, Langroudi, H. F., Khazanov, C., Lillie, J., Gustafson, J. L. and Kudithipudi, D. (2019). "Deep Positron: A Deep Neural Network Using the Posit Number System," *Design, Automation and Test in Europe (DATE)*, Florence, Italy, 1421-1426.

[11] Cococcioni, M., Rossi, F., Ruffaldi, E. and Saponara, S (2021). "Vectorizing Posit operations on RISC-V for faster deep neural networks: experiments and comparison with ARM SVE" *Neural Computation and Applications*, 33, 10575–10585.

[12] Klöwer, M., Düben, P. and Palmer, T. (2019). "Posit numbers as an alternative to floating-point numbers for weather and climate models", *Conference for Next Generation Arithmetic*, Singapore, 1-8.

[13] Gustafson J. and Yonemoto I. (2017). "Beating floating point at its own game: Posit arithmetic." *Supercomputing Frontiers and Innovations: an International Journal*, 4(2), 71–86.

Appendix A

RV32IMF_ XPosit Processor

This chapter presents the instructions supported by *RV32IM_XPosit* processor and the simulation results of various instruction types.

A.1 *RV32IMF_ XPosit* processor instruction set

The *RV32IMF_XPosit* processor is developed to support a subset of instructions within standard extensions such as Integer (I), Integer Multiplication and Division (M), floating-point (F) and non-standard extensions such as Posit, MOB or DTC extension. The *RV32IMF_XMOB_XPosit* and *RV32IMF_XDTC_XPosit* processor is designed to operate only in user mode. The standard instructions are as listed in Table A.1. The non-standard instructions for Posit, MOT and the DTC approach are listed separately in Table A.2, A.3 and A.4 respectively.

Extension	**Function**	**31 25**	**24 20**	**19 15**	**14 12**	**11 7**	**6 0**
I	ADD	0000000	rs2	rs1	000	rd	0110011
I	SUB	0100000	rs2	rs1	000	rd	0110011
I	OR	0000000	rs2	rs1	110	rd	0110011
I	AND	0000000	rs2	rs1	111	rd	0110011
I	XOR	0000000	rs2	rs1	100	rd	0110011
M	MUL	0000001	rs2	rs1	000	rd	0110011
M	DIV	0000001	rs2	rs1	100	rd	0110011
I	ADDI	Imm[11:0]		rs1	000	rd	0010011
I	XORI	Imm[11:0]		rs1	100	rd	0010011
I	LW	Imm[11:0]		rs1	010	rd	0000011
I	SW	Imm[11:5]	rs2	rs1	010	Imm[4:0]	0100011
I	BEQ	Imm[12|10:5]	rs2	rs1	000	Imm[4:1|11]	1100011
I	BNE	Imm[12|10:5]	rs2	rs1	001	Imm[4:1|11]	1100011
F	FADD.S	0000000	rs2	rs1	rm	rd	1010011
F	FMUL.S	0001000	rs2	rs1	rm	rd	1010011
F	FDIV.S	0001100	rs2	rs1	rm	rd	1010011

Table A.1: Standard *RV32IMF* instructions

Extension	Function	31 25	24 20	19 15	14 12	11 7	6 0
Posit	PADD.S	0000000	rs2	rs1	000	rd	0001011
Posit	PMUL.S	0100000	rs2	rs1	000	rd	0001011
Posit	PDIV.S	0000000	rs2	rs1	110	rd	0001011

Table A.2: Non-standard Posit instructions

Extension	Function	31 29	28 25	24 20	19 15	14 12	11 7	6 0
MOB	ADD	xd[1]xs1[1]xs2[1]	0000	rs2	rs1	xd[0]xs1[0]xs2[0]	rd	0101011
MOB	MUL	xd[1]xs1[1]xs2[1]	1000	rs2	rs1	xd[0]xs1[0]xs2[0]	rd	0101011
MOB	DIV	xd[1]xs1[1]xs2[1]	1100	rs2	rs1	xd[0]xs1[0]xs2[0]	rd	0101011

Table A.3: Non-standard MOB instructions

Extension	Function	31 24	23 20	19 15	14 12	11 7	6 0
DTC	Data Conversion	00000000	xd[1]xd[0xs[1]xs[0]	rs	000	rd	0101011

Table A.4: Non-standard DTC instruction

A.2 Simulation results

A.2.1 Stand-alone arithmetic instructions

Integer instructions

Instruction No	Operation Implemented	Machine Instruction
1	ADDI R1, R0, 10	00A00093
2	ADD R4,R1,R2	00208233
3	SUB R6,R5,R1	40128333
4	OR R7,R3,R2	0021E3b3
5	XORI R10,R3,49	0311C513

Table A.5: Integer stand-alone test instructions

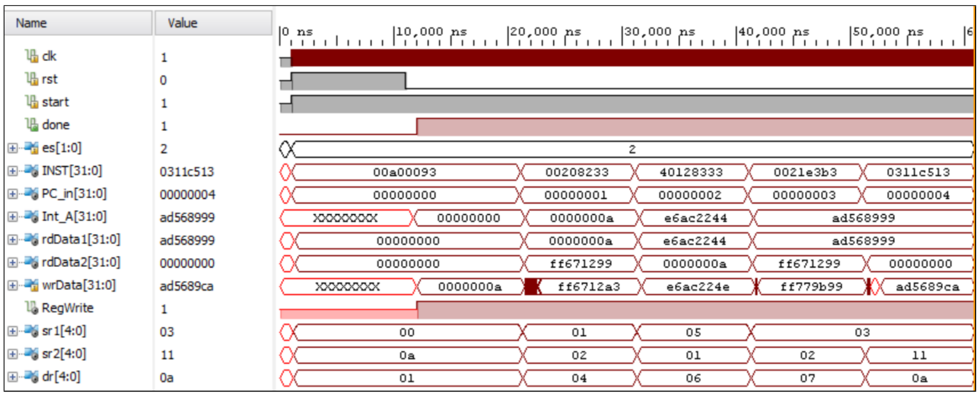

Name
Value
0 ns
10,000 ns
20,000 ns
30,000 ns
40,000 ns
50,000 ns
clk 1
rst 0
start 1
done 1
es[1:0] 2 — 2
INST[31:0] 0311c513 — 00a00093 00208233 40128333 0021e3b3 0311c513
PC_in[31:0] 00000004 — 00000000 00000001 00000002 00000003 00000004
Int_A[31:0] ad568999 — XXXXXXXX 00000000 0000000a e6ac2244 ad568999
rdData1[31:0] ad568999 — 00000000 0000000a e6ac2244 ad568999
rdData2[31:0] 00000000 — 00000000 ff671299 0000000a ff671299 00000000
wrData[31:0] ad5689ca — XXXXXXXX 0000000a ff6712a3 e6ac224e ff779b99 ad5689ca
RegWrite 1
sr1[4:0] 03 — 00 01 05 03
sr2[4:0] 11 — 0a 02 01 02 11
dr[4:0] 0a — 01 04 06 07 0a

Figure A.1: Simulation of integer stand-alone test instructions

Floating-point instructions

Instruction No	Operation Implemented	Machine Instruction
1	FADD.S R3,R2,R1	001101D3
2	FMUL.S R3,R2,R1	101101D3
3	FDIV.S R3,R2,R1	181101D3

Table A.6: Floating-point stand-alone test instructions

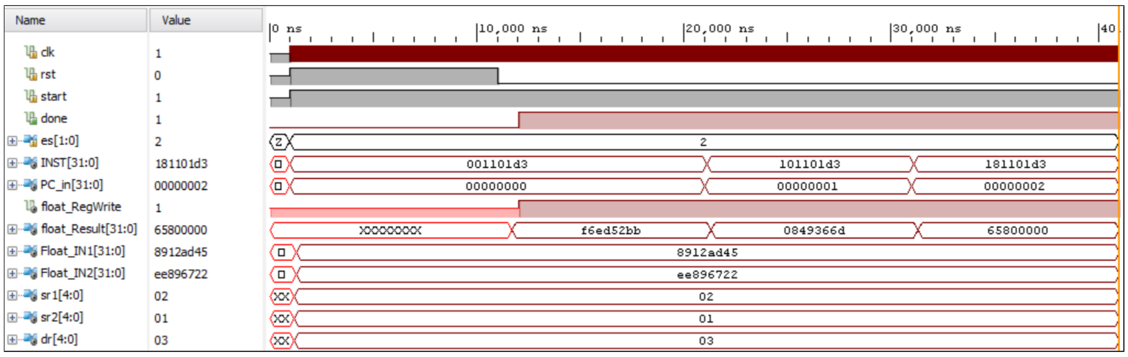


Figure A.2: Simulation of floating-point stand-alone test instructions

Posit instructions

Instruction No	Operation Implemented	Machine Instruction
1	PADD PR3,PR2,PR1	0011018B
2	PMUL PR5,PR2,PR1	1011028B
3	PDIV PR7,PR2,PR1	1811038B

Table A.7: Posit stand-alone test instructions

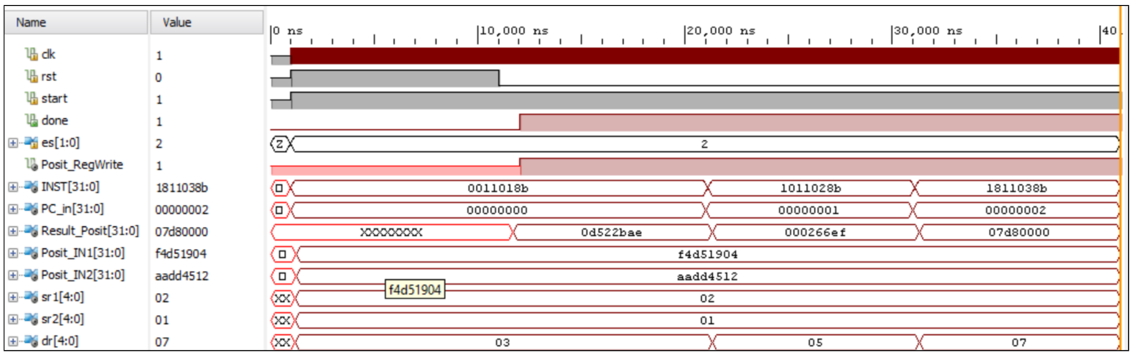


Figure A.3: Simulation of Posit stand-alone test instructions

A.2.2 Mixed arithmetic instructions

Mixed arithmetic addition instruction

Instruction No	Operation Implemented	Machine Instruction
1	$F6 = P2 + I5$	C051332B
2	$P6 = F2 + I5$	C051532B
3	$I6 = P2 + F5$	6051632B

Table A.8: Mixed addition test instructions

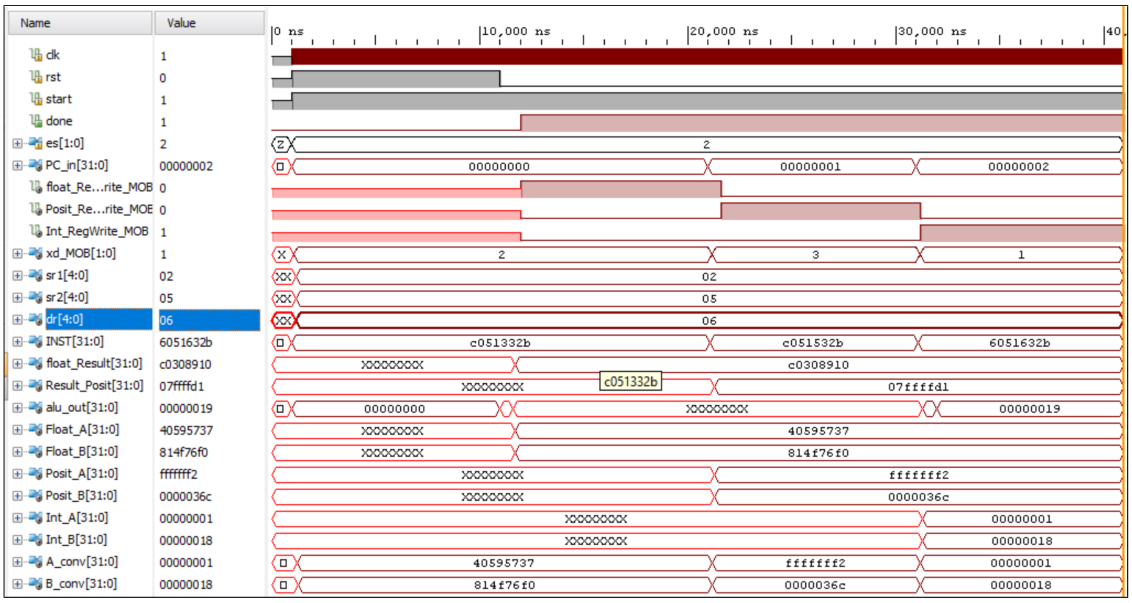


Figure A.4: Simulation of mixed arithmetic addition test instructions

Mixed arithmetic multiplication instruction

Instruction No	Operation Implemented	Machine Instruction
1	$F6 = P2 * I5$	D051332B
2	$P6 = F2 * I5$	D051532B
3	$I6 = P2 * F5$	7051632B

Table A.9: Mixed arithmetic multiplication test instructions

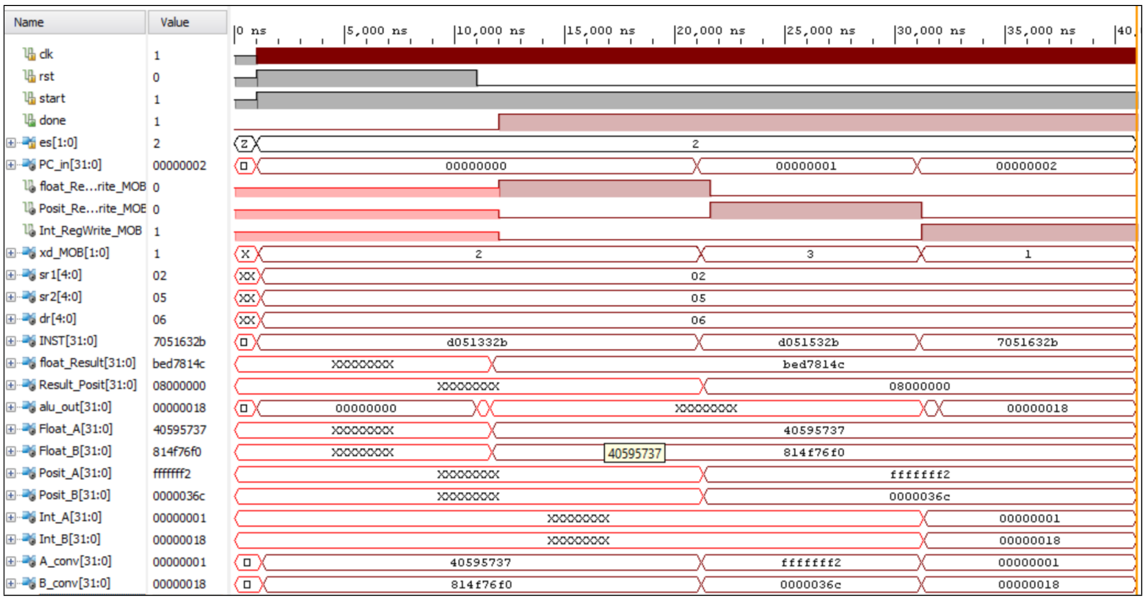


Figure A.5: Simulation of mixed multiplication test instructions

Mixed arithmetic division instructions

Instruction No	Operation Implemented	Machine Instruction
1	$F6 = P2/I5$	D851332B
2	$P6 = F2/I5$	D851532B
3	$I6 = P2/F5$	7851532B

Table A.10: Mixed arithmetic division test instructions

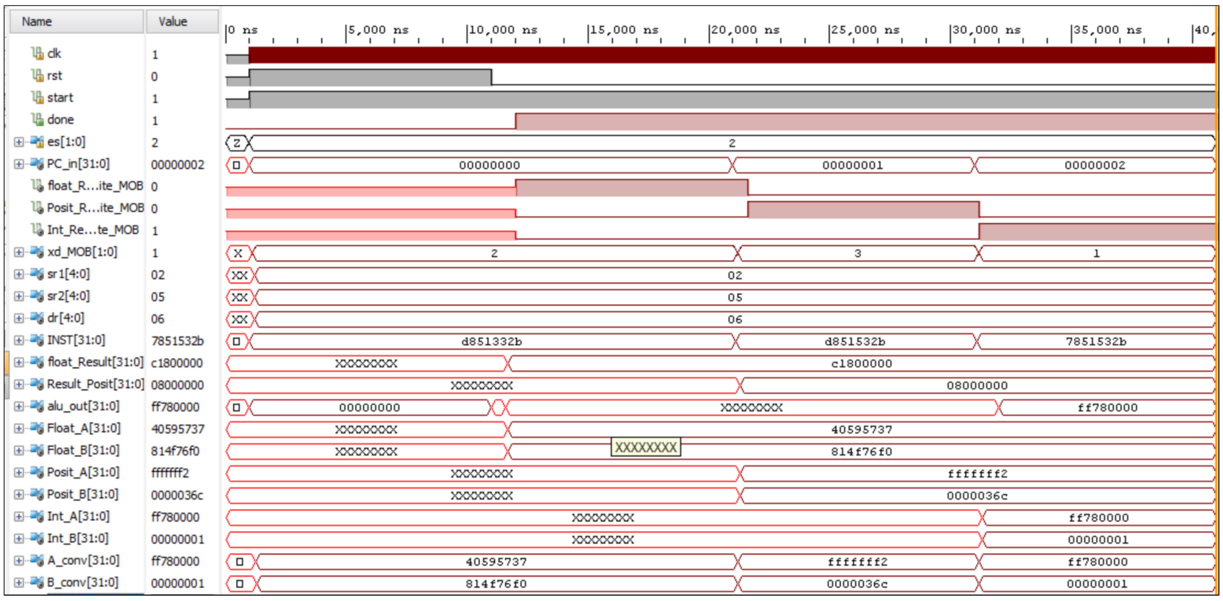


Figure A.6: Simulation of mixed arithmetic division test instructions

Publication based on the thesis

Ashley Kurian and Ramesh Kini M., "Posit Extended RISC-V Processor and its Enhancement Using Data Type Casting", 3rd International Conference on Data Science and Applications (ICDSA 2022), March 26-27.