



CSD-Driven Speedup in RISC-V Processor

Farhad Ebrahimiandaryani and Dietmar Fey

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

February 3, 2025

CSD-Driven Speedup in RISC-V Processor

Farhad EbrahimiAzandaryani and Dietmar Fey

Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, 91058, Germany
{farhad.ebrahimiazandaryani,dietmar.fey}@fau.de,


Abstract. This paper introduces a synthesizable μ -architectural design method to boost the performance of a given RISC-V processor architecture by utilizing Canonical Signed Digit (CSD) representation during the execution stage within the processor pipeline. CSD is a unique ternary number system that enables carry/borrow-free addition/subtraction in constant time $O(1)$ regardless of word length N . The CSD extension is exemplarily demonstrated to the Potato processor, a simple RISC-V implementation for FPGAs. However, the method can also be applied to other implementations in principle. Our performance boost due to the CSD requires an overhead for conversion between binary and CSD representation. This overhead is compensated by an extension to a seven-stage pipeline architecture, featuring a three-step execution stage that increases the throughput and the operating frequency and enables loop unrolling, which is especially advantageous in applications with consecutive calculations, e.g., signal processing. By experimental results, we compared our CSD-based ternary solution to the original implementation, which utilizes the usual pure binary number representation of the operands. Our approach achieved a 2.41X increase in operating frequency over the original RISC-V processor on FPGA, with over 20% of this gain attributed to the CSD encoding. This enhancement resulted in up to a 2.40X improvement in throughput and a 2.37X reduction in execution time for computation-intensive benchmark applications.

Keywords: Ternary · Canonical Signed Digit · FPGA · RISC-V.

1 Introduction

The ongoing demand for more computing power drives innovation in processor technology, particularly in optimizing arithmetic operations, as they directly influence overall performance. A key example is multiplication, a frequently used operation in fields like digital signal processing and machine learning, where tasks often involve consecutive multiplication operations, such as filtering. Although numerous algorithms have been developed to enhance binary multiplication, most are still limited by carry chain delays in partial product summation, reducing their efficiency. From the 1960s to the 1990s, Avizienis [1], Parhami [2] and others were already researching carry-free summation in ternary encoding that worked in $O(1)$, expediting operations like multiplication from $O((\log N)^2)$ to $O(\log N)$. However, adopting this encoding comes with trade-offs: it abandons

uniform number representation, introduces challenges in comparison operations, and increases memory requirements, as three states instead of two must be stored per digit. Nonetheless, CSD representation allows a unique number of representations of ternary operands and despite its potential, it has not been widely integrated into standard processor architecture. In this work, we integrate this encoding in the execution stage of a RISC-V processor and show performance increment is feasible.

This paper proposes a μ -architectural design method employing CSD representation during execution to enhance RISC-V processors' performance. Instead of using binary digits, CSD operates with balanced ternary digits $[-1, 0, +1]$, enabling carry/borrow-free operations and faster computation. The architectural shift focuses on transitioning the processor's execution stage from binary to ternary components while keeping the ISA intact. NOVA , the name we gave our processor, implements the introduced method supporting RV32IZmmul_Zicsr¹ ISA, which includes integer multiplication instruction but omits hardware-based division due to its low computational frequency. This trade-off optimizes the processor for compute-intensive applications like deep learning, where multiplication is more common. The key contributions of this paper can be summarized as follows:

- **CSD Integration:** This is the first FPGA implementation of a RISC-V processor that supports "M" ISA extension using CSD-operands in its μ -Architecture. The employed encoding features minimal Hamming weight, indicating fewer non-zero digits and reducing the number of partial products. This lowers the computation effort required for multiplication, ultimately improving performance and the latency of the execution unit in the processor.
- **Three-Step Execution:** As a technical contribution, NOVA integrates an optimized, balanced pipeline architecture in the execution stage by tripling the Instruction Execution (IE1, IE2, IE3) stage, distributing multiplication among them to increase performance. This architectural choice enables loop unrolling leading to lower branch overhead and stall cycles.

The rest of the paper is organized as follows: Section 2 provides an overview of Signed Digit (SD) representation, with a specific emphasis on CSD representation. Section 3 presents an extensive review of the current state-of-the-art techniques in applying SD and CSD in the digital circuit design field. Section 4 provides a comprehensive overview of the method and its implementation within an open-source RISC-V processor. Furthermore, Section 5 addresses the technical setup, tools, and benchmarks where NOVA is deployed, along with a detailed presentation of the FPGA-based evaluation results. Finally, Section 6 concludes the paper by summarizing the key findings and contributions of this research work.

¹ The RV32IZmmul_Zicsr ISA provides a RISC-V processor with a base set of integer arithmetic instructions (RV32I), Multiplication extension excluding division (Zmmul), and instructions for manipulating control and status registers (Zicsr).

2 Preliminary

In this section, a concise overview of the key concepts associated with SD representation, specifically the CSD representation and its properties, is presented as follows.

2.1 SD representation

Researchers have investigated the utilization of SD representation as an alternative approach to address the limitations associated with the conventional binary representation[1][2]. According to a definition in [2], a generalized SD number system to a radix r utilizes a digit set $S = \{-\alpha, -(\alpha - 1), \dots, -1, 0, 1, \dots, \beta\}$, with $\alpha, \beta > 0$ and $\alpha + \beta + 1 > r$ and $r > 1$. E.g., for the so-called balanced Binary Signed Digit (BSD) system holds, that the digit set is symmetric ($\alpha = \beta = 1$), and for the radix holds, $r = 2$, which yields the digit set $S = \{-1, 0, +1\}$. SD representation's carry/borrow-free addition/subtraction property accelerates arithmetic operations and facilitates efficient computation in various applications [2]. Fig.1 illustrates the detailed design of a 32-digit adder/subtractor for two provided SD numbers², $x = x_{31}^n x_{31}^p x_{30}^n x_{30}^p \dots x_1^n x_1^p x_0^n x_0^p$ and $y = y_{31}^n y_{31}^p y_{30}^n y_{30}^p \dots y_1^n y_1^p y_0^n y_0^p$. Notably, the critical path has been constrained to two adjacent digits, regardless of the input data word length. Nonetheless, regular SD representation has drawbacks: it lacks uniqueness, allowing multiple representations of the same number that complicates comparisons. Additionally, it possesses a non-minimal Hamming weight, leading to more effort in result generation (e.g., more partial products in multiplication), and reducing performance improvements.

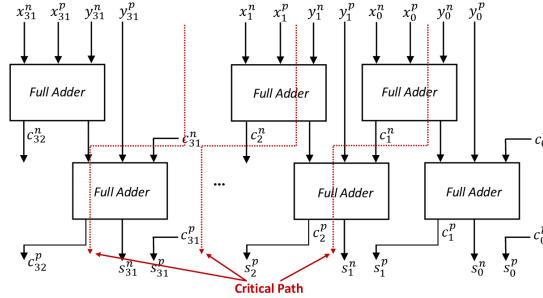


Fig. 1. 32-digit SD/CSD Adder/Subtractor

² $x = (x_i^n x_i^p)$ is to interpret as follows. In a strict digital system a ternary digit requires two binary values, e.g. x_i^n , and x_i^p , called plus-minus coding. It codes a ternary digit x as follows $x = x_i^p - x_i^n$, e.g. (1,0) is equal to -1 .

2.2 CSD representation

CSD representation is a specific form of radix-2 SD representation that possesses distinct properties as follows [3]:

- Adjacent CSD digits are never both non-zero, meaning $|d_i \cdot d_{i-1}| \neq 1$, where $1 \leq i \leq N - 1$, and N is the number of digits.
- It exhibits the minimum number of non-zero digits among various representations. Consequently, the probability of a CSD digit denoted as d_i being non-zero is determined by:

$$P(|d_i| = 1) = \frac{1}{3} \left(1 + \frac{1}{4N} \right)$$

- Utilizing the first property guarantees that each arbitrary number has a unique and exclusive encoding ensuring the distinct representation of numbers.

The first two properties of CSD representation show that as the number of digits N increases, the probability of a non-zero digit approaches $\frac{1}{3}$, reducing the Hamming weight and capping the number of non-zero partial products to $\frac{N}{2}$ in multiplications. In contrast, binary or regular SD representations have a fixed probability of $\frac{1}{2}$, potentially producing up to N non-zero partial products. Additionally, the third property allows for simple, digit-wise comparisons within the ternary encoding, addressing the non-uniqueness issue present in regular SD representation (see [4]).

2.3 Encoding

A radix-2 CSD representation is a redundant expression of binary numbers that allows for more than 2 values per digit. In other words, each digit d_i should be encoded using two bits ($r = 2$). In practice, Table 1 showcases the two most commonly utilized encodings [5]. Encoding 1 can be characterized as a sign-value representation that adheres to the following relation:

$$d_i = (-1)^{d_i^s} \times d_i^v \quad (1)$$

where d_i^s and d_i^v represent the sign bit and the value bit, respectively. To determine the encoding that corresponds to each digit in the first scheme, Eq. (1) provides the corresponding computation. Encoding 2 called Positive-Negative, allows an additional valid representation of 0 when $d_i^p = 1$ and $d_i^n = 1$, which is useful in some arithmetic implementations and is used in this research. The corresponding encoding of each digit in the second scheme can be achieved by employing Eq. (2).

$$d_i = d_i^p - d_i^n \quad (2)$$

Table 1. Two common encodings used in the binary representation of a CSD digit

| SD/CSD digit | Encoding 1 | | Encoding 2 | |
|--------------|------------|---------|------------|---------|
| d_i | d_i^s | d_i^v | d_i^p | d_i^n |
| -1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |

3 Prior works

This section thoroughly explores the latest techniques and designs related to SD and CSD representation. By examining these approaches, it aims to provide a detailed understanding of the current advancements in utilizing these representations.

Recent research [4] has shown that switching from conventional binary number representation to alternative schemes can significantly improve processor arithmetic logic unit performance. This is due to the reduced logic depth in these systems, which can surpass the speed of traditional binary circuits. However, a key limitation is the lack of unique representations for integers, requiring the processor to revert to binary for tasks like comparisons, where distinct number representation is necessary.

The work in [6] focuses on a ternary processor leveraging Carbon Nanotube Field Effect Transistor (CNTFET)-based ternary logic for the control unit and memory. This approach demonstrates performance improvement and power efficiency compared to traditional binary systems. However, it faces challenges at the circuit level, particularly with carry chain propagation. Unlike our method, which uses base-2 with CMOS technology and benefits from established infrastructure, their design employs a base-3 system and does not adopt a redundant number representation, complicating effective carry chain management and limiting optimization potential for processor performance.

The work presented in [7] introduces a comprehensive design and verification framework for developing a fully functional emerging ternary processor utilizing balanced ternary number representation based on ternary logic. While arithmetic operations like addition and subtraction show improved efficiency in ternary logic, more complex operations such as multiplication present additional challenges compared to binary counterparts. The results primarily remain conceptual, lacking concrete implementation details. In contrast, our approach is grounded in a binary logic system that operates on ternary-represented numbers as data operands. In the work by [8], CSD encoding has been leveraged in the development of a Neural Network-based image compression architecture. While effectively deployed on FPGA platforms for grayscale image processing, the investigation underscores the unexplored domain of extending this methodology

to color image compression, attributed to the intricate complexities associated with the employed encoding scheme.

Researchers in [9][10] have explored the use of CSD-encoded coefficients in FIR filter designs, highlighting several key benefits. Specifically, CSD reduces the number of additions required during multiplication, a central operation in FIR filters, resulting in lower computational complexity and improved numerical stability, especially in FPGA implementations. However, while non-binary digits boost filter performance, they can introduce complexity and increase implementation costs, which is particularly problematic for smaller-sized FIR filters where the overhead might outweigh the performance gains. These insights into CSD’s efficiency in filter design have inspired its application in the execution stage of general-purpose processors, aiming to tackle computation bottlenecks effectively.

In [11], the authors introduced a ternary representation to improve processor performance and computational efficiency within their proposed architecture. However, this design is constrained by its support for only the base RV32I instruction set, limiting its applicability in AI and signal processing fields where performing additional instructions directly can offer more speed-up compared to the traditional indirect techniques e.g. add/shift loop for multiplication.

While researchers have explored the application of SD/CSD representation across various domains, yielding valuable outcomes, none of the proposed solutions specifically addressed the demands of computation-intensive applications to alleviate bottleneck issues related to binary encoding in processors.

4 Proposed design method in a RISC-V core

This section presents an in-depth overview of the proposed method and its implementation in an open-source RISC-V processor [12] that suffers from a computation bottleneck in the execution stage.

To harness CSD computation benefits initially, integrating a Binary-to-CSD (B2C) circuit is unavoidable. This circuit converts incoming binary values into CSD numbers in IE1, before their utilization in the next steps for arithmetic instructions execution. We are favoring [5] due to its low time complexity which is employed in NOVA architecture for B2C purposes. Refer to [5] for detailed conversion guidelines. The efficient computation achieved by CSD representation is attributed to the constrained carry chain (see Fig.1), limited to two adjacent digits. Subtraction operations benefit from this constraint as the requirement for the negative value of the second operand can be met by individually interchanging the positive and negative bits (y_i^p and y_i^n) for each digit. This approach allows the adder circuit to effectively compute $A + (-B)$ instead of requiring a separate circuit for $A - B$. Consequently, the overall circuit design is simplified, and the computational efficiency of subtraction operations is enhanced. In binary representation, $N \times N$ multiplication produces N non-zero partial products. In contrast, CSD multiplication reduces this number to $N/2$ in the worst case because at least one of every two adjacent digits is always zero, resulting in a zero partial product. A bitwise OR operation on adjacent partial products, as shown

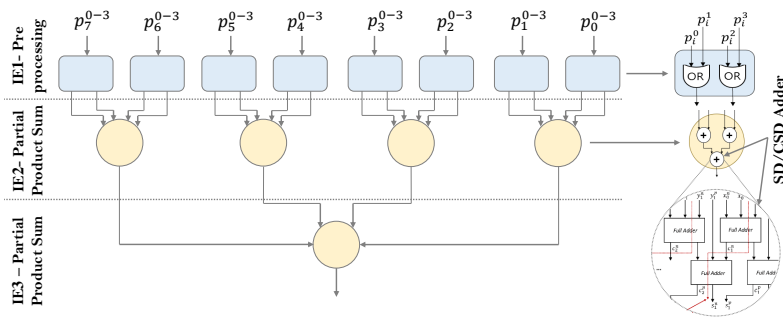


Fig. 2. Pipelined CSD multiplier unit along with internal structure of employed blocks.

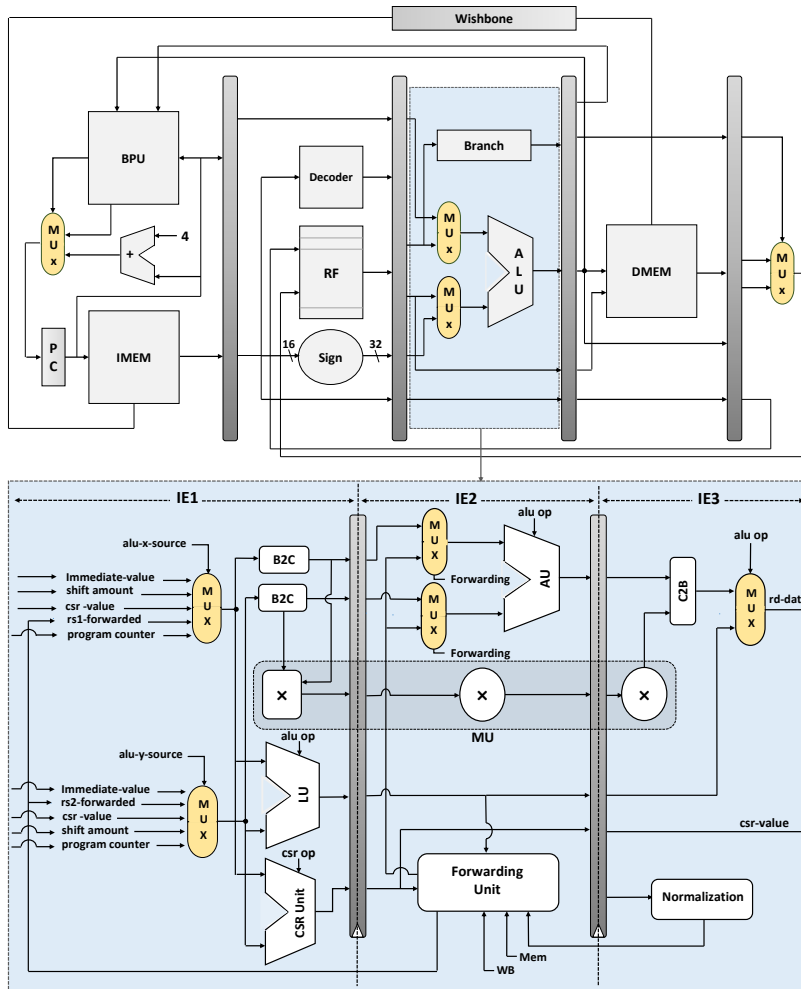


Fig. 3. The overall architecture of NOVA with a detail view of the execution stage.

in Fig. 2, effectively filters out the zero values. This allows only the non-zero products to continue to the next stages of the multiplication tree in IE2 and IE3, enhancing overall performance. Additionally, using SD/CSD adder circuit for partial sum computation improves performance from $O(\log N)^2$ to $O(\log N)$, thereby increasing the efficiency of arithmetic operations. After the execution unit performs the instruction and generates the final result, it becomes necessary to convert the result back into binary representation for storage in memory or register files. The C2B (CSD to binary) conversion is straightforward and can be achieved by decomposing the result into a positive (y^p) and one negative (y^n) vector. This is followed by performing binary summation of y^p and y^n with $C_0 = 1$, where $y^{n'}$ is the one's complement representation of y^n . The output is the binary equivalent of the CSD unit result, excluding the overflow bit in this sequence. All these steps can be executed in $O(\log N)$.

Certainly, the developed arithmetic instructions involve several steps. Initially, the input values are transformed from binary to CSD using the B2C module. Subsequently, the Arithmetic Unit (AU)/Multiplication Unit (MU) executes the instruction using the converted operands. Finally, the outcome is converted to binary encoding using the C2B module, yielding the final result. However, the sequential conversions involved in this process may introduce delays that impact the critical path of the overall operation, potentially resulting in slower performance compared to binary computations and diminishing the advantages of fixed time-frame calculations using CSD. Increasing the number of pipeline stages emerges as a promising solution to address this bottleneck, which is discussed in detail in the next section.

The architecture of NOVA, shown in Fig. 3, consists of the IF, ID, IE (IE1, IE2, IE3), Mem, and WB stages. By dividing the execution stage into three parts, the B2C conversion result can be accessed immediately by the MU for partial product calculation in constant time $O(1)$ during IE1 or registered for use by the AU in IE2. C2B conversion occurs in IE3, reducing conversion delays and maintaining the advantages of CSD representation over binary. The IE1 stage also provides outputs for logical instructions (Shift, XOR, AND, OR) through the Logical Unit (LU), with results available for subsequent instructions via the forwarding unit to prevent data hazards. Meanwhile, the CSR unit manages operations on control and status registers, essential for tasks like system monitoring. Data hazards require careful handling in NOVA's architecture. When an instruction in IE1 needs the result of a previous instruction in IE2/IE3, particularly one involving arithmetic calculations, a stall of one or two cycles is necessary. This stall ensures that operands are accurately retrieved for the computation in IE1, as the required operand will be available later in the Mem stage. However, suppose the output from an Add/Sub instruction in IE2 is used as input for another Add/Sub instruction in subsequent cycles. In that case, it can be forwarded through the forwarding unit without stalling. Caution is needed here because repeatedly forwarding partial results in CSD or SD representation can lead to pseudo-overflow. To mitigate this issue, a normalization unit is essential to address the problem in a fixed time without significantly affecting circuit

performance. As discussed earlier, due to the deeply pipelined execution unit, NOVA can support loop unrolling for both arithmetic and logical instructions. This reduces stall cycles and efficiently fills the 3-step pipeline of the execution stage, resulting in higher throughput. It is important to note that logical operations, such as Shift, AND, OR, and XOR, must be executed using traditional binary representation. This limitation arises from CSD encoding, which primarily optimizes arithmetic instructions but does not enhance logical operations. However, since these logical instructions are not part of the critical path, they do not disrupt the overall functionality of the data path. Consequently, while a different approach is necessary for these operations, their effect on the circuit’s performance and data path functionality is minimal.

5 RESULT AND DISCUSSION

This section details the technical setup and benchmarks used for the FPGA experiments. It presents the results obtained from an open-source RISC-V core [12], the Potato processor (PRISC-V), as the original core tested with various configurations, alongside results from NOVA.

5.1 Technical setup, tools, and benchmarks

In FPGA implementation, the Xilinx Vivado 2023.1 design suite software was used for synthesis, mapping, placement, and routing. Additionally, the Ultra96-v2 (XCZU3EG-1SBVA484I) from the Zynq Ultrascale+ development board family was chosen as the FPGA platform to establish results in all tables and figures in this section. The results provided are specifically derived from the analysis conducted using the specified synthesis tool and FPGA board chosen for the processor’s implementation and may differ when using alternative synthesis tools or FPGA boards, reflecting variations in architecture and optimization strategies. The efficiency evaluation of cores made use of applications from the Mibench suite [13], as well as three well-known applications widely used for processor performance, memory bandwidth examination, and matrix multiplication, namely Dhrystone [14], Schönauer vector triad [15], and Matmul[16]. The chosen applications span various domains, including signal processing and telecommunications, automotive, networking, error-checking, sorting, and arithmetic calculations.

We conducted a thorough evaluation of performance metrics, including throughput, execution time, and resource utilization, to assess computational efficiency in NOVA versus PRISC-V (full binary). Both support the same 32-bit ISA and in-order execution. PRISC-V configurations include a 5-stage pipeline with/out DSP (PRISC-V_{DSP}/PRISC-V), a 7-stage pipeline with/out DSP (PRISC-VII_{DSP}/PRISC-VII) featuring a 3-step execution stage. All five architectures are identical as Fig. 3 except IE stage where PRISC-V configurations do not include conversion circuits and its 7-stage configurations the multiplier is implemented in three steps: IE1 computes four partial products in parallel ($PP_1 = X_{15:0} \times Y_{15:0}$, $PP_2 = X_{31:16} \times Y_{15:0}$, $PP_3 = X_{15:0} \times Y_{31:16}$, and $PP_4 = X_{31:16} \times Y_{31:16}$). IE2 and IE3 add these partial products in a binary tree model to produce the final result.

Table 2. Comparative Logic and Network delay for binary/CSD multipliers

| Article | Net delay (ns) | Logic delay (ns) |
|-----------------------|----------------|------------------|
| BinMul _{DSP} | 3.441 | 4.201 |
| BinMul | 8.132 | 3.027 |
| CSDMul | 8.305 | 2.256 |

5.2 Experimental results

Initially, we analyzed the critical path delays of a 32-bit binary ALU and a 32-digit CSD ALU, both passing through the multiplier module. Table 2 illustrates that the binary multiplier, utilizing DSP (BinMul_{DSP}), features a logic delay of 4.201 ns and a network delay of 3.441 ns. In contrast, the binary multiplier without DSP utilization (BinMul) has a lower logic delay of 3.027 ns but suffers from a higher network delay of 8.132 ns, reducing the efficiency of a non-DSP binary multiplier. Despite an optimized logic delay of 2.256 ns in the CSD multiplier, it has a substantial network delay of 8.305 ns, overshadowing its performance. A more detailed analysis shows that 35% and 48% of the reported logic and network delays in CSD multiplication are due to the conversion circuits. Deeply pipelining the multiplication unit effectively reduces network delay, making designs with lower logic delays well-suited for enhancing performance. As a result, the CSD multiplier stands out as a compelling option.

The frequency values reported in Table 3 represent the maximum frequencies that each core was able to operate with, ensuring there were no timing violations or clocking issues. As depicted in Table 3, NOVA achieves a 2.41X frequency boost over PRISC-V, with a 1.62X increase in FFs and 1.37X in LUTs due to its 3-step execution and encoding scheme. When compared to PRISC-V_{DSP}, NOVA improves frequency by 2.12X which comes with an increase in resource usage—2.61X more LUTs and 1.85X more FFs while PRISC-V_{DSP} uses 12 DSP blocks occupying a large area. Deep pipelining enhances PRISC-VII’s frequency by 1.97X, matching PRISC-VII_{DSP}’s, 1.74X improvement. NOVA, achieves 22% better performance, though it consumes 1.23X LUTs and 1.08X FFs than PRISC-VII, and 1.24X FFs than PRISC-VII_{DSP}. Despite higher resource usage, NOVA remains competitive due to its frequency gains. In Fig. 4 (a) and Fig. 4 (b), we compared NOVA with all PRISC-V configurations to illustrate the performance

Table 3. Resource Utilization and Maximum Operating Frequency

| Article | SRC (LUTs, FFs, DSP) | Freq_Max (MHz) |
|--------------------------|----------------------|----------------|
| PRISC-V | 4212, 1425, 0 | 114 |
| PRISC-V _{DSP} | 2206, 1251, 12 | 130 |
| PRISC-VII | 4641, 2141, 0 | 225 |
| PRISC-VII _{DSP} | 2660, 1864, 4 | 225 |
| NOVA | 5762, 2309, 0 | 275 |

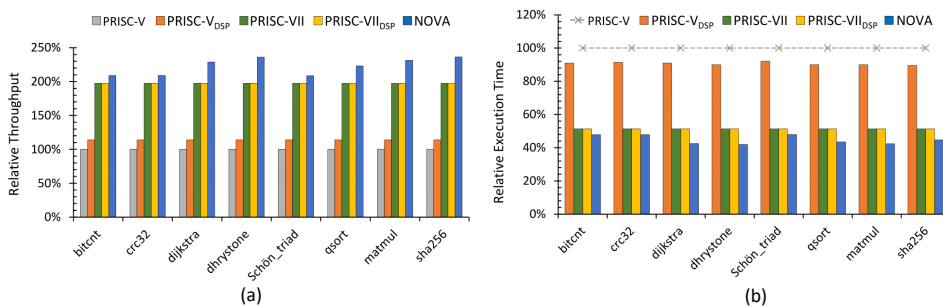


Fig. 4. Relative Throughput and Execution time for Various Applications.

improvement achieved by the employed method in terms of throughput and execution time. These metrics are measured using internal timers. The results have been normalized to the outcomes of the 5-stage DSP-less design, PRISC-V, serving as a reference for evaluating the efficiency of other cores. NOVA achieves a 2.40X throughput improvement over PRISC-V, especially in compute-intensive tasks like Dhrystone, SHA256, and Matmul, and 2.17X in memory-bound applications like Schönauer vector triad and CRC32. When compared to PRISC-V_{DSP}, NOVA shows a 2.10X boost in compute-intensive tasks and 1.83X in less demanding ones. The increase in pipeline stages for both PRISC-V and PRISC-V_{DSP} yields a 1.96X and 1.73X improvement respectively, but NOVA still surpasses them by over 22%, thanks to its efficient encoding scheme. Regarding execution time, NOVA offers a 2.37X reduction in compute-intensive tasks and a 2.11X reduction for memory-bound tasks over PRISC-V. PRISC-VII and PRISC-VII_{DSP} achieve 1.95X and 1.73X reductions compared to their 5-stage counterparts. However, NOVA still achieves over 20% better performance than both of them for compute-intensive tasks highlighting the effectiveness of NOVA’s CSD encoding and its capability to enhance processor performance. To evaluate NOVA’s loop-unrolling capability, we tested it on the Schönauer vector triad, a memory-bound application. Despite inherent performance limitations, 3-way unrolling for 9×10^6 iterations reduced execution time from 3.5 seconds to 2.5 seconds. However, due to limited compiler support and insufficient instructions, the full theoretical benefits were not achieved.

6 Conclusion

In conclusion, this paper presented a synthesizable μ -architectural design method that improved RISC-V processor performance by integrating CSD representation into the execution stage and extending the pipeline to seven stages. While the method introduced some conversion circuits and resource utilization overheads, it proved to be effective, especially in computation-hungry applications like signal processing. Its implementation on an open-source RISC-V processor

showcased its adaptability to other architectures, making it a promising solution for performance enhancements. Experimental results demonstrated a 2.41X boost in frequency, up to a 2.40X increase in throughput, and a 2.37X reduction in execution time for computation-intensive benchmark applications, where over 20% of the gain was due to the CSD encoding.

References

1. Avizienis, A., 1961. "Signed-digit number representations for fast parallel arithmetic". *IRE Transactions on electronic computers*, (3), pp.389-400.
2. Parhami, Behrooz. "Generalized signed-digit number systems: a unifying framework for redundant number representations." *IEEE Transactions on Computers* 39, no. 1 (1990): 89-98.
3. Hashemian, Reza. "A new method for conversion of a 2's complement to canonic signed digit number system and its representation." *Conference Record of the Thirtieth Asilomar Conference on Signals, Systems and Computers*. IEEE, 1996.
4. M. Reichenbach, J. Knödtel, S. Rachuj and D. Fey, "RISC-V3: A RISC-V Compatible CPU With a Data Path Based on Redundant Number Systems," in *IEEE Access*, vol. 9, pp. 43684-43700, 2021.
5. Ruiz, G.A. and Granda, M., 2011. "Efficient canonic signed digit recoding". *Microelectronics journal*, 42(9), pp.1090-1097.
6. Karthikeyan, S., Reddy, M.C.K. and Monica, P.R., 2017, August. Design of CNTFET-Based ternary control unit and memory for a ternary processor. In *2017 International Conference on Microelectronic Devices, Circuits, and Systems (ICMDCS)* (pp. 1-4). IEEE.
7. Kam, Dongyun, et al. Design and evaluation frameworks for advanced riscbased ternary processor. *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, 2022.
8. Kiran, M.L., Nihileswar, K., Ramanaiah, K.V. (2021). FPGA Implementation of CSD-Based NN Image Compression Architecture. *ICTACT Journal on Microelectronics*, 16(102), 102.
9. Srivastava, A.K., Raj, K. (2022). An Efficient FIR Filter Based on Hardware Sharing Architecture Using CSD Coefficient Grouping for Wireless Applications. *Wireless Personal Communications*, 123, 3433-3448.
10. Lee, H., Sobelman, G.E. (2002). FPGA-based Digit-Serial CSD FIR Filter for Image Signal Format Conversion. *Microelectronics Journal*, 33(5-6), 501-508.
11. F. EbrahimiAzandaryani and D. Fey, ExTern: Boosting RISC-V core performance using ternary encoding, *Microprocessors and Microsystems*, volume 107(2024), <https://doi.org/10.1016/j.micpro.2024.105058>.
12. K.K Skordal, D. Siorpaes, P. Cotret & J. Thomas. Potato Project, (2023). <https://opencores.org/projects/potato>
13. Guthaus, Matthew R., et al. "MiBench: A free, commercially representative embedded benchmark suite." *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4* (Cat. No. 01EX538). IEEE, 2001.
14. Weicker, Reinhold P. "Dhrystone: a synthetic systems programming benchmark." *Communications of the ACM* 27.10 (1984): 1013-1030.
15. Hofmann, J., Eitzinger, J. and Fey, D., 2015. Execution-cache-memory performance model: Introduction and validation. *arXiv preprint arXiv:1509.03118*.
16. Benson, Austin R., and Grey Ballard. "A framework for practical parallel fast matrix multiplication." *ACM SIGPLAN Notices* 50.8 (2015): 42-53.