Power Aware and Density Optimized High Fan-in Posit-HUB Using Modified CSLA

J.Prasanthi Kumari[1], Dr.B.Ratna Raju [2]

[1] M.Tech Student in Dept. of ECE at Srinivasa Institute of Engineering & Technology (Autonomous), NH-216, Cheyyeru(V), Amalapuram-533216.

[2] M.Tech, Ph.D, Professor and HOD of ECE at Srinivasa Institute of Engineering & Technology (Autonomous), NH-216, Cheyyeru(V), Amalapuram-533216.

ABSTRACT:
In recent years, the posit number system has gained attention as a promising alternative to the traditional IEEE floating-point representation, especially within the field of deep learning. Thanks to its non-uniform distribution of values, the posit format aligns more closely with the data characteristics observed in deep learning models. This advantage enables faster training and improved computational efficiency. Among various arithmetic operations, multiplication is one of the most frequently used in these applications. To address the hardware complexity associated with such operations, the HUB (Hybrid Unum Binary) approach, introduced in 2016, focused on reducing the cost of floating-point hardware units. Building upon this idea, a novel format known as the HUB posit has been proposed. This format aims to further decrease the hardware overhead typically found in posit arithmetic units. Specifically, this research introduces a power-efficient posit adder architecture. While the mantissa adder is still designed to accommodate the maximum possible bit-width, the architecture cleverly segments the adder into several smaller units. These smaller adders operate based on the regime bit-width, which effectively determines the required mantissa bit-width for a given operation. This design methodology is implemented and tested on both 6-bit and 11-bit posit configurations. Moreover, the architecture is enhanced by incorporating a modified carry select adder along with Binary Excess Converters. This enhancement significantly optimizes the performance by reducing both power consumption and processing delay, making it well-suited for efficient and scalable deep learning hardware solutions.

Keywords:—Posit number system, posit addition, computer arithmetic, low-power arithmetic circuit, HUB posit, Carry select adder, Binary Excess Converter.

INTRODUCTION:
The posit number system is a modern data format introduced to fully replace the traditional IEEE 754 floating-point standard (commonly referred to as floats). Unlike earlier forms of universal number (unum) arithmetic, posits do not rely on interval arithmetic or variable-length operands. Instead, they behave similarly to fractions—automatically rounding results when exact values cannot be represented. Posits offer several advantages over conventional floats, including a broader dynamic range, improved precision, and consistent bit-level results across platforms. Additionally, they simplify hardware design and exception handling while providing better closure in mathematical operations. One standout feature is that posits inherently avoid overflow and underflow conditions. They also redefine how "Not-a-Number" (NaN) is handled—representing it as an operational state rather than a reserved bit pattern. In terms of hardware, posit processing units (PPUs) are generally more efficient and require fewer resources than IEEE-based floating-point units (FPUs). Their asymmetrical or non-uniform data distribution aligns well with the nature of data in many real-world scenarios, particularly in deep learning. Posit formats like 8-bit and 16-bit are now commonly used in deep learning applications, thanks to their efficiency and ability to maintain precision with lower memory usage. In scientific computing, the 32-bit posit format is often employed as a more compact and effective alternative to the standard 64-bit floating-point representation. A posit number is typically defined as Posit(nb, es), where *nb* represents the total number of bits, and *es* denotes the number of exponent bits. Each posit consists of four segments: the sign bit, regime, exponent, and mantissa (also known as the fraction). Except for the fixed 1-bit sign, the lengths of the remaining components are dynamic and vary depending on the encoded value. Originally proposed in [1], the posit format was designed to offer a more efficient and accurate numeric representation across a variety of

applications [2–5]. Its adaptive structure and compatibility with uneven data patterns make it particularly well-suited for memory-constrained and computation-intensive environments like machine learning and scientific research.

In the posit number format,

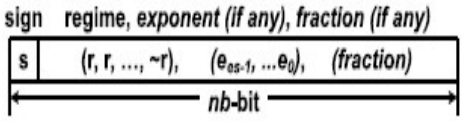


Fig. 1. Each component of a posit number.

The sign bit and the regime field are always present. However, the exponent and mantissa fields are included only if there are remaining bits after allocating space for the sign and regime. As a result, the bit-width of the mantissa (including the implicit bit) can range from just 1 bit up to a maximum of *(nb − es)* bits, where *nb* represents the total number of bits in the posit format and *es* indicates the number of exponent bits.

In earlier designs of posit-based arithmetic units such as posit adders and multipliers [6], posit adder generators [7], and multiply-accumulate (MAC) unit generators [8] the mantissa multiplier is typically constructed to handle the full maximum width of *(nb − es)* bits. However, in practical usage, the actual mantissa size often falls below this maximum value. As a result, using a full-width multiplier for smaller mantissas leads to unnecessary power consumption and inefficient energy usage.

This design mismatch highlights the need for more adaptive hardware implementations, where the mantissa multiplier dynamically adjusts to the actual bit-width requirements, thereby optimizing both power and performance in posit arithmetic operations.

LITERATURE SURVEY:

The mathematical foundation for the Chinese Remainder Theorem (CRT) and the conceptual underpinnings of the POSIT multiplication (POSIT MUL) approach were originally established by Carl Friedrich Gauss in his influential 19th-century work *Disquisitiones Arithmeticae* [8]. The practical implementation of POSIT MUL in hardware first appeared in 1955, through the efforts of Lehmer, Svoboda, and Valach [1]. During the 1950s and 1960s, significant development occurred across various research laboratories, focusing on building hardware based on POSIT MUL principles. Notably, scholars such as Szabo and Tanaka [11], as well as Watson and Hastings [14], documented many of these advancements in publications released in 1967.

However, from 1967 to 1977, research in this domain slowed, largely due to skepticism among processor designers about the feasibility of building practical POSIT MUL-based computing systems. A renewed wave of interest emerged in 1977 when Jenkin and Leon [6] presented innovative research that inspired a resurgence in this field. Between 1977 and 1985, several noteworthy contributions were made, culminating in a comprehensive survey of the field by 1986 [10]. The momentum continued with Mohan [8], who compiled and shared further research findings up to 2001.

Since 2002, new developments have consistently emerged, with several contemporary researchers making valuable contributions [7]. In the area of converting residue number systems to binary, two main methods have been dominant: CRT and Mixed Radix Conversion (MRC). CRT is particularly advantageous due to its inherent support for parallel computation, whereas MRC, being a sequential approach, is more commonly used in modern POSIT MUL-to-binary or decimal converters. This is primarily because the modulo-M operations involved in CRT—where M is the system's dynamic range—are computationally intensive and slower.

In more recent advancements, the PACoGen framework has been proposed, offering an open-source hardware generator for posit arithmetic cores. This tool supports automated generation of Verilog HDL code for posit arithmetic units across various word sizes (*N*) and exponent sizes (*ES*). PACoGen also introduces pipelined architectures for 32-bit posits with a 6-bit exponent field, targeting key arithmetic operations such as addition, subtraction, and division. These designs have been validated on platforms like the Virtex-7 FPGA (xc7vx330t-3ffg1157) and the 15nm Nangate ASIC, showing promising results in terms of performance and hardware efficiency.

In another study [2], a power-efficient posit multiplier design is introduced to address the high energy cost typically associated with large mantissa multipliers. Due to the dynamic bit-width nature of the posit format, earlier designs often defaulted to using the maximum mantissa width, leading to unnecessary power consumption. The new architecture optimizes this by dividing the full-width multiplier into smaller blocks, achieving significant power savings—up to 16% on average—across 8-bit, 16-bit, and 32-bit posit implementations with minimal impact on area and delay.

Additionally, [3] outlines the development of a high-speed floating-point multiply-accumulate (MAC) unit using FPGA technology. Recognizing the critical role of MAC units in modern computing, the

authors focus on reducing the critical path delay to boost performance. Their design uses pipelining and parallelism to accelerate operations and demonstrates superior speed on a Xilinx Virtex-7 FPGA, without compromising area or power efficiency.

Lastly, [4] presents a novel parameterized and pipelined Posit Fused Multiply-Accumulate (P-FMA) unit capable of performing multiple arithmetic operations—addition, subtraction, multiplication, and fused MAC. Designed with a five-stage pipeline, the P-FMA unit offers flexibility and is compared against other Posit-based FMA units. To evaluate its accuracy, the authors employed the Chudnovsky algorithm to compute π (pi). The results show that a 64-bit Posit format with varying exponent sizes (es = 1 to 6) produced a 16-digit accurate estimate of π surpassing the 15-digit accuracy achieved by double-precision IEEE 754 floating-point implementations using Y-Cruncher. This unit was synthesized and tested on FPGA platforms, reinforcing its value in high-precision arithmetic tasks.

PROPOSED METHOD:

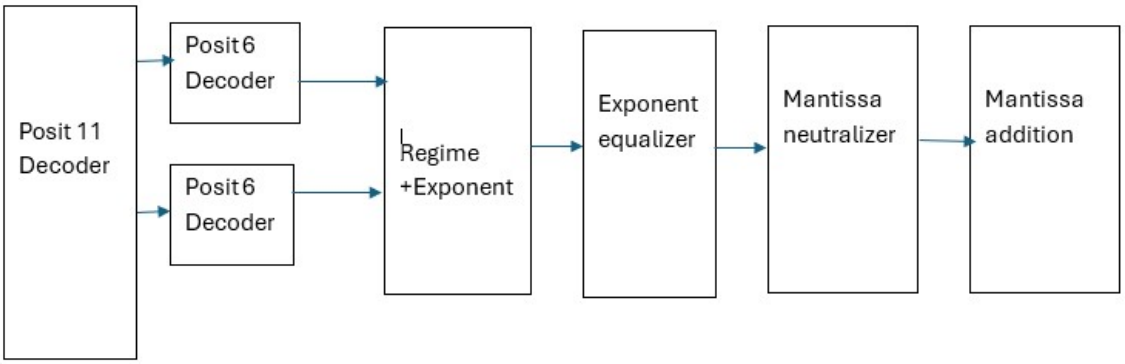


Fig2: Proposed Block diagram

Table I shows several examples comparing the values represented for conventional and HUB 6-bit posit numbers. The r, e, and f columns represent the corresponding values in (2) extracted from the bit representation, including the iLSB for the HUB version. Notice how the HUB format always results in values greater than the corresponding posit in both positive and negative cases, since all values are biased in the same direction.

TABLE I CONVENTIONAL AND HUB VALUES FOR POSIT6

Binary	Conventional			HUB		
	r	e	f	r	e	f
000011	−3	2	0	−3	3	0
001011	−1	1	0.5	−1	1	0.75
010000	0	0	0	0	0	0.25
011111	4	0	0	5	0	0
100001	−4	0	0	−4	0	0
110000	0	0	0	0	0	0.25
110101	0	2	0.5	0	2	0.75
111101	2	2	0	2	3	0

HUB POSIT ARCHITECTURES:

Posit arithmetic units first decode the operands to obtain the exponent and the fraction. This is performed by detecting the size of the regime and shifting accordingly the exponent and fraction bits to the left. In this way, the

size of the exponent and fraction are expanded to their regular size. Fig. 2(a) shows the hardware involved in this operation. In contrast to the FP implementation, where the inclusion of the iLSB can be delayed to just right before the fixed-point operation , for posit numbers the iLSB should be included at this stage to append it to the right position (either the fraction, the exponent or the regime). Fig. 2(a) illustrates this in blue.

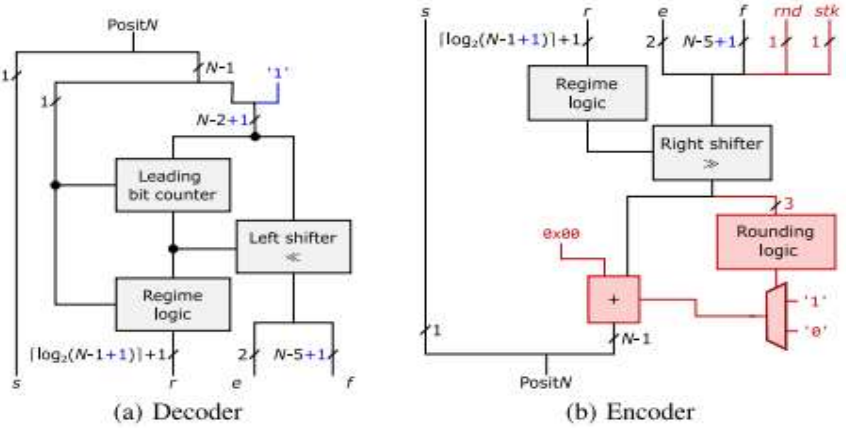


Fig. 3 Architecture of a HUB Posit decoder and decoder modules.

Once the intermediate Least Significant Bit (iLSB) is added to the result, the number behaves like a standard posit but with one additional bit. It's important to note that when the posit format uses a bit-width that is a power of 2, the regime field remains unchanged. Instead, the increase in bit-width reflects solely in the fraction (mantissa) field. This means the resulting value can still be processed using traditional posit hardware circuits, provided that the extra bit is accounted for during computation.

However, to enhance efficiency, certain optimizations are introduced in the hardware depending on the type of operation. One significant change lies in the encoding or packing process. In this stage, the regime is determined from the scaling factor of the result, and the exponent and fraction fields are right-shifted accordingly. Standard posit implementations utilize the bits lost during shifting to perform rounding—typically by rounding the fraction or, if all fraction bits are discarded, rounding the exponent. This rounding step involves additional logic and possibly an adder.

In contrast, the HUB (Hardware Under Budget) approach skips this rounding logic entirely. The result is simply truncated after shifting, thereby reducing hardware complexity. As illustrated in Figure 2(b), the logic removed in the HUB version is highlighted in red. Beyond adjustments in the encoder and decoder, further circuit simplifications can be made based on the operation being executed, especially for addition.

In a standard posit addition, the mantissas are aligned by right-shifting the smaller operand based on exponent difference. Three extra bits—including the "sticky" bit—are preserved from the shifted mantissa to aid in proper normalization and rounding to the nearest value. However, in the HUB design, rounding is omitted, so these bits are unnecessary. Instead, one additional bit must be considered in the fixed-point adder to accommodate the iLSB.

Nevertheless, a few edge cases require special handling in the HUB variant:

- **Excessive Right Shifting of Negative Numbers:** When a negative operand is right-shifted by more bits than the size of its mantissa—effectively aligning it to a much smaller value—it can unintentionally subtract one Unit in the Last Place (ULP) from the larger operand. This occurs due to sign extension and the limited bit-width of the adder. In traditional implementations, this is naturally corrected during rounding. Since HUB eliminates rounding, this case is handled by applying a logical right shift instead of an arithmetic one when detected.
- **Catastrophic Cancellation:** This situation arises when two nearly equal values are subtracted, leading to significant loss of precision. In the regular implementation, a discarded LSB from the smaller operand is preserved in the rounding bit. But for the HUB version, where rounding logic is removed, this discarded bit must be explicitly added back to the result before the normalization step to maintain numerical accuracy.

In summary, while HUB streamlines the posit arithmetic circuits by removing rounding-related logic, it introduces a few specific adjustments in the datapath to address precision and alignment issues, particularly for addition operations.

DESIGN OF SQUARE ROOT CSLA

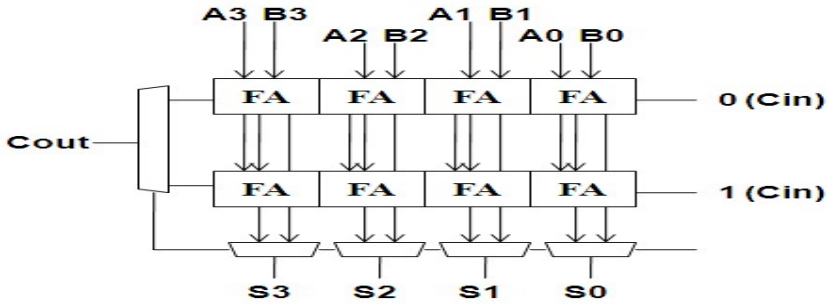


Fig4 : Basic building block of CSLA

The basic building block of the square root carry select adder of block size '4' is shown in the above figure.

Block diagram of 16-bit Carry Select adder:

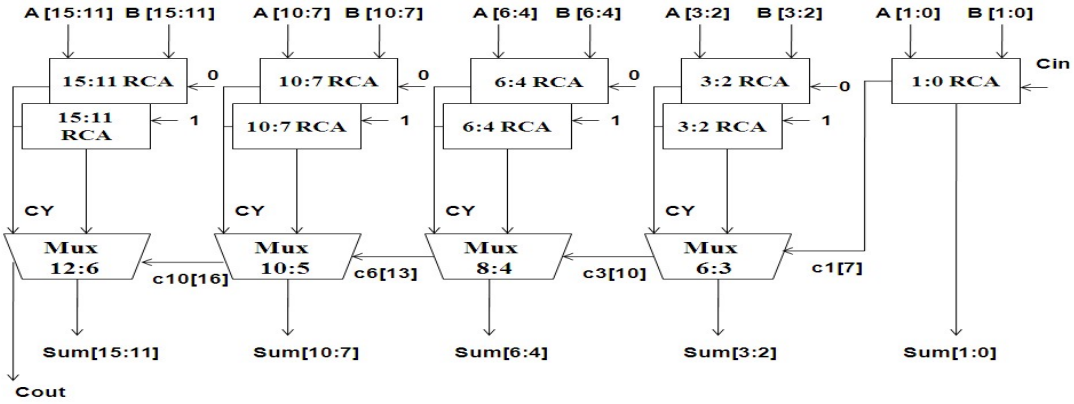


Fig5: Existing system (Regular 16-bit Carry select adder)

The block diagram of the regular 16-bit square root CSLA is shown in the figure 3.2. This adder is a variable sized adder.

The carryselect adder generally consists of two ripple carry adders and a multiplexer. Adding two n-bit numbers with a carry-select adder is done with two adders (therefore two ripple carry adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known.

Binary to Excess-1 Converter (BEC):

Design of CSLA with BEC:

As stated above, the main idea of this work is to use BEC instead ofthe RCA with Cin=1 in order to reduce the area and power consumptionof the regular CSLA. To replace the n-bit RCA, an (n+1)-bit BECis required. The structure and the function table of a 4-b BEC are shownin the figure 3.7 and Table 3.3 respectively.

Structure of BEC:

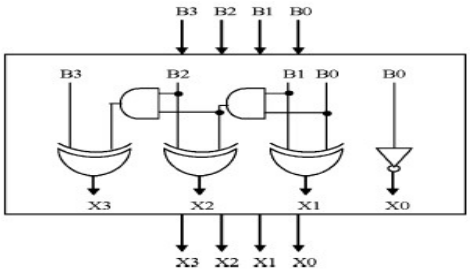


Fig6 : Structure of 4-bit BEC

B [3:0] (Binary inputs)	X[3:0] (Excess-1 outputs)
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	1010
1010	1011
1011	1100
1100	1101
1101	1110
1110	1111
1111	0000

TableII: Function table of BEC

The Boolean expressions of the 4-bit BEC are listed as below:

X0 = ~B0
X1 = B0 ^ B1
X2 = B2 ^ (B0 & B1)
X3 = B3 ^ (B0 & B1 & B2)

RESULTS:

The figure7 illustrates how the basic function of the CSLA is obtained byusing the 4-bit BEC together with the mux. One input of the 8:4 muxgets its input from B3 B2 B1 B0 and another input to the mux is theBEC output. This produces the two possible partial results in paralleland the mux is used to select either the BEC output or the direct inputsaccording to the control signal Cin.

Block diagram of CSLA with BEC:

The structure of the proposed SQRT CSLA using BEC for RCAwithCin=1 to optimize the area and power is shown in the figure

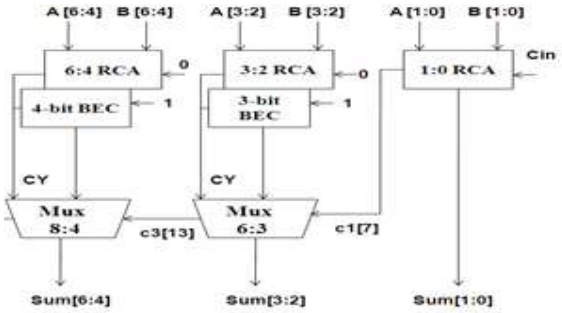


Fig7: Modified system (Modified SQRT CSLA)

Comparing the block diagram of the regular square root CSLA with the modified square root CSLA, it can be seen that the RCA with Cin=1 is replaced by BEC (binary to excess-1 converter). This is done to reduce the area consumption. This can be seen after evaluating the group delay and the number of gates required for the design.

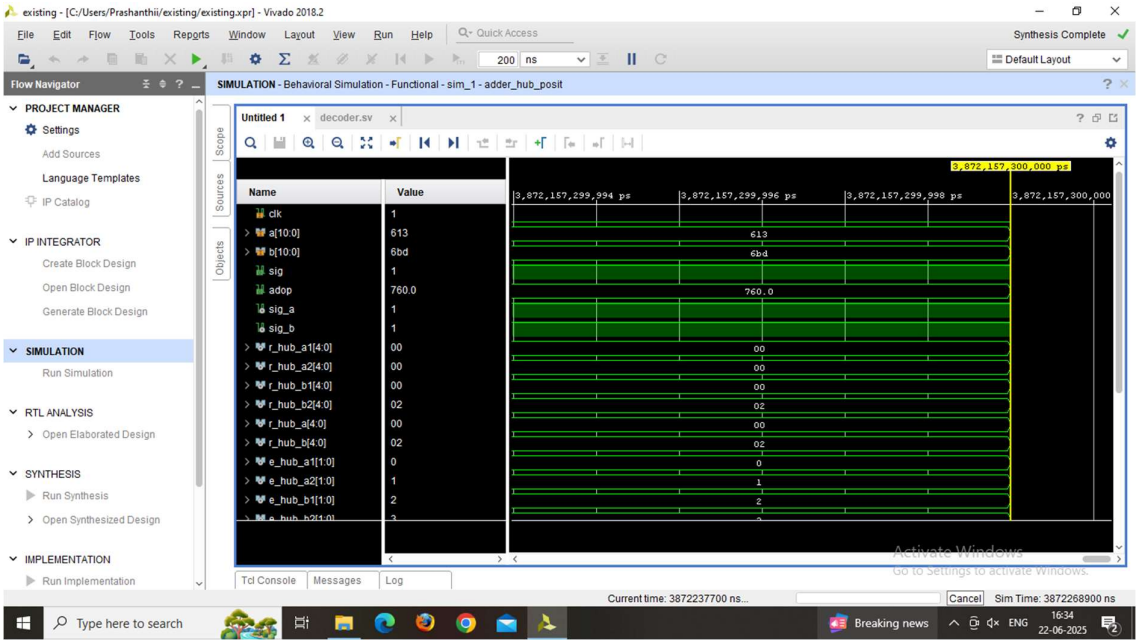


Fig a. Simulation result captured in XILINX VIVADO

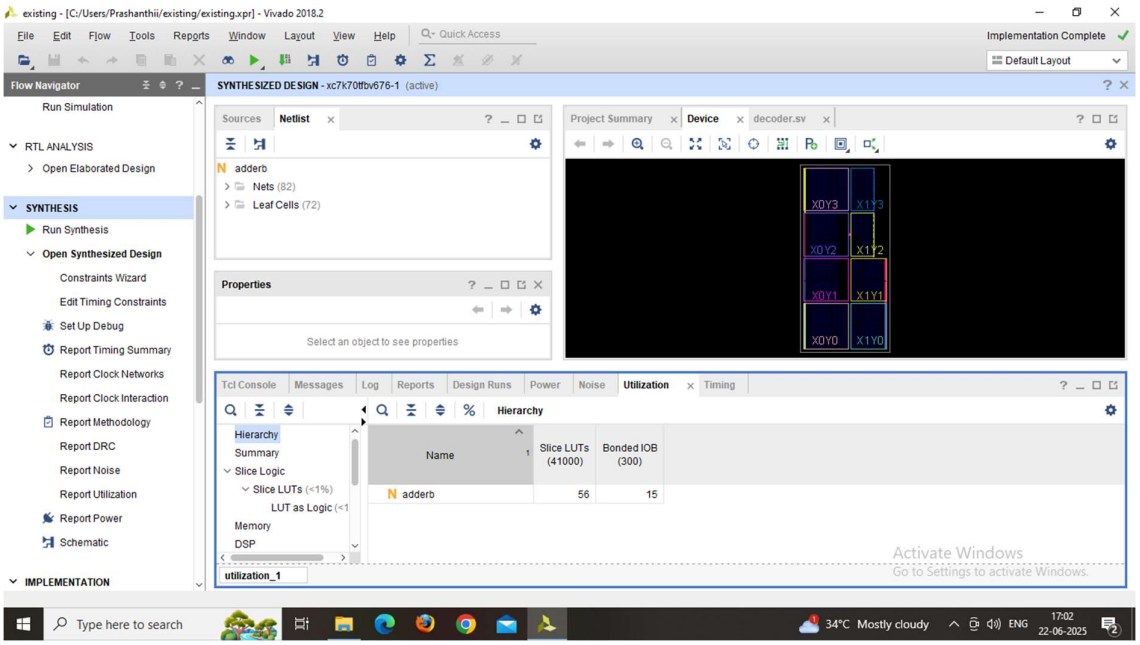
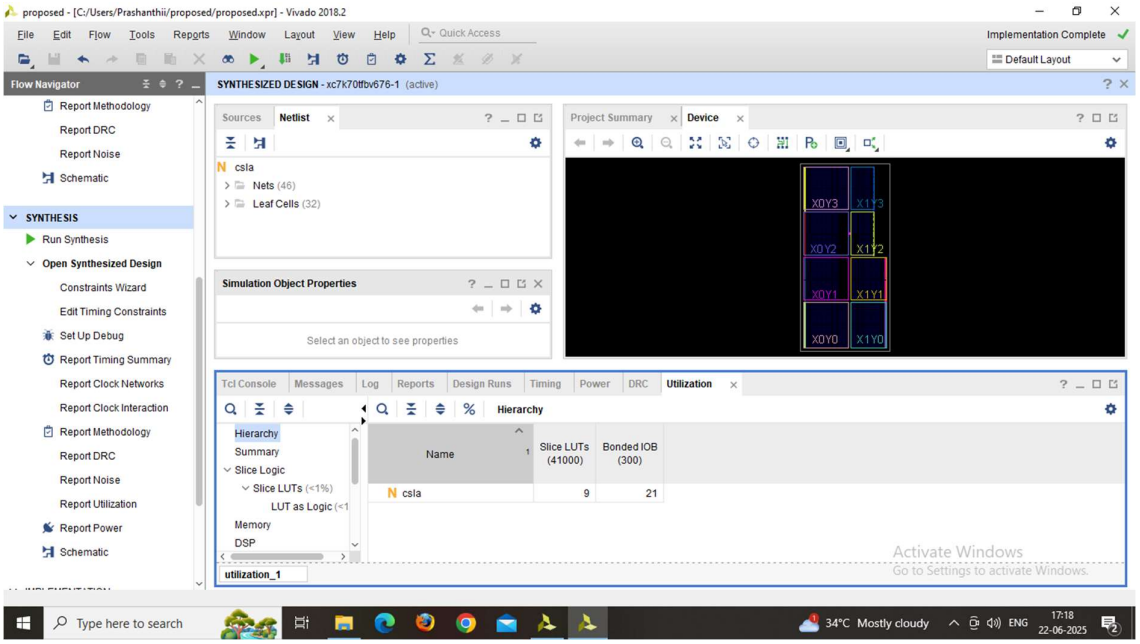


Fig b: Existing Area report in terms of gate density



Figc: Proposed Area report in terms of gate density

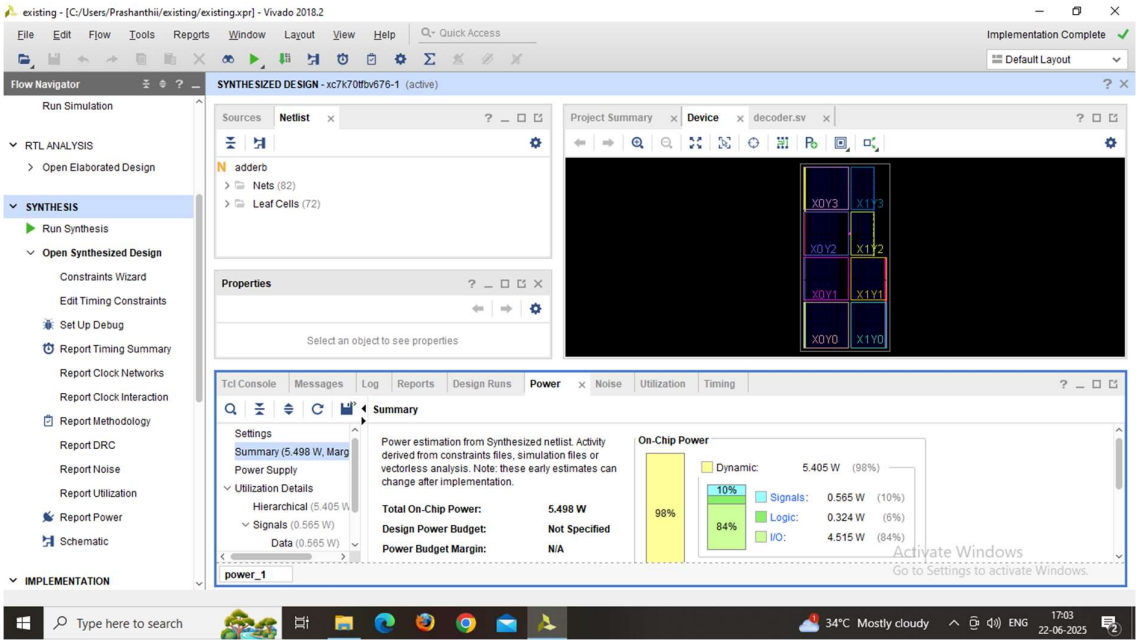


Fig d: Power report for existing method

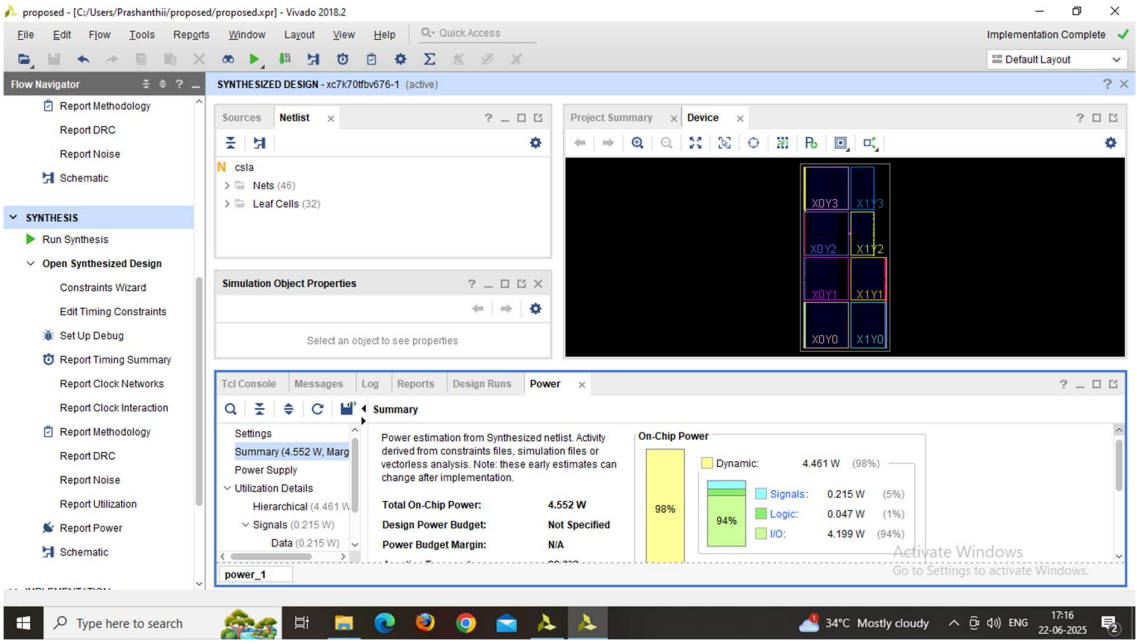


Fig e: Power report for proposed method

CONCLUSION:

In this work a power-efficient architecture for posit adders has been introduced. This design strategically activates only the necessary components during run-time, thereby minimizing unnecessary signal transitions and significantly reducing power consumption. While the overall power savings achieved through the HUB (Hardware Under Budget) approach in posit arithmetic units may not be as substantial as those seen in floating-point (FP) units, the benefits are nonetheless noteworthy. Importantly, these optimizations are accomplished without compromising the numerical accuracy typically offered by standard posit implementations. This technique proves especially beneficial for developing energy-efficient posit arithmetic units and is well-suited for applications where power constraints are a primary concern, such as in portable or embedded systems.

Future Scope:

In the future, more power reduction opportunity in the posit complex fused multiplier architecture will be explored. In addition, the investigation of power efficient posit arithmetic unit design will be extended to posit adder and posit multiply-accumulate unit.

REFERENCES:

[1] E. Grosswald, "Topics from the Theory of Numbers," New York: McMillan, 1996.

[2] W. K. Jenkins, "Finite Arithmetic Concepts in Handbook for DSP," S. K. Mitra and J. F. Kaiser, eds., Wiley, 1993, pp. 611-675.

[3] F. J. Taylor, "Residue arithmetic: A tutorial with examples," Computer (IEEE), vol. 17, no. 5, pp. 50-63, May 1984.

[4] A. Omondi and B. Premkumar, "Residue Number System: Theory and Implementation," Imperial College Press 2007, ISBN 978-1-86094-866-4.

[5] N. Szabo and R. Tanaka, "Residue Arithmetic and its Applications to Computer Technology," New York: McGraw Hill, 1967.

[6] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, "Residue Number System Arithmetic: Modern Applications in Digital Signal Processing," New York: IEEE Press 1986.

[7] Wei Wang, M. N. S. Swamy, and M. O. Ahmad, "An area-time-efficient residue-tobinary converter," Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems, Vol. 2, pp. 904-907, 2000.

[8] F. Barsi and P. Maestrini, "Error correcting properties of redundant residue number systems," IEEE Transactions on Computers, vol. 23, no.9, pp. 915-923.

[9] R. Conway and J. Nelson, "Improved POSIT MUL FIR Filter Architectures," IEEE Transactions On Circuits and Systems II, Vol. 51, No. 1, pp. 26-28, 2004.

[10] M. K. Jaiswal and H. K. So, "PACoGen: A hardware posit arithmetic core generator," IEEE Access, vol. 7, pp. 74586–74601, 2019.

[11] L. Sommer, L. Weber, M. Kumm, and A. Koch, "Comparison of arithmetic number formats for inference in sum-product networks on FPGAs," in Proc. IEEE 28th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM), 2020, pp. 75–83.

[12] R. Murillo, A. A. Del Barrio, and G. Botella, "Customized posit adders and multipliers using the FloPoCo core generator," in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), Oct. 2020, pp. 1–5.

[13] H. Zhang and S.-B. Ko, "Design of power efficient posit multiplier," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 67, no. 5, pp. 861–865, May 2020.

[14] L. Crespo, P. Tomás, N. Roma, and N. Neves, "Unified posit/IEEE-754 vector MAC unit for transprecision computing," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 69, no. 5, pp. 2478–2482, May 2022.

[15] J. Hormigo and J. Villalba, "Measuring improvement when using HUB formats to implement floating-point systems under round-to-nearest," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 24, no. 6, pp. 2369–2377, Jun. 2016.

[16] J. Hormigo and J. Villalba, "HUB floating point for improving FPGA implementations of DSP applications," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 64, no. 3, pp. 319–323, Mar. 2017.

[17] S. Bose, A. De, and I. Chakrabarti, "Framework for automated earthquake event detection based on denoising by adaptive filter," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 67, no. 9, pp. 3070–3083, Sep. 2020.