2023 36th International Conference on VLSI Design and 2023 22nd International Conference on Embedded Systems (VLSID)

2023 36th International Conference on VLSI Design and 2023 22nd International Conference on Embedded Systems (VLSID) | 979-8-3503-4678-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/VLSID57277.2023.00059

Design and Analysis of Posit Quire Processing Engine for Neural Network Applications

Pranose J Edavoor*, Aneesh Raveendran*, David Selvakumar*, Vivian Desalphine*, Dharani Shankar G* and Gopal Raut*
*Secure Hardware and VLSI Design Group, Centre for Development of Advanced Computing Bangalore, India.
{pranoseje,raneesh,david,viviand,dharanis,gopalraut}@cdac.in

***Abstract*—Multiplications-Accumulations (MACs) are the significant computations in deep learning networks that must be hardware efficient and result in higher inference accuracy for the given set of network layers. Typically, the processing engines (PEs) which comprise the MAC units are the most resource-intensive element in a Deep Neural Network (DNN) accelerator. This paper proposes efficient PEs and presents an architectural design, accuracy, and resources cost analysis for hardware-based parameterized Posit Quire Processing Engine (PQPE) supporting P(8,0), P(16,1), P(32,2), and P(64,3). PQPE performs the vector computations (dot products) on Posit numbers with a wide quire accumulator and has been enabled with instructions, commands, and responses. PQPE has been modelled in Python to evaluate the inference accuracy for various DNN kernels and datasets. LeNet-5, AlexNet, custom 5 layers and 4 layers CNN models have been adopted and evaluated for inference accuracy with MNIST, CIFAR-10, and Alphabet recognition datasets. The PQPE has been used for matrix computations in convolutions. P(16,1) without quire provides same accuracy as float32 with less hardware resources. P(16,1) with quire brings same accuracy level as without quire and hence quire is not advantageous for larger posit data widths. Posit (8,0) with / without quire results insignificant loss of 0.8% / 3.28% (LeNet + MNIST) to 3.8% / 10.1% (AlexNet + CIFAR-10) accuracy as compared to P(16,1)/float32. Further, PQPE was modelled using Verilog HDL, synthesized targeting Xilinx Virtex-7 FPGA using *Xilinx-Vivado*. PQPE (8,0) achieved a f_{max} of 372 MHz and utilized 77.6% fewer hardware resources as compared to float32 at the cost of insignificant accuracy loss for CIFAR-10 which is 0.8% on LeNet and 3.8% on AlexNet. PQPE (16,1) has used 6.5% less resources and achieved equivalent accuracy as compared to float32.**

***Index Terms*—Posit MAC Engine, Posit Quire Accumulators, Posit for AI/ML, Neural Network Processing Engine, Inference Accuracy.**

I. Introduction

ARTIFICIAL Intelligence and Machine Learning (AI/ML) has been applied and adopted in various technologies, in recent times. AI/ML algorithmic architectures have many variants including deep neural networks (DNNs), which are popular due to their diverse applications in various fields viz. scientific, engineering, agriculture, healthcare etc. Although DNN is a powerful technique, the hardware implementation cost for a DNN model is very high [1]. DNN consists of one input layer, one output layer, and multiple hidden layers, and the primary architectures involved in DNN are Fully Connected Neural Network (FCNN), Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN). Each DNN layer performs matrix (vector) multiplication and accumulation, and a layer's output is input to the subsequent layers within the network [2]. In general, the arithmetic representations for various numbers in neural net computations are IEEE-754 float, fixed-point, bfloat16 or Posit [3].

The selection of number representation scheme and the associated numeric computations play a significant role in efficient AI/ML hardware architecture design and the accuracy of machine learning inference. For DNN to have better inference accuracy, IEEE-754 floating-point representations are mostly preferred with a precision of 16-bit (half) or 32-bit (single) or bfloat16 [4]. However, the hardware design comes with higher resource utilization and more power consumption. For an efficient hardware design, fixed-point representation is used with $Q_{4.4}$ or $Q_{8.8}$ schemes. Recently, a new number system, i.e., Posit, has found a place with an ML inference accuracy level similar to floating-point but requiring fewer hardware resources like fixed-point arithmetic [5]. Posit is a recent development in Universal Number (`Unum`) system and is the Type-3 Unum. It is a representation that provides a wider dynamic range as compared to IEEE-754 for the same data width viz. 8/16/32/64-bits. Further, a 16-bit IEEE-754 number can be represented using an 8-bit Posit number achieving a double data range with varying exponents size [6]. Hence, the Posit scheme is better suitable for multiply-accumulate (MAC) computations in DNN's Processing Elements.

In DNN, the processing engines (PEs) are the foremost arithmetic computational units, on which major multiplication and accumulation operations are performed [4]. The PEs take more than 90% of computation time in DNN accelerators [7]. In DNN layer's functions, the PEs perform iterative computations for multiplication and accumulation to make hardware resources efficient design [8], [9]. For an image convolution, with the matrices of the order N*N, N^2 multiplications and N×(N-1) addition operations are required. If multiplication and accumulation are done separately each will take one clock cycle. Therefore, the latency for the MAC layer's computation is large, i.e., sum of N^2 and $\big(\text{N}\times(\text{N-1})\big)$ clock cycles in the case of both fixed point and floating point schemes without the fused MAC. But, fused MAC without wider register reduces the accuracy of MAC computations significantly. Moreover, Posit has introduced Quire, having a wide internal register for fused accumulation of multiplication results and avoiding external addition.

The Posit with Quire computation involved larger adders with long carry propagation which requires a larger quire. However, the computation provides better accuracy (25%) [10], higher throughput, and dynamic pipeline configuration. The Posit without Quire will have lesser resources at the cost of lower accuracy. To make a balanced resources and performance-centric design, the proposed design of PE has used Quire, which gives much better accuracy. Few literature have adopted Posit and Posit with Quire for inference, but both are not compared and analyzed in terms of inference accuracy for various Neural Network kernels. Hence, in this work, we have designed and realized a Posit PE and a PQPE in hardware and analyzed the inference accuracy of both Posit and Posit with Quire and compared with float32 and float16, etc. as well. Further, the hardware resources for the PEs are also compared and analyzed.

In order to improve the inference accuracy with reduced hardware complexity in comparison with float32, we proposed a Posit Quire Processing Engine (PQPE) for DNN accelerators. The design is efficient in terms of hardware resources and has less critical path delay, which improves the throughput. The major contributions under this work are summarised as follows:

- PQPE that enables better inference accuracy and higher throughput due to fused and wider MAC
- Neural computations specific Instructions Encoded Commands and Responses
- Error Estimates and Analysis of PQPE with other PEs viz. Posit without Quire, IEEE 754 Float (16 and 32 bits) etc.

The proposed PQPE based design used 22.4% fewer resources as compared to the other state-of-the-art designs and provided better inference accuracy as well. The remainder of this paper is organized as follows: Section II details the related works on various Posit arithmetic based PE architectures and the design techniques. The proposed PQPE architecture and the algorithm kernels are explained in Section III. Section IV shows the hardware architecture of the proposed design. The hardware implementation results have been extracted, and evaluation and analysis of the results are given in

2380-6923/23/$31.00 ©2023 IEEE
DOI 10.1109/VLSID57277.2023.00059

Section V. Section VII concludes the article.

II. Related Works and Motivation

The attention towards the hardware implementation of Posit number system based computational units for AI/ML applications has rapidly increased since its first introduction in 2017. The work presented in [10] by Raul Murillo *et. al*. explains a framework for performing neural network training and inference based on Posit. However, the hardware implementation for the same had not been projected. De Dinechen *et. al.* discussed the advantages and disadvantages of Posits in comparison to floating point number system [11]. The authors have discussed the areas where Posits are better and vice versa, and gave an insight about Posit hardware cost. Authors further discussed the implementation of fundamental functions viz. Logarithmic, Exponential etc. for a Posit number system. In this paper, the authors have introduced and explained how Posit is more suitable for machine learning applications. In recent times, various approaches have been introduced using Posit number system to make the machine learning training and inference faster, more accurate and energy efficient. Chen *et. al.* introduced a matrix multiplier with re-configurable logic for 32-bit Posit number based matrices [12]. The multiplier comprises of five vector dot-product units, and no intermediate rounding is done in vector dot-product computation and instead a 512-bit quire register has been used for accumulation.

Carmichael *et. al.* proposed DNN accelerator hardware with three different number systems (fixed-point, floating-point and Posit) with less than 8-bit precision. [3]. For inference, the authors introduced the exact multiply-and-accumulate (EMAC) units. This paper suggested that Posit arithmetic is a natural fit for inference at ultra-low bit-width. At same bit-widths, the accelerator architectures have been prototyped on FPGA platform and was compared for all three number systems. Study showed that Posit EMAC provided better maximum operating frequency in comparison to floating point number system. Also, accuracy comparison has been done on various datasets. Lim *et. al.* developed a method that used Coordinate Rotation Digital Computer (CORDIC) for the development of Posit math library [13]. Further, CORDIC-based MAC with minimal resources utilization have been designed in [8]. At each iteration, algorithm rotates a vector with different angles to estimate trigonometric functions. Posit CORDIC library was five times faster and produced more accurate results in comparison to 1024 bits high precision library.

H.F. Langroudi *et. al.* developed a DNN framework named PositNN using Posit number system [14]. Authors proposed an algorithm to perform Posit dot product with n-bit inputs and for accumulation, quire was used. Authors claim similar accuracy with ultra-low precision PositNN for image classification with 5-bit to 8-bit low precision Posit as compared to 32-bit high precision DNNs. Further, in comparison to fixed-point and floating-point systems, the scheme was able to provide lesser memory utilization due to the reduced bit-widths. N. Neves *et. al.* developed a Posit based architecture named Dynamic Fused Multiply-Accumulate (DFMA) that provides high dynamic range and supports adjustable exponent size [15]. Murillo *et. al.* introduced a Deep PeNSieve framework with the use of Posit number system for DNN training and inference [10]. For low precision inference, the framework used Posit8 and fused operations. The framework used Posit(32,2) and Posit(16,1) for training, and quantization to Posit(8,0) was done with the use of quire and fused dot product. As compared to floating point number, training framework achieved 4% improvement. The performance comparison of 8-bit Posit with respect to 16 bit float and 8 bit integer on CNN models for training and inference have shown similar results. H.F. Langroudi *et. al.* proposed an adaptive Posit numerical format for non–uniform weights and activation functions of DNNs [16] and have claimed 10% boost in performance with adaptive Posit as compared to fixed Posit based uniform weights etc.

Forget *et. al.* did a comparison of hardware cost, for the implementation of floating-point (IEEE-754) and Posit number system. [17]. The study suggests that for same bit-width, Posit-to-Posit operators are more expensive than IEEE-754 in terms of delay and resources. A review was provided about the development and usage of Posit in high-performance computing (HPC) and edge computing by Poulos *et al* [18]. S. Nambi *et. al.* introduced a novel representation for Posit number system as well as implementations of fixed-point arithmetic for ANNs, with higher precision and better dynamic range for different applications [19]. A novel Posit to fixed-point converter algorithm, enabled high performance and energy efficient hardware implementations with minimum fall in output frequency for ANNs. B.Zhou *et. al.* proposed a fully-pipe-lined floating-point multiply-accumulator (FPMAC) architecture with signed soft multiplier and a single cycle FAAC [4]. G. Raposo *et. al.*, introduced a new neural network framework: PositNN, which can perform training and inference using Posit [20]. The proposed framework considerably improved the accuracy of different operations i.e. matrix multiplications, convolutions etc., with the use of quires for accumulation. Framework produced similar results with float32 and Posit8 for both training and inference. R. Murillo *et. al.* proposed a Posit MAC unit to perform fused arithmetic with a support for quire [21]. The performance of proposed design was assessed on DNNs by accuracy estimates and evaluation for different data-sets, and produced nearly similar accuracy for both float32 and Posit32. Further, with an increase of only 7.44% area, the design was able to reduce latency and energy consumption upto 87.36% and 88.45% respectively.

In this work, in order to make a balanced resources and performance-centric design of the PE, a low-level hardware architecture for parameterized Posit quire with instruction encoding enabled multiply-accumulate unit (dot product) has been proposed. Further, the absolute error analysis for dot-product computations based on Quire in neural network layers for various Posit precision values have also been presented. The proposed quire based design is evaluated for various neural network models for inference accuracy. Besides, hardware resources consumption is also compared and analyzed.

III. Posit Quire Processing Engine (PQPE) - Algorithms

The conventional Posit PE and its design, synthesis and FPGA realization show that encoding, decoding, and multiplication components consuming 50% of the total processing engine resources, and the Posit adder unit consuming 40% of the total energy of the PE. Further, critical path delay mostly depends on the encoding and encoding scheme [22]. To make an efficient design, the proposed parameterized Posit Processing Engine with Quire (PQPE) mainly performs three operations: Multiply-and-Accumulate (MAC) with Quire, Clear Quire Register, and Read Out Quire Register. The PE multiplies the two input Posit operands, converts the result into fixed-point quire format and adds to the current Quire register value for accumulation. The MAC operation comprises multiple steps like unpacking Posit operands, Scaling Factor calculation, Mantissa Multiplication, product's mantissa to Quire (fixed-point) conversion, and Quire Accumulation. The Quire register is reset by `clear` operation. The `read` operation with the Quire register converts the fixed-point quire value to Posit and emits the output. The `read` operation comprises multiple steps requiring a scaling factor calculation, k-value, exponent & mantissa extraction from the quire register, Posit reconstruction, rounding, and packing of the results. The following subsections elaborates the algorithms flow that have been adopted for efficient design of Posit Quire Processing Engine.

A. Posit Operands Unpacking and Scaling Factor Calculation

The parameterized Posit PE having Multiply and Accumulate with Quire accepts two (N, es) Posit operands as shown in Algorithm 1. Besides, PE takes 3 control bits, 1-bit for `clear`, 1-bit for `read` and 1-bit for `accumulate` modes and overall it produces N-bits Posit result and 5-bits exceptions.

Algorithm 1: Unpacking of Posit Operands and Scaling Factor Calculation (N, es)

1) N : Posit Word Size
2) ES : Posit Exponent Size
3) RS : Regime Size ($log_2 N$)
4) QS : Quire Size $\big((N\times N)/2\big)$
5) QF : Quire Fixed Point $\big(2\times(N\text{-}2)\big)$
6) OP : Operations – clear, accumulate, read
7) op1 : Input operand1 [N-1:0]
8) op2 : Input operand2 [N-1:0]

Sign Extraction

9) sg1 = op1 [N-1]
10) sg2 = op2 [N-1]

Special Number checking

11) infinity1 = op1 [N-1] AND $\sim\big($OR op1 [N-2:0]$\big)$
12) infinity2 = op2 [N-1] AND $\sim\big($OR op2 [N-2:0]$\big)$
13) InFlag = infinity1 OR infinity2

Two's complement calculation

14) op1Two = sg1 ? ($\sim$ op1 [N-2:0] + 1) : op1
15) op2Two = sg2 ? ($\sim$ op2 [N-2:0] + 1) : op2

Regime Extraction

16) temp_regm1 [N-2:0] = (op1Two [N-2]==1)? ($\sim$ op1Two [N-2:0] + 1) : (op1Two [N-2:0])
17) temp_regm2 [N-2:0] = (op2Two [N-2]==1)? ($\sim$ op2Two [N-2:0] + 1) : (op2Two [N-2:0])
18) R1 = LOD (temp_regm1)
19) R2 = LOD (temp_regm2)
20) K1 [RS+1:0] = (op1Two [N-2] == 1) ? (R1-1) : –R1
21) K2 [RS+1:0] = (op2Two [N-2] == 1) ? (R2-1) : –R2
22) temp_expo1 [N-2:0] = op1Two [N-2:0] << (R1 +1)
23) temp_expo2 [N-2:0] = op2Two [N-2:0] << (R2 +1)

Exponent Extraction

24) eop1 [ES-1:0] = temp_expo1 [N-2:N-ES-1]
25) eop2 [ES-1:0] = temp_expo2 [N-2:N-ES-1]

Mantissa Extraction

26) mantissa1 [N-ES-3:0] = {1, temp_expo1 [N-ES-2:2]}
27) mantissa2 [N-ES-3:0] = {1, temp_expo2 [N-ES-2:2]}
28) SF1 [RS+ES+1:0] = {K1, eop1}
29) SF2 [RS+ES+1:0] = {K2, eop2}

Algorithm 2: Mantissa Multiplication

1) Mulsg = sg1 XOR sg2
2) mantissaPdt [2(N-3-ES)+1:0] = mantissa1 $\times$ mantissa2
3) SFpdt [RS+ES+1:0] = (ES == 0) ? (K1 + K2) : ({K1, eop1} + {K2, eop2})
4) mantissaMag [2(N-3-ES)+2:0] = Mulsg ? ($\sim$ mantissaPdt + 1) : mantissaPdt

B. Mantissa Multiplication:

Function XOR of sign sg1 and sg2 bits produces the product sign. Mantissas 'mantissa1' and 'mantissa2' are multiplied to produce product mantissa. If the product sign is negative (product sign bit=1), the product mantissa is converted to 2's complement form as mentioned in the Algorithm 2. The product scaling factor is calculated by adding decomposed SF1 and SF2 (Algo. 1) from input operands.

C. Product's Mantissa to Quire and Quire Accumulation:

Shift value is calculated by subtracting product SF and bias values (QF+3) from the Quire Size (QS). The product mantissa is made into Quire-size by appending zeros. The product mantissa is converted into fixed-point Quire format by performing arithmetic right shift based on the shift value as depicted in Algorithm 3. If the input operation (OP) is Accumulate, the Quire accumulator adds the product mantissa to the Quire register. Also, if the input operation is Clear, the Quire register value is set to zero.

Algorithm 3: Quire Construction, Accumulation and Clear

1) QuireReg [QS-1:0] = {mantissaMag, $\big($QS-(2$\times$(N-3-ES)+3)$\big)${1'b0}}
2) shiftVal [N-1:0] = QS-QF-SFpdt-3
3) QuireRegTmp [QS-1:0] = QuireReg >>> shiftVal
4) QuireReg [QS-1:0] = (OP == Clear) ? 0 : QuireReg
5) QuireReg [QS-1:0] = (OP == Accumulate) ? (quireReg + QuireRegTmp) : QuireReg

D. Result Scaling Factor, k-value and Mantissa extraction:

Algorithm 4: Quire to Posit Reconstruction

1) sgRec = QuireReg [QS-1]
2) QuireMag [QS-1:0] = (sgRec == 1) ? ($\sim$ QuireReg + 1) : QuireReg
3) QLOD [N-1:0] = LOD (QuireMag)
4) ZeroFlag = $\sim\big($OR QuireMag [S-1:0] $\big)$
5) SF_ r [N-1:0] = QS-QF-QLOD-1
6) SF_final [RS+ES+1:0] = SF_r [RS+ES+1:0]
7) k_r [RS+1:0] = SF_final [RS+ES+1:ES]
8) m_r [RS+1:0] = k_r [RS+1] ? ($\sim$ k_r + 1) : (k_r + 1)
9) e_r [ES-1:0] = SF_final [ES-1:0]
10) mantissa_r [QS-1] = QuireMag << (QLOD+1)

Quire to Posit Reconstruction calculates the final sign, regime, exponent, and mantissa from the Quire register. If the final sign bit is 1 (i.e., negative), then Quire magnitude is 2's complement of Quire register or else Quire register value. From the Quire magnitude value the Position of the leading one is computed. The procedural evaluation is shown in Algorithm 4. Mantissa (mantissa_r) is extracted by left shifting the Quire magnitude and then Quire scaling factor is computed. The least significant es bits of the quire scaling factor are the exponent (e_r), and the remaining are regime values (k_r).

E. Posit Reconstruction, Rounding and Post Processing:

A full bit string is reconstructed with (N-1) bits regime run, exponent, and the remaining Quire magnitude. The most significant N-bits are the resultant Posit value. The last bit and the Guard bit are determined from the reconstructed bit string. The remaining bits are ORed to get the sticky bit. A round bit is generated based on the rounding mode and added to the least significant bit of the reconstructed result. The final Posit result is 2's complemented if the result sign is negative; otherwise, it is unchanged. If the regime value is greater than the maximum possible regime value, the Posit result is made as MAXPOS value. If the regime value is lesser than the minimum possible regime value, the result Posit is made as MINPOS value. Thus, the overflow and underflow exceptions won't occur in the PE computations. If the Quire value becomes 0 or NaR, the Posit result is made as 0 or NaR accordingly. A valid result is given only when any of the Quire operations are carried out on the PE.

IV. Posit Quire Processing Engine - Architecture

Figure 1 depicts the low level hardware architecture of parameterized Posit Quire Multiply-Accumulate unit. This unit accepts two N-bit posit operands, desired rounding mode, Clear, Accumulate, and Read signals as inputs. Based on the operation signal, this unit performs the operations and outputs the value.

Algorithm 5: Posit Reconstruction, Rounding and Post Processing

1) shiftval [N−2:0] = N − m_r − 2
2) PositRecTmp [QS+N−2+ES:0] = (ES == 0) ?
   ((N-2)*(∼ k_r [RS+1]), k_r [RS+1], mantissa_r) :
   ((N-2)*(∼ k_r [RS+1]), k_r [RS+1], e_r, mantissa_r)
3) PositRec [QS+N−2+ES:0] = PositRecTmp << shiftval
4) Sticky = OR PositRec [QS+ES−2:0]
5) Last = PositRec [TQS+ES]
6) Guard = PositRec [QS+ES−1]
7) RoundBit = (rm == 0) ?
   Guard AND (Last OR Sticky) : ((rm == 2) ?
   sgRec AND (Guard OR Sticky) : ((rm == 3) ?
   ∼ sgRec AND (Guard OR Sticky) : ((rm == 4) ?
   Guard : 0)))
8) PositVal [N−2:0] = PositRec [QS+N−2+ES : QS+ES] + RoundBit
9) PositVal [N−2:0] = sgRec ? (∼ PositVal + 1) : PositVal
10) Maxpos = k_r >= (N − 2) ? 1 : 0
11) Minpos = k_r < (2 − N) ? 1 : 0
12) PositResult [N−1:0] = ZeroFlag ?
    {N {1'b0}} : Maxpos ?
    { sgRec, {N-2{∼ sgRec }}, 1'b1} : Minpos ?
    {{N-1{ sgRec }}, 1'b1} : { sgRec, PositVal}
13) PositResultException [4:0] = [4'h0, (Guard OR Sticky)]

The inputs are fed to a Posit decoder which extracts sign, regime values, exponent and mantissa from inputs. The Mantissas of two N-bit operands (N-es-3 bit) are passed to the mantissa multiplier and generate the mantissa product. 1-bit XOR is used to compute the sign of the mantissa product. The extracted regime and exponent bits are fed to the scaling factor calculation block to calculate the scaling factors of two input operands, which are fed to a N-bit adder to calculate the shift value. The output of multiplier is fed to a 2's complement converter which computes the negative of the product mantissa value. The negative product mantissa and product mantissa are fed to a MUX (sign bit as select line), which selects the final product mantissa. Final product mantissa is shifted by shift value and based on operation signal, it performs the computation and stores in fixed point Quire register. If the operation is `accumulate` then adds the shifted final product mantissa with existing Quire register value. If the operation is `clear`, Quire register is set to 0 and the computed final product mantissa is discarded.

If the operation is `read` from Quire register, then computes the Quire magnitude from Quire register. The Quire register value is fed to a 2's complement calculator and selects either 2's complement value or existing Quire value based on the sign of Quire register (MSB bit in Quire register). Leading One Detector calculates the shift required and left shifts the Quire magnitude. The result scaling factor calculation block, calculates the resultant scaling factor. The Posit reconstruction block, computes sign bit, regime bit, exponent bit, and mantissa bit from resultant scaling factor and shifted Quire magnitude. The output of the reconstruction block is fed as input for Rounding and result generation. After performing the specified rounding, N-bit Posit and 5-bit exceptions are generated as outputs.

V. Effects and Evaluation of Quire Accumulation

A. Evaluation Framework and Model

An exhaustive analysis was carried out for Posit(8,0) to measure the efficiency of the Posit(8,0) Multiply Accumulate Unit with quire. The analysis for Posit(16,1), Posit(32,2), and Posit(64,3) Multiply-Accumulate operation with quire also was carried out with the random Posit input that is generated with the help of the Python function. The output of the software PQPE is verified with their

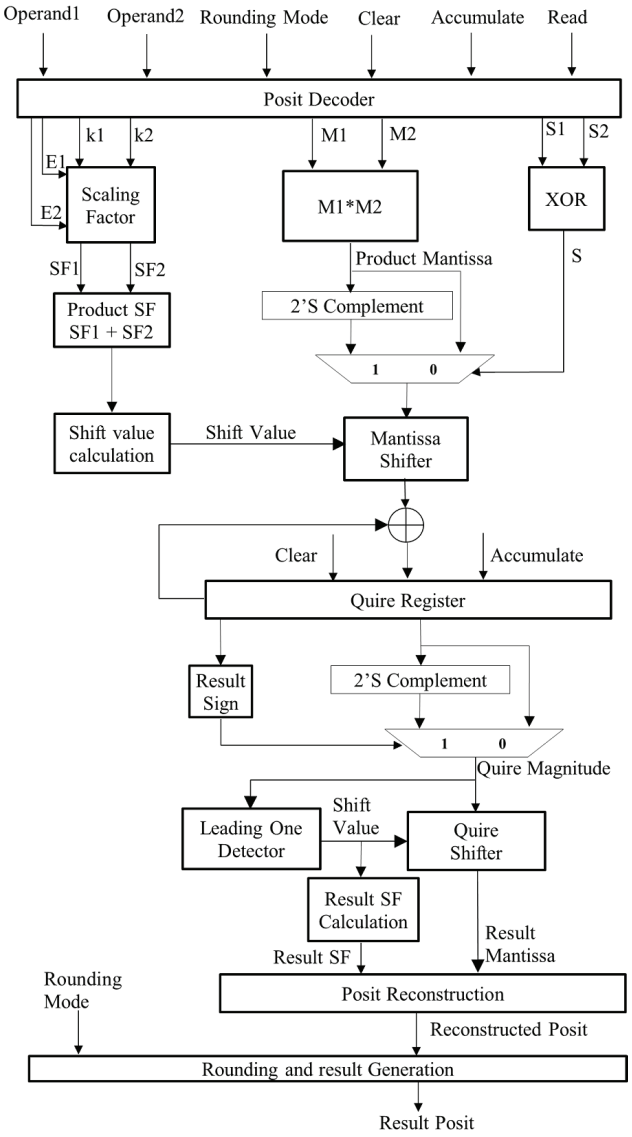


Fig. 1. Hardware Architecture of Proposed Posit Quire Multiply-Accumulate.

corresponding golden result that is generated using Python predefined function and framework of soft-Posit. The testing framework for multiply-accumulate (MAC) and MAC with quire is developed in Python based environment. The framework uses soft-Posit-based Python libraries to generate different Posit inputs. Two different 'N' Posit inputs are generated to verify the efficiency of matrix multiplication operation (iterative multiplication and addition) using quire. Here, N represents the number of iterations (depth of matrix) used to test iterative MAC operation with quire and without quire. An iterative `for` loop is created to perform the iterative operation within a MAC with a generated input, with iterative multiplication-and-addition with quire and without quire. The MAC operation with quire and without quire is performed for Posit(8,0), Posit(16,1), and Posit(32,2). The MAC operation without quire is done using sp.Posit8( ), sp.Posit16( ), and sp.Posit32( ) which are predefined soft Posit Python functions. The MAC operation with quire is done with the help of the Python soft Posit quire function, which is defined as quire8( ), quire16( ), quire32( ), and quire64( ) for Posit(8,0), Posit(16,1), Posit(32,2) and Posit(64,3) respectively. It uses a quire

TABLE I
REFERENCE VALUES IN FLOAT32 FOR ERROR ANALYSIS

# of iterations	500	1K	1.5K	2K	2.5K	3K	3.5K	4K
Ref. values float32	8.933	14.687	19.611	21.259	19.866	67.810	37.970	36.057

fused multiply-addition function that is defined as q.qma(a,b) where a and b inputs. q.clr( ) function is used to clear the quire before getting the final output. The MAC operation's result with quire and without quire is compared with the output obtained after iterative multiplication and addition based on the float32 number system for the generated N number of the same random inputs samples. The reference float32 values for the different number of iterations are shown in TABLE I. Finally, the error is calculated for the output of Posit MAC with quire and without quire for varying N with float32 outputs as references. The obtained error for Posit MAC operations with quire and without quire for different N has been plotted for different Posit number representations.

In Fig. 2, the graphs represents the error of the Posit(8,0), Posit(16,1) and Posit(32,2), which is obtained after MAC operation with quire and without quire. It shows that MAC operation with quire provides approximately zero absolute error (in reference to the outputs from IEEE 754 float32 based PEs as indicated in Table 1) compared to the MAC operation without quire. It can be seen that as the number of iterations increases (N increases) results with quire are more accurate and thus can effectively be used in matrix multiplication operations in neural network processing engines and has been adopted as basis for quire based accumulation in hardware.

VI. PQPE HARDWARE - PERFORMANCE EVALUATION

A. Evaluation Scheme and Model

The verification and validation of the five-stage parameterized Posit Multiply Accumulate Unit with quire is done by creating and developing an environment as illustrated in Section-V-A that compares golden Quire output generated using soft Posit Python packages and Quire output obtained from the PQPE hardware.

B. Accumulation Error Vs No. of Iterations - An Analysis

As illustrated in Section-V-A that as the number of iterations of accumulation increases the quire based accumulation provides better absolute error as compared to no quire and closely meets float32 based system. Similar to the evaluation detailed in Section-V-A which is on software based PQPE, the hardware PQPE also verified for absolute error and the errors have been as per Fig. 2.

C. Neural Network Kernels and Inference Accuracy

The proposed PQPE architecture has been modelled in Python to evaluate inference accuracy for various DNN kernels and datasets. LeNet-5, AlexNet, custom 5 layers and 4 layers CNN models have been adopted and evaluated for inference accuracy with MNIST based hand written digit recognition, CIFAR-10 image classification, and Alphabet (A to Z) recognition datasets as captured in Table-II. Since, the software model of PQPE considers Posit data as inputs and outputs the above datasets and neural nets parameters which are typically in float32 have been converted into Posit representations. All the computations of inference algorithms flow had been transformed as matrix multiplications with quire-based Posit and performed with the PQPE Python model. The obtained results on inference accuracy for various DNN / CNN kernels and datasets have been presented in Table-II. The results clearly indicate Posit (16,1) without quire provides same accuracy as float32 [4] with less hardware resources as in Table-III. P(16,1) with quire brings same accuracy level as P(16,1) without quire and hence quire is not advantageous for larger Posit data widths. P(8,0) with / without quire reaches an accuracy with a loss of 0.8% / 3.28% (LeNet + MNIST) to 3.8% / 10.1% (AlexNet + CIFAR-10) as compared to P(16,1)/float32 which makes us to conclude that P(8,0) with/without quire will be the best option for severely hardware constrained designs/applications which can tolerate inference accuracy loss. Our results on P(8,0) bettered [16] and [20].

TABLE II
INFERENCE ACCURACY OF DNN MODELS - A COMPARISON

Arithmetic Number System	LeNet-5 MNIST (%)	5 LAYER CNN MODEL CIFAR-10 (%)	4 LAYER CNN MODEL Alphabet recog (%)	AlexNet CIFAR-10 (%)
32-bit Floating-Point [4]	98.98	85.00	98.52	80.2
16-bit Floating-Point	96.42	82.10	95.86	78.9
PQPE without quire				
Adaptive Posit (8,0) [16]	92.60	81.10	—	—
Posit (16,1) (This work)	98.98	85.00	98.52	80.2
Posit (8,0) (This work)	95.70	78.00	92.30	70.1
PQPE with quire				
Posit (8,2) [20]	90.25	68.65	—	—
Posit (32,2) (This work)	98.98	85.00	98.52	80.2
Posit (16,1) (This work)	98.98	85.00	98.52	80.2
Posit (8,0) (This work)	98.18	81.01	97.00	76.4

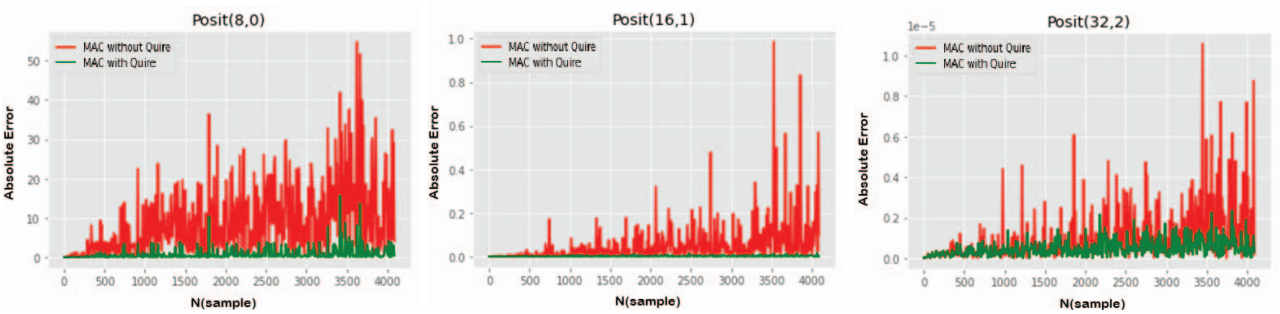
TABLE III
PQPE HARDWARE ENGINE – EVALUATION AND COMPARISON

Design Arithmetic Techniques	LUTs	Registers	DSPs	Freq. (MHz)
32-bit floating point [4]	2226	1062	0	–
32-bit Posit (with quire) [15]	4134	1580	4	85
16-bit Posit (with quire) [15]	1344	407	1	175
8-bit Posit (with quire) [15]	667	205	0	200
Posit(32,2) (with quire) [12]	1031	2618	4	200
PQPE with quire and DSP				
Posit (64,3) (This work)	35606	6638	16	95
Posit (32,2) (This work)	8669	1775	4	125
Posit (16,1) (This work)	1874	528	1	230
Posit (8,0) (This work)	517	175	0	372
PQPE with quire & without DSP				
Posit (64,3) (This work)	41127	6746	0	95
Posit (32,2) (This work)	9813	1806	0	125
Posit (16,1) (This work)	2083	528	0	230
Posit (8,0) (This work)	517	175	0	372

D. Hardware Performance Analysis

The proposed Posit MAC with quire unit was modelled using Verilog HDL, functionally verified with test vectors, synthesized, and implemented targeting Xilinx UltraScale VCU118 FPGA using *Xilinx-Vivado*. The evaluation and comparison of the proposed hardware based Quire Posit MAC (PQPE) with the other published float32 MAC and Posit MAC with quire units is depicted in Table III. Our PQPE units have been realized with 5 stages pipeline for (8,0), (16,1) and with 10 stages pipeline for (32,2) and (64,3). The quire register sizes are 32, 128, 512, and 2048 respectively. The synthesis results show that Posit (64,3) utilized 41127 LUTs and 6746 FFs and achieved a f_{max} of 75MHz. The Posit (32,2), (16,1), and (8,0) utilized 9813, 2083, and 517 LUTs and 1806, 528, and 175 FFs respectively, while targeting the Xilinx UltraScale VCU118 FPGA. The single-stage Posit (8,0) MAC with quire achieved a f_{max} of 148 MHz, whereas 5 stages achieved a f_{max} of 372 MHz as shown in Table-III. The f_{max} achieved by Posit (32,2) and (16,1) are 125MHz and 230MHz respectively.

Ref. [12] presents P(32,2) dot product engine with 512 bits quire realization on Xilinx Virtex7 VX690 FPGA board requiring 1031 LUTs, 2618 registers, and 4 DSPs and a fmax of 200MHz. It has adopted Leading Zero/One Counter (LZC) to calculate the number of regime bits and to obtain the Position of exponent bits and fraction bits. The 512 bits quire accumulation has been realized with Block RAM and URAM resources of FPGA which has drastically reduced the LUTs and FFs utilization, whereas all our realizations presented in this article is fully based on LUTs, FFs and with or without DSPs slices and no memory blocks are used. Ref. [15] has adopted 5 stages pipeline for P8 and P16 and 6 stages for P32 with quire



(a) Posit (8,0) (b) Posit (16,1) (c) Posit (32,2)

Fig. 2. Absolute Error variation with number of iterations for Posit(N, es).

and considered variable exponent sizes realized on a Xilinx Virtex-7 FPGA device (xc7vx485t-2ffg1761). P(8,0) utilizes 667 LUTs and f_{max} of 200 MHz. For P(32,2) the resources increased to 4134 LUTs, and the f_{max} dropped to 85 MHz mainly due 512 bits quire and for P(16,1) with a five-stages pipeline achieving 175 MHz. Ref. [15] has results with and without DSPs slices and adopts LZC. Our realization of P(8,0) is better than [15] in resources and f_{max} considering both are with 5 stages pipeline and 32 bits quire. Our P(16,1) achieves better f_{max} as compared to [15]. Additionally, PQPE (16,1) has used 6.5% less resources as compared to float32 [4] and achieved equivalent accuracy as float32.

VII. Conclusions

This paper proposed a hardware-based Posit Quire Processing Engine (PQPE) supporting P(8,0), P(16,1), P(32,2), and P(64,3) MAC for convolution operations in neural network layers. The hardware engine is enabled with instructions, encoded commands, and responses. The PQPE has been realized as Python model and in Verilog HDL, analyzed for resource utilization and inference accuracy as compared with Posits with and without quire and float32-based PEs. LeNet-5, AlexNet, custom 5 layers and 4 layers CNN models and MNIST, CIFAR-10, Alphabet recognition datasets have been explored for accuracy analysis. P(16,1) without quire provides same accuracy as float32 with less hardware resources. Posit (8,0) with/without quire results insignificant loss of 0.8% / 3.28% (LeNet + MNIST) to 3.8% / 10.1% (AlexNet + CIFAR-10) accuracy as compared to P(16,1)/float32. PQPE (8,0) achieved an f_{max} of 372 MHz and utilize 77.6% fewer hardware resources as compared to float32 at the cost of insignificant accuracy loss. PQPE (16,1) has used 6.5% less resources as compared to float32 and achieved equivalent accuracy as float32. Hence, P(8,0) could be adopted in area-constrained designs with acceptable accuracy loss and P(16,1) will be helpful to achieve more accuracy compromising on higher resource utilization. In comparison with float32 both are well suited to achieve equivalent comparable accuracy with fewer hardware resources.

References

[1] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[2] G. Raut *et al.*, "Data multiplexed and hardware reused architecture for deep neural network accelerator," *Neurocomputing*, vol. 486, pp. 147–159, 2022.

[3] Z. Carmichael *et al.*, "Performance-efficiency trade-off of low-precision numerical formats in deep neural networks," in *Proceedings of the conference for next generation arithmetic 2019*, 2019, pp. 1–9.

[4] B. Zhou *et al.*, "A high-speed floating-point multiply-accumulator based on fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 10, pp. 1782–1789, 2021.

[5] Z. Carmichael *et al.*, "Deep positron: A deep neural network using the posit number system," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1421–1426.

[6] S. Jean *et al.*, "P-fma: A novel parameterized posit fused multiply-accumulate arithmetic processor," in *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*. IEEE, 2021, pp. 282–287.

[7] F. U. D. Farrukh *et al.*, "Power efficient tiny yolo cnn using reduced hardware resources based on booth multiplier and wallace tree adders," *IEEE Open Journal of Circuits and Systems*, vol. 1, pp. 76–87, 2020.

[8] G. Raut, S. Rai, S. K. Vishvakarma, and A. Kumar, "Recon: resource-efficient cordic-based neuron architecture," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 170–181, 2021.

[9] H. Chhajed *et al.*, "Bitmac: Bit-serial computation-based efficient multiply-accumulate unit for dnn accelerator," *Circuits, Systems, and Signal Processing*, vol. 41, no. 4, pp. 2045–2060, 2022.

[10] R. Murillo, A. A. Del Barrio, and G. Botella, "Deep pensieve: A deep learning framework based on the posit number system," *Digital Signal Processing*, vol. 102, p. 102762, 2020.

[11] F. De Dinechin *et al.*, "Posits: the good, the bad and the ugly," in *Proceedings of the Conference for Next Generation Arithmetic 2019*, 2019, pp. 1–10.

[12] J. Chen, Z. Al-Ars, and H. P. Hofstee, "A matrix-multiply unit for posits in reconfigurable logic leveraging (open) capi," in *Proceedings of the Conference for Next Generation Arithmetic*, 2018, pp. 1–5.

[13] J. P. Lim, M. Shachnai, and S. Nagarakatte, "Approximating trigonometric functions for posits using the cordic method," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, 2020, pp. 19–28.

[14] H. F. Langroudi *et al.*, "Positnn framework: Tapered precision deep learning inference for the edge," in *2019 IEEE Space Computing Conference (SCC)*. IEEE, 2019, pp. 53–59.

[15] N. Neves, P. Tomás, and N. Roma, "Dynamic fused multiply-accumulate posit unit with variable exponent size for low-precision dsp applications," in *2020 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2020, pp. 1–6.

[16] H. F. Langroudi *et al.*, "Adaptive posit: Parameter aware numerical format for deep learning inference on the edge," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 726–727.

[17] L. Forget, Y. Uguen, and F. de Dinechin, "Comparing posit and ieee-754 hardware cost," 2021.

[18] A. Poulos, S. A. McKee, and J. C. Calhoun, "Posits and the state of numerical representations in the age of exascale and edge computing," *Software: Practice and Experience*, vol. 52, no. 2, pp. 619–635, 2022.

[19] S. Nambi *et al.*, "Expan (n) d: Exploring posits for efficient artificial neural network design in fpga-based systems," *IEEE Access*, vol. 9, pp. 103 691–103 708, 2021.

[20] G. Raposo, P. Tomás, and N. Roma, "Positnn: Training deep neural networks with mixed low-precision posit," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 7908–7912.

[21] R. Murillo *et al.*, "Energy-efficient mac units for fused posit arithmetic," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, 2021, pp. 138–145.

[22] J. Lu *et al.*, "Training deep neural networks using posit number system," in *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2019, pp. 62–67.