A 22-nm 4.92 TOPS/W end-to-end RNS DNN Accelerator for Edge-AI Devices

Vasilis Sakellariou, *Student Member, IEEE,* Vassilis Paliouras, *Member, IEEE,* Ioannis Kouretas, Hani Saleh, *Senior Member, IEEE,* Thanos Stouraitis, *Life Fellow, IEEE*

***Abstract*—This work presents an end-to-end Residue Numbering System (RNS) Deep Neural Network (DNN) accelerator targeting edge-AI devices. The developed architecture enables translating the advantages of RNS regarding the implementation of a single multiply-add operation into system-level power efficiency gains. This is made possible by the introduction of novel architectural features, such as the amortization of non-trivial RNS operations (base extension, activation and scaling), and the integration of bespoke RNS low-power techniques, such as a clock-gating scheme that exploits the periodic usage of the non-trivial RNS units, and voltage scaling that exploits RNS capability to achieve clock frequency constraints with smaller supply voltages. Systematic analysis of trade-offs between hardware performance metrics (area, power, throughput) of the RNS implementation for various operation scenarios and RNS bases, as well as comparisons against the conventional positional binary implementation, are conducted, guiding the optimal selection of design space parameters. Silicon power measurements on the 22-nm FDSOI prototype chips underpin the theoretical analysis and simulation results, which show considerable benefits of RNS-based DNN processing. These prove that RNS can not only increase the maximum achievable frequency of the arithmetic circuits, but also results in 1.33× more energy-efficient processing, compared to conventional binary counterparts. Compared to the state-of-the-art RNS-based DNN accelerator, the proposed architecture is shown to be 9× more power efficient, reaching a peak power efficiency of 4.92 TOPS/W.**

***Index Terms*—RNS, DNN, AI hardware accelerator, ASIC**

I. Introduction

The upsurge of Artificial Intelligence (AI) models and applications, coupled with the inexorable trend of moving AI inference to resource-constrained edge devices [1], such as smartphones, wearables, microntrollers and sensors, has prompted the development of innovative AI processing architectures [2], [3], [4], [5]. In order to enable hardware systems to keep up with the unprecedented computational requirements introduced by modern AI models, new computing paradigms that broaden our capacity of extracting performance from the available hardware resources need to be sought.

Computer arithmetic has always been at the forefront of designing efficient computing architectures. It provides a fundamental way to increase the performance of digital processing systems by efficiently utilizing the available bits and logic gates, according to the needs of a specific application. Thus, employing alternative number representations naturally presents appealing opportunities to enhance the performance of AI-processing systems. Among various alternative numbering systems, the Residue Numbering System (RNS) stands out as a promising choice for AI inference due to its inherent digit (residue)-level parallelism. This work leverages the high-speed arithmetic properties of RNS to design an energy-efficient end-to-end RNS DNN processor. It proposes a number of innovative architecture and circuit-level optimizations which allow the digit-level processing efficiency of RNS to be translated into system-level performance improvements. The contributions of this work can be summarized as follows:

- A silicon-implemented end-to-end RNS accelerator with a balanced modulus base and a low number of non-trivial RNS arithmetic units (scaling, activation function, base extension). The proposed architecture amortizes the usage of these units over a large number of multiply-add operations and thus minimizes their overhead.
- The integration of RNS-tailored, low-power design practices which allow to translate the efficiency of the RNS-based MAC operation to system-level power efficiency gains. More specifically, the infrequent usage of the costly non-trivial RNS operations is exploited by hierarchically clock-gating the relevant circuits. Furthermore, the high-speed capabilities of RNS are exploited so as to allow a reduction of the processing logic supply voltage with respect to the desired timing constraints. Overall, a $1.33\times$ reduction of processing power compared to the conventional binary counterpart is achieved.
- The design of an RNS-friendly activation unit, as well as a novel overflow handling technique that allows mitigating accuracy loss in applications that require higher dynamic range with minimal impact on performance.
- A systematic analysis of trade-offs between hardware performance metrics (area, power, frequency) of the RNS implementation for various operational scenarios and moduli, which guided the optimal selection of RNS design space parameters.

The remaining of the paper is organized as follows: Section II introduces the basic properties of RNS and provides an outline of its usage in DNN accelerators. Section II-B highlights the motivation for employing RNS and quantifies its efficiency compared to conventional hardware implementation of the multiply-add operation. Section IV uses a top-down approach to describe the developed end-to-end RNS archi-

Vasilis Sakellariou, Hani Saleh, and Thanos Stouraitis are with the Electrical Engineering and Computer Science Department of Khalifa University, Abu Dhabi, UAE

Vassilis Paliouras and Ioannis Kouretas are with the Electrical and Computer Engineering Department, University of Patras, Greece

tecture, highlighting its innovative features. Section V reports silicon measurement results on the fabricated chips, evaluates the system in terms of accuracy and conducts comparisons with state-of-the art DNN processors.

II. Background and related Work

A. RNS basics

RNS is an alternative numbering representation, which maps a given integer X to a tuple of residues with respect to a modulus set $\mathcal{B} = \{m_1, m_2, \ldots, m_n\}$:

$$X \mapsto (x_1, x_2, \ldots, x_n),\ x_i = \langle X \rangle_{m_i}, \tag{1}$$

where $\langle \cdot \rangle_m$ is the modulo-m operator. $\mathcal{B}$ is the *base* of the representation, while the product of the moduli of the base defines the dynamic range $R = \prod_{i=1}^{N} m_i$ of the representation. If the moduli are *co-prime*, each integer inside the range $[0, R)$ has a unique RNS representation. Inverse transformation to the original integer representation of X can be realized by means of the *Mixed Radix Conversion* or the *Chinese Remainder Theorem* [6]. Due to the properties of the modulo operation, addition and multiplication can be executed in parallel for each residue channel, i.e. without inter-channel propagation of information. Suppose $a = (a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_n)$ then

$$a \oplus b = (\langle a_1 \oplus b_1 \rangle_{m_1}, \langle a_2 \oplus b_2 \rangle_{m_2}, \ldots, \langle a_n \oplus b_n \rangle_{m_n}), \tag{2}$$

where $\oplus$ can be either the addition or the multiplication operator. This property results in a considerably more efficient implementation of the multiply-add operation (MAC). However, operations like division and comparison, are harder to implement in the RNS domain and introduce overhead.

B. RNS in DNN accelerators

The inherent efficiency of RNS regarding the implementation of MAC operations has motivated its usage in Deep Neural Network (DNN) accelerator design. In this section we provide an overview of RNS-based DNN accelerators in literature, classifying the approaches into two categories: (a) partially RNS-based and (b) end-to-end RNS architectures, depending on how they handle non-trivial operation between successive layers.

1) Partially RNS-based systems: A typical strategy utilized in RNS-based DNN setups involves conducting all multiply-add operations of a convolutional layer within the RNS domain. Subsequently, partial results are converted to a standard positional binary form [7], [8], [9], [10]. This interim outcome allows for the computation of non-linear activation functions (such as ReLU, tanh, softmax). Afterwards, the outcomes are re-converted to RNS format for input into the subsequent layer. An RNS TPU (Tensor Processing Unit) is proposed and reported to perform a 32×32 fixed-point matrix multiplication up to $9\times$ more efficiently than a conventional matrix multiplier [7]. The RNS usage is also extended to convolutions, with a 7.86%–37.78% reduction of the hardware costs of a single convolutional layer compared to two's-complement implementation, depending on the RNS base [9]. A variant of the RNS, called the *Nested RNS* (NRNS), utilizes a recursive decomposition of the residue channels into smaller ones [8].

2) End-to-end RNS systems: While the above circuits manage to achieve some performance gains in the implementation of a single convolutional layer, they require significant amounts of additional hardware to perform conversions. More recent approaches focus on overcoming the difficulties of performing operations such as sign detection, comparison, and scaling, which are usually required after multiplication, directly in the RNS domain. Mechanisms for dealing with this problem through a fully RNS-based architecture have been proposed in [11]. The comparison of two RNS numbers is implemented by calculating auxiliary partitioning functions [12]. A base extension operation takes place once before each multiplication to ensure that the product lies within the dynamic range and then again before the accumulation. Up to 61% reduction in energy consumption compared to the Eyeriss [3] accelerator is achieved. A method to drastically reduce the number of multiplications in DNN RNS-based accelerators is also proposed [13]. It utilizes a modified hardware mapping of the convolution algorithm where the order of operations is rearranged, leveraging the increased number of common factors inside the weight kernels during convolution. A complete multiplier-free RNS accelerator utilizing this approach is also developed [14]. RNS has been also utilized in the design of in-memory computing (IMC) systems [15], [16]. This work addresses the shortcomings of existing state-of-the-art RNS DNN architectures (Sec. V-F) by alleviating the overhead of non-trivial RNS operations, through innovative usage of activation function, scaling and overflow control techniques while maintaining a small maximum word-length among the residue channels, and presents the first silicon-implemented RNS-based DNN accelerator.

III. Efficient multiply-add using RNS

The decomposition of a large computation into smaller independent channels, an inherent property of RNS, naturally leads to a decrease in the delay of the arithmetic circuits. Long carry propagation chains are eliminated and arithmetic circuits can operate at higher frequencies and/or with reduced power dissipation. For example, a 32-bit multiplication consumes more than $15\times$ more energy than an 8-bit one [17]. Since the power consumption of a multiplication increases approximately quadratically with the number of bits, replacing a kn-bit multiplication with k n-bit multiplications can result in significant energy cost reduction. RNS naturally enables this arithmetically equivalent decomposition. Providing that the dynamic range of the RNS system is sufficient for a given application, and that the overhead of the modulo operator and the cost of implementing other (non-trivial in RNS) functions can be kept small RNS can provide significant gains in terms of power efficiency.

A. RNS base selection

The selection of the RNS base significantly influences the system's complexity. The precision requirements of the specific application and network dictate the dynamic range of the RNS representation. Specific choices of moduli simplify the implementation of arithmetic circuits by reducing the

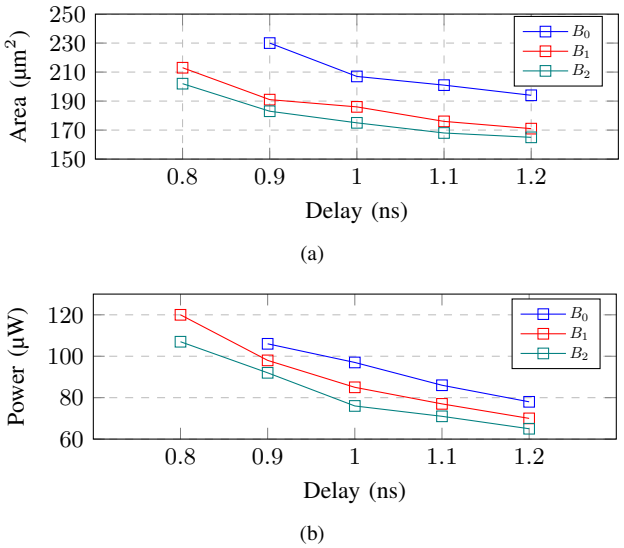


Fig. 1. (a) Area-delay and (b) power-delay plot for synthesized MAC units using RNS bases with 3, 4 and 5 moduli and a 22-nm library.

overhead of the modulo operation. In our design, we limit the RNS base to moduli of the form 2^k and $2^k \pm 1$, allowing for the use of simplified adder and multiplier designs. The modulo-2^k MAC operation is straightforward, as the mod operator effectively limits operations to the k least significant bits. For modulo-(2^k-1) arithmetic, the end-around-carry technique is applicable, exploiting the fact that $2^k \bmod (2^k-1) = 1$. Similarly, for moduli of the form (2^k+1), diminished-1 multiplication can be employed [18], reducing all operands by 1 and utilizing inverted end-around-carry logic.

In order to investigate the trade-offs between the size of the RNS base and the size of the individual RNS channels, various selections of RNS bases are explored, using three, four, and five moduli and providing a range of approximately 16, 17, and 18 total bits, respectively. These bases are:

- $\mathcal{B}_0 = \{31, 32, 63\}$ (approx. 16 bits): This base has a simple sign detection mechanism [11]. The small number of channels comes at the cost of one of them being a relatively large channel (6 bits), leading to a design of larger complexity and critical path.
- $\mathcal{B}_1 = \{7, 15, 31, 32\}$ (approx. 17 bits): By replacing the larger channel 63 with two smaller ones (15 and 7), the critical path of the design becomes that of the 5-bit channel (31), resulting in decreased latency. This can also lead to a reduction in the area and power of the processing elements, even though the total number of bits increases. This is due to the approximately quadratic scaling of a multiplier's area with the number of bits ($3^2 + 4^2 < 6^2$).
- $\mathcal{B}_2 = \{3, 5, 7, 31, 32\}$ (approx. 18 bits): This choice decomposes channel 15 into two smaller channels, namely 5 and 3, to take further advantage of the above observations.

RNS bases with a larger number of smaller-bitwidth channels are found to perform better, both in terms of latency (largest channel is the bottleneck) as well as area/power, as explained by the quadratic scaling of multiplier complexity. To illustrate this, power and area plots of the synthesized MAC units using $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2$ are presented in Figs. 1a and 1b. The cost of implementing operations that require combining information from different channels (i.e., division, sign detection) typically becomes higher as the number of channels increases. However, the cost of these operations in DNN processing is amortized across a considerably higher total number of MAC operations and thus becomes negligible.

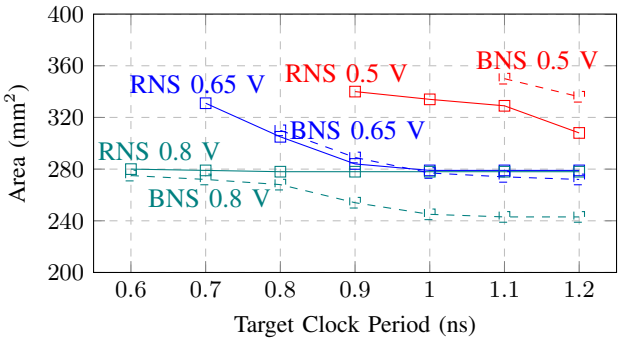
The developed architecture targets 8-bit integer quantized (INT8) convolutional neural network (CNN) models, as these have been shown to be capable of recovering the original floating-point accuracy, either through post training quantization techniques [19] or quantization aware training [20], in state-of-the-art vision models like ResNet [21], MobileNet [22] and VGG [23]. In order to avoid any RNS-induced accuracy penalty due to the limited dynamic range during accumulation and to be able to match the INT8 model accuracy without requiring any further finetuning, we choose to replace the modulo-3 in $\mathcal{B}_2$ with a modulo-33 channel, leading to a dynamic range of ≈ 20.1 bits. Hence, we select the 5-channel RNS base $\mathcal{B}_c = \{5, 7, 31, 32, 33\}$ for the complete accelerator architecture. An overflow protection unit is also utilized to support applications that require even higher dynamic ranges and is presented in Section IV-C3.

B. *RNS* vs. *BNS MAC comparison*

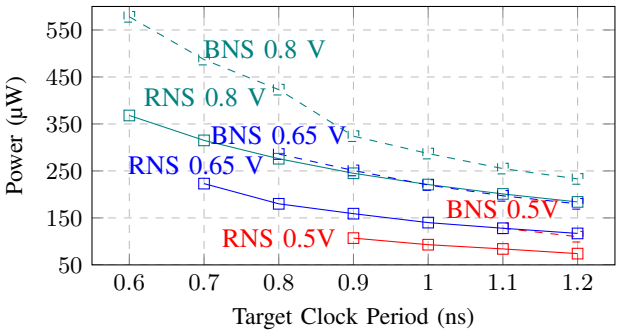
In order to compare the processing element (MAC) implementations, both the 5-channel RNS ($\mathcal{B}_c$) processing element (PE) and conventional positional binary system (BNS) MAC units of an equivalent dynamic range are synthesized, using a GlobalFoundries 22-nm FDSOI library. To comprehensively evaluate their performance (power and area efficiency) the designs are synthesized targeting various clock periods and using three supply voltages. Results are visualized in Fig 2. RNS holds a distinct edge over conventional BNS, particularly when the target clock period or supply voltage is decreased. This is due to the inherent speed of RNS; the independent processing of the residue channels breaks long carry propagation chains leading to naturally faster circuits. In contrary, in the case of BNS, the synthesis tool must utilize more complex multiplier and adder structures, or cells with greater driving strength. Results show that RNS MAC is up to $1.57\times$ more power-efficient even with a smaller area. The inherent speed of the RNS arithmetic circuits can be further exploited by taking advantage of the fact that the supply voltage has a quadratic effect on a circuit's power consumption, while it only linearly affects its delay. This means that we can trade speed for power reduction, by using a lower supply voltage for powering the RNS arithmetic circuits, which can achieve the same frequency constraint with a smaller supply voltage compared to the BNS counterparts. For example, an RNS-based processing element can achieve a frequency of 1.42 GHz (0.7 ns) with a 0.65 V supply, while the BNS counterpart would require 0.8 V. This translates to a $2.1\times$ energy reduction. Motivated by these observations we designed an end-to-end RNS accelerator which manages to take advantage of this inherent RNS efficiency.

IV. Proposed RNS-based DNN Architecture

This section describes both circuit and architecture-level techniques and optimizations employed for the design of an



(a) Area - Target Clock Period



(b) Power - Target Clock Period

Fig. 2. Area (a) and power (b) *vs.* target clock period for RNS and BNS PEs and 0.5 V, 0.65 V and 0.8 V supply voltages.

efficient end-to-end RNS DNN accelerator. The developed architecture targets energy-efficient real-time inference on edge devices. It enables translating the RNS MAC efficiency, which was quantified in the previous section, into system-level performance gains by minimizing the overhead of complex RNS operations, like division, comparison and base extension, as well by utilizing RNS-tailored low power techniques.

A. Top-level architecture

The implemented accelerator consists of a set of RNS processing cores, which perform the elementary MAC operations and a set of activation scaling units (A/S) units which apply the activation functions and scale the result back to the desired range. A high-level representation of the architecture is given in Fig. 3. The on-chip memory consists of four hierarchical memory components: a feature-map memory (FMEM), a weight memory (WMEM), a block buffer and a border buffer. A detailed description of the functionality and organization of these components, is given in Section IV-D. A feature-map router and a memory controller provide the necessary feature-map inputs to the cores.

1) Dataflow: The convolution operation translates into a series of nested loops (Alg. 1). In the implemented architecture we choose to unroll the loop that corresponds to the output channels (different filter kernels), referred to as loop-K, over $K = 16$ parallel processing cores, which all receive the same input. Inside each core, the computation of a $N_r \times N_c$ block is mapped to N_r rows and N_c columns of the processing array. Therefore, the loops that iterate the input feature-map

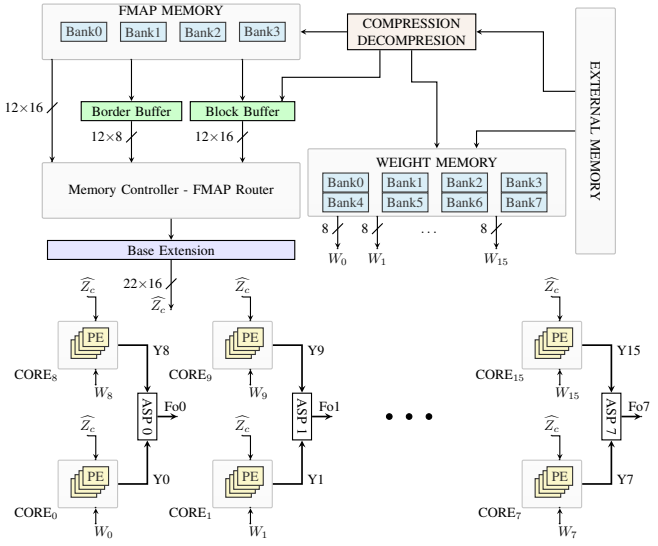


Fig. 3. General architecture. A block of 16 12-bit feature-maps is fetched from FMEM and/or border and block buffers, base-extended to obtain a 22-bit representation of each feature-map $\widehat{Z_c}$ and broadcast to all cores, while 16 weights W_i, $i \in \{0, \ldots 15\}$, are fetched from WMEM and directed to each of the cores. Each pair of cores shares one A/S unit, which serially selects their outputs, applies the activation function and generates the final output Foj, $j \in \{0, 1, \ldots, 7\}$of the current layer.

Algorithm 1 Convolution loops

```
for c_o ← 0 to C_out do                                   ▷ Loop-K
  for x ← 0 to X do                                       ▷ Loop-B.x
    for y ← 0 to Y do                                     ▷ Loop-B.y
      for c_i ← 0 to C_in do
        for f_x ← −F_X/2 to F_X/2 do
          for f_y ← −F_Y/2 to F_Y/2 do
            O[c_o][x][y] += I[c_i][x+f_x][y+f_y] × W[c_i][f_x][f_y]
```

dimensions are also (partially) unrolled. This corresponds to an output-stationary dataflow, where outputs are accumulated inside each processing element. All processing elements inside a core operate on the same weight. Therefore, in this dataflow, weights are reused $N_r \times N_c$ times and feature maps are reused K times (feature maps are also reused inside the shift registers, depending on the kernel size). The loop that iterates the input blocks is denoted as Loop-B.

If $C_{\text{out}} > K$, then Loop-K cannot be completely unrolled and it has to be tiled into chunks of size K (number of cores), thus multiple iterations of mapping output channels to pro-

Algorithm 2 In Mode_0 (left), Loop-K is the innermost, while in Mode_1 (right), Loop-B is the innermost. N_B and N_K denote the number of input feature-map blocks and weight kernel sets (output channels/number of cores).

```
for i ← 1 to N_B do                          for i ← 1 to N_K do
  LOAD(block_i)                                LOAD(kernel_i)
  for j ← 1 to N_K do                          for j ← 1 to N_B do
    LOAD(kernel_j)                               LOAD(block_j)
    CONV(block_i, kernel_j)                      CONV(block_i, kernel_j)
```

cessing cores might be required. In the implemented design, the execution order of Loop-K and Loop-B is programmable. Specifically, two modes of operation are possible: *mode_0*, where Loop-K is the innermost, and *mode_1*, where Loop-B is the innermost. If all layer weights and inputs fit in the on-chip memory, the loop order does not effect the system's performance. If, however, parts of the input feature-map or weights have to be transferred from an off-chip memory, then the execution order of these loops is significant: During *mode_0*, a 2D feature-map block can remain inside the on-chip memory until Loop-K is complete, while weights must be updated for every Loop-K tile. This means that if the entire feature-map tensor does not fit in the on-chip memory while all the weight kernels do, then *mode_0* is preferable since the largest transfer cost (off-chip to on-chip) associated with the feature-map blocks is only paid once. In the opposite case, when the entire feature-map tensor fits on chip while weights do not, *mode_1* reduces transfer cost and is thus preferred.

2) Feature map base extension and routing: A 2D slice Z_c of the input feature map corresponding to the c-th channel is fetched from memory and directed to the feature-map (FMAP) router at each step. The main controller of the system generates the necessary signals for the implementation of the desired operation, based on the layer type and size parameters. These signals include addresses and read/write enable signals for the memories, shifting and padding signals for the feature-map router, and various control signals for the RNS processing core operation. The feature-map router is a collection of shift registers that can perform either vertical or horizontal shifting and is used to implement the sliding window functionality required during convolution and to increase data reuse. This design offers flexibility and allows the efficient mapping of any convolutional layer shape to the PE processing arrays. It should also be noted that the control and routing units are independent of the selection of moduli, and thus they could remain unchanged if a different RNS base were to be used.

An RNS base with five channels $\mathcal{B}_c = \{5, 7, 31, 32, 33\}$ is employed for performing the convolution. Weights are stored in a two-channel 8-bit equivalent RNS base $\mathcal{B}_w = \{7, 32\}$, and feature maps in a three-channel, 12-bit equivalent base $\mathcal{B}_f = \{7, 16, 31\}$. This scheme avoids the additional memory storage usage that would be required if all five channels of the representation were stored. *Base Extension* units are used to obtain the residue values for the remaining channels that are required for the accumulation of partial products. The resulting 22-bit representation of the input slice $\widehat{Z_c}$ is then broadcast to all cores to compute the output of the convolutional or fully connected layer, which is stored in FMEM memory, potentially after applying a pooling operation. The most common base extension method is the Szabo-Tanaka method [24]. This method is based on an intermediate Mixed Radix (MR) representation, where an RNS number $X = (x_1, x_2, \ldots, x_n)$ is expressed as $X = a_1 + a_2 m_1 + \cdots + a_n m_1 m_2 \cdots m_{n-1}$, where m_i are the moduli of the RNS and a_i are the MRC coefficients, which are obtained sequentially [25]. This method has the advantage that it only involves small-width operations and can be easily pipelined for increased throughput.

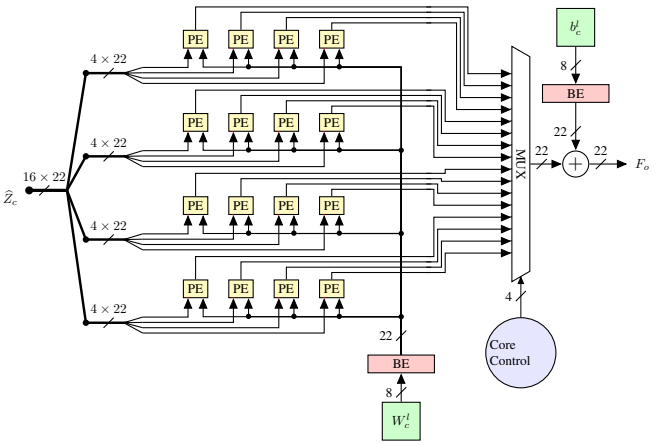


Fig. 4. RNS Processing Core. It consists of a 4×4 PE array, where each PE receives the same weight but different feature-map values.

B. RNS Processing Core

The main component of the accelerator is the RNS Processing Core, shown in Fig. 4, which implements the elementary arithmetic computations for the various layer types. Each core consists of set a of 16 Processing Elements (PEs) organized in a 2D array with $N_r = 4$ rows and $N_c = 4$ columns. The convolution of the input feature-map block with the c-th filter (kernel) takes place inside the k-th core, where $k = c \bmod K$. Thus, each PE receives two 22-bit inputs corresponding to the five-channel RNS representation: a feature-map value (fetched from the input data router and broadcast to all cores) as well as a weight value which is fetched from the weight memory and base-extended. All PEs share the same weight, while each of them receives a different feature-map value $Z_c^{i,j}$ corresponding to a pixel at position (i, j) in the input.

The processing elements perform all the multiply-accumulate operations to obtain a single output pixel value. This requires the accumulation of $F_x \times F_y \times C_{\text{in}}$ partial products, where F_x, F_y, C_{in} are the filter's width, height and number of input channels, respectively. Therefore, at every $F_x \times F_y \times C_{\text{in}}$ cycles a 4×4 output block is calculated by each core. Once the accumulation process is complete, PEs are serially selected and their output is sent to the Activation/Scaling (A/S) unit (details in Section IV-C).

Each RNS processing element (PE) consists of five modulo MAC units, corresponding to $\mathcal{B}_c = \{5, 7, 31, 32, 33\}$. As explained in Section III-A, the base moduli are restricted to numbers of the form 2^k (i.e., modulo 32), $2^k - 1$ (moduli 7,31) and $2^k + 1$ (moduli 5,33), in order to simplify hardware implementations. All modulo MAC units implement a fused multiply-add operation, using array multipliers and end-around carry logic, in the case of $2^k - 1$, while for $2^k + 1$ channels we modify the multiplier proposed in [18] to receive a third operand for the fused addition.

C. Activation/Scaling (A/S) unit

The Activation Scaling (A/S) Unit performs three functionalities: (1) It applies the non-linear activation function to the result of the accumulation. Since the prototype targets

CNN applications, it only includes a simplified activation unit which implements the ReLU function; (2) It scales the result back to the 12-bit three-channel ($\mathcal{B}_f$) range that is used for storing feature-maps; (3) It periodically checks for overflow by extending the dynamic range to a six-channel representation, as explained in Sec. IV-C3.

In RNS, unlike a conventional representation, activation and scaling are generally not trivial to implement. This makes A/S the primary overhead component of an RNS DNN accelerator. One of the major advantages of the proposed architecture is that, unlike other state-of-the-art architectures [11], it amortizes the number of A/S units over a larger number of PEs, and thus minimizes its overhead. One A/S unit is shared by two cores (32 PEs), as shown in Fig. 3. This is possible because the activation and scaling operation only need to be applied to the final result after the accumulation of multiple partial products, $F_x \times F_y \times C_{\text{in}}$ in particular. Since C_{in} ranges from 64 to 1024 for most layers of state-of-the-art CNN models, these A/S units are actually utilized only a small fraction of the total layer execution time. By overlapping the parallel MAC computations with the serial computations of the activation function in a smaller number of A/S units, the processing throughput is not affected and the PE utilization remains close to 100%. Moreover, when the A/S units are not utilized their dynamic power consumption can be eliminated by using clock-gating cells. As an example, consider a layer that takes as input a $(64\times64\times256)$ feature-map and performs a 3×3 convolution. According to the employed dataflow, the calculation of each output feature-map value requires $3 \times 3 \times 256 = 2304$ cycles. Only at the end of these calculations, the 32 outputs of the two cores will be sent to the A/S unit that they share. This means that during every 2304 cycles the A/S units will be utilized for 32 cycles, or 13% of the time. During the inactive period, the dynamic power of the A/S units is eliminated through clock-gating.

1) RNS activation function: The implementation of the ReLU function practically translates into a comparison with zero. In a RNS representation, numbers in the range $0 < X < R/2$ are positive, whereas numbers in the range $R/2 \leq x < R$ are negative, where R is the dynamic range of the representation. Sign detection is a difficult operation in RNS, and typically requires the conversion to a conventional representation. Similarly, magnitude comparison of two RNS numbers, which is required for the MaxPooling layers, is also difficult to directly implement. Algorithms for particular moduli sets [12], or more complex general comparison algorithms that can eliminate the overhead of the conversion have been proposed [26]. If the choice of moduli is restricted to some specific bases, simple and efficient algorithms have been reported for sign detection [27] and, consequently, comparison.

In Algorithm 3, a method for obtaining the sign of a RNS number for bases with an even modulus is presented. Assuming, without loss of generality, that the even modulus is m_{n-1}, the algorithm maps the input $X = (x_0, x_1, \ldots, x_{n-1})$ to the *nearest* number (smaller than X) of the form $X' = (0, 0, \ldots, x'_{n-1})$ and uses the value of x'_{n-1} to decide the sign. To do this, we perform base extension of the first $n-1$ channels to get k, which is the last channel offset between X and X' and is given by

$$k = \left(X \bmod \prod_{i=0}^{n-2} m_i \right) \bmod m_{n-1}. \tag{3}$$

Then we obtain $x'_{n-1} = (x_{n-1} - k) \bmod m_{n-1}$, and use this value to determine whether the input lies in the lower (positive) or upper (negative) half of the dynamic range, by means of a look-up table (LUT). The sign unit implemented in hardware operates in a six-channel extended RNS base $\mathcal{B}_e = \{5, 7, 17, 31, 32, 33\}$. The need for the additional channel (modulo 17) is explained in paragraph IV-C3. While the prototype chips only support the ReLU function, this method can be generalized to implement other activation functions, such as tanh and sigmoid using piece-wise linear approximations and Algorithm 3 to obtain the interval that the input lies [28].

2) Scaling: A division operation is required after the activation function application, to bring the accumulated output feature-map entries back to the $\mathcal{B}_f$ dynamic range. This operation corresponds to the truncation of the lowest part of the accumulation result in a conventional representation. Division in RNS can be significantly simplified when the divisor is one of the moduli [29]. Here, we use a generalized

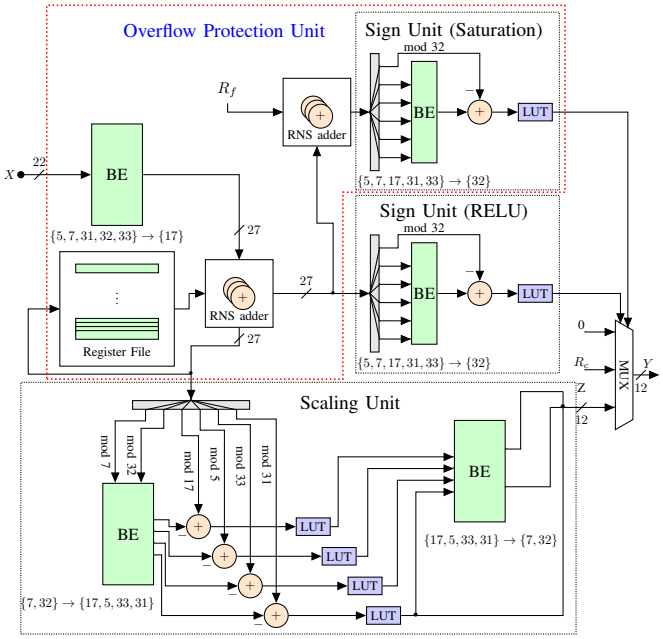


Fig. 5. Activation/Scaling Unit. It consists of a sign unit which applies the ReLU function to the 12-bit input partial sum, a scaling unit and an overflow protection unit for increased dynamic range operation. It produces the 12-bit output feature map for of each of the PEs that it serves.

Algorithm 3 Sign Detection

Require: $X = (x_0, x_1, \ldots, x_{n-1})$
Ensure: sign s of X
1: $k = \text{BE}(m_{n-1}, (x_0, x_1 \ldots, x_{n-2}))$ ▷ base extension of the first $n-1$ channels to m_{n-1}
2: $x'_{n-1} = (x_{n-1} - k) \mod m_{n-1}$ ▷ computes x'_{n-1}
3: $s = \text{LUT}(x'_{n-1})$ ▷ lookup table to get the sign

version of this method, where the divisor is a product of a subset of the RNS base moduli. The implementation of this method involves only small table (1-channel wide) lookups and a standard base extension process. In this implementation, the two-channel RNS-base $\mathcal{B}_w = \{7, 32\}$ is used for storing the network weights. This means that an original real weight $w \in (-1, 1]$ is mapped to the range $(-112, 112]$, and therefore the activation (feature-map outputs) should be divided by $(112 = 7 \cdot 32/2)$. This is done by the scaling unit of Fig. 5.

3) Overflow protection unit: In order for the proposed RNS-based system to handle applications that require a higher dynamic range than the 20-bit equivalent range $\mathcal{B}_c$ offers, an overflow protection unit has been added. The goal is to reduce hardware cost by maintaining a reduced number of non-trivial RNS units (mainly base extensions), while mitigating any accuracy loss due to overflow. The outputs of the PEs (output feature-maps) can be periodically extended from $\mathcal{B}_c$ to a six-channel RNS base $\mathcal{B}_e = \{5, 7, 17, 31, 32, 33\}$. The result can be then checked against R_c, the maximum representable value of $\mathcal{B}_c$, and if it exceeds that, then the result is truncated to R_f (maximum value of $\mathcal{B}_f$). More specifically, the calculation of the convolution is split in the input channel dimension: The accumulation of C_{split} feature-map channels ($C_{\text{split}} < C_{\text{in}}$) in the five-channel base without overflow checking inside each PE is performed, where $n = C_{\text{split}} \times F_x \times F_y$ partial products are added together. Then, one base-extension unit (for all 32 PEs of two cores) adds another channel (modulo 17) and a six-channel RNS addition with the previous result takes place.

The intuition behind the design of this unit lies on the fact that the probability of overflow during the computation of the dot product of two vectors increases as the length of the vectors increases. Consider the computation of the dot product between a vector of weights $\mathbf{W}$ and a vector of feature-maps $\mathbf{X}$ with L elements, following Gaussian distributions, i.e., $w \in \mathbf{W} \sim \mathcal{N}(\mu_w, \sigma_w^2)$ and $x \in \mathbf{X} \sim \mathcal{N}(\mu_x, \sigma_x^2)$. The probability of overflow P_{ovf}, given the dynamic range (scale factor) of weights R_W and feature-maps R_X, and the total dynamic range R, can be approximated using the central limit theorem as

$$P_{\text{ovf}} = 1 - \text{erf}\left(\frac{x_{\text{ovf}} - \mu_p}{\sqrt{2}\sigma_p/\sqrt{L}}\right), \tag{4}$$

where $x_{\text{ovf}} = \frac{R}{2L}$, $\mu_p = \mu'_x\mu'_w$ (mean of the element-wise products), $\sigma_p = \sqrt{(\sigma'^2_w + \mu'^2_w)(\sigma'^2_x + \mu'^2_x) - \mu'^2_w\mu'^2_x}$ (standard deviation of the element-wise products), and $\mu'_x = R_X\mu_x$, $\sigma'_x = R_X\sigma_x$, $\mu'_w = R_w\mu_x$, $\sigma'_w = R_w\sigma_w$. By plotting P_{ovf} as a function of C_{in} for weight distributions of different variances ($\mu_w = 0.001$, $\sigma_w \in \{0.02, 0.035, 005\}$) and a fixed feature-map distribution ($\mu_x = 0.1, \sigma_x = 0.5$), using typical values for mean and standard deviation according to the distribution of benchmark CNN parameters, we observe that the probability of overflow quickly increases with C_{in}, especially as the variance of the weights (or equivalently the variance of the feature-maps) increases, making impossible to accumulate all $L = F_x \times F_y \times C_{\text{in}}$ ($= 9 \times C_{\text{in}}$ for 3×3 convolution) channels using $\mathcal{B}_c$ without a significant error due to overflow. We can use this analysis to select the maximum C_{split} value in order to achieve an overflow probability below a certain threshold, less than 1% for example, as indicated in Fig. 6. This probability threshold can be tuned based on the employed NN model and by using a small calibration subset of a given task. When the distribution of the network parameters is narrower, we can accumulate a larger number of partial products before sending the result to the overflow unit, thus reducing the average power consumption of this unit. However, even with high variance distributions, we can still achieve a low utilization ratio of the overflow units, and save dynamic power using clock-gating

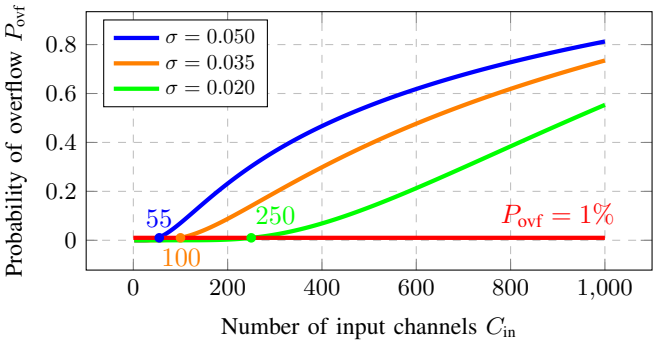


Fig. 6. Probability of overflow against the number of input channels C_{split} processed without overflow checking. The maximum value of C_{split} to ensure a probability of overflow less than 1% is indicated.

D. Memory hierarchy and organization

The on-chip memory consists of four hierarchical memories:

1) Feature-map Memory (FMEM): The 192 KB feature-map memory FMEM is implemented using SRAM macros and is organized into 48-bit wide banks. The output of the current layer is temporarily stored in this memory, in a row-interleaved fashion (Fig. 7). Only three of the channels ($\mathcal{B}_f = \{7, 16, 31\}$) are used for storing feature-map values, hence each FMEM word saves four 12-bit feature-map values. A 2D block, denoted as Z_c, where c is depth (channel) index, is loaded from FMEM at each processing step and broadcast to all cores, after being base extended,

2) Weight Memory (WMEM): The 256 KB weight memory WMEM is implemented using SRAM macros and organized into 64-bit wide banks. Each WMEM word stores eight weights to be read by eight of the cores. Two channels ($\mathcal{B}_W = \{7, 31\}$, 8-bits) are used for storing weights, while the remainder of the channels are added on the fly using base extension.

3) Border Buffer: A 36 KB border buffer implemented using register file macros is also utilized. As described in Section IV-A1, the input is processed in a block-wise manner, where block refers to a 2D feature-map slice. In order for a core to calculate a 4×4 output slice $O[x : x + 4, y : y + 4]$, where x is the row index and y is the column index, it requires pixels in the input slice $I[x - F_x/2 : x + 4 + F_x/2, y - F_y/2 : y + 4 + F_y/2]$. In other words, due to the sliding convolution window and depending on the kernel dimensions, at each step the memory system needs to provide feature-map inputs corresponding to the current block coordinates, but

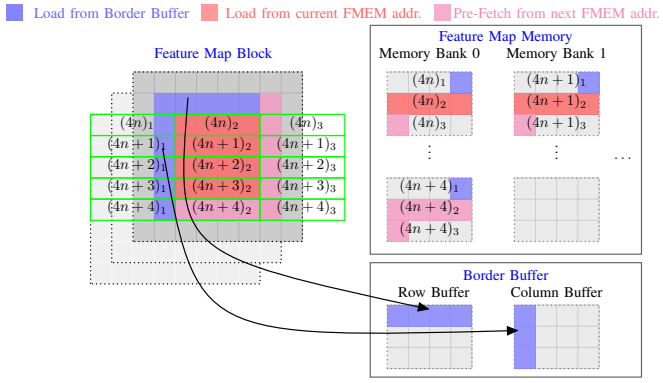


Fig. 7. Feature-map memory and border buffer organization. Row $(4n+i)_j$ of the feature-map block is stored in the j-th row of the i-th FMEM bank. Pixels already processed are read from the border buffer.

also feature-map values belonging to previous or subsequent blocks. For example, in a 3×3 convolution, input feature map values that are in located exactly in the border of the current 4×4 block are also required. This is illustrated in Fig. 7. Top and left border pixels have already been accessed at some previous step; hence we utilize a border buffer consisting of a column and a row buffer to store these values. Since a row-first order is followed during the execution of the nested loops, entries in the column buffer need only be retained for one subsequent block calculation, while entries of the row buffer must be retained for one entire row calculation.

4) Block Buffer: One 42 KB block buffer composed of two banks implemented with register file macros, has also been added to the memory hierarchy. Each of these banks banks has the capacity to store a 3D input feature-map slice of size $4\times4\times C_{\text{in}}$ and serves two purposes: Firstly, when part of the input feature-map needs to be loaded from the off-chip memory, this buffer acts as an intermediate storage between the off-chip memory and the processing cores, bypassing FMEM and allowing to overlap the convolution computations with the loading process, while reducing power consumption since the larger and more power consuming FMEM SRAM can be turned off when loading data from the external memory. If the loading time of a block is less than its execution time, then complete overlapping can be achieved and the loading process does not affect the system's throughput at all. The necessary condition can be expressed as follows:

$$F_x \times F_x \times C_{\text{in}} \times R > 4\times4\times C_{\text{in}} \times b_f(1-c_r) \times \frac{F_{\text{int}}}{F_{\text{io}}/4}, \quad (5)$$

where F_x, F_y are the filter's dimensions, C_{in} is the number of input channels, R is the number of times a block is reused or equivalently the number of Loop-K tiles C_{out}/K, c_r is the compression ratio when transferring feature-maps, $b_f = 1.5$ is the bit-width of each input value and F_{int} and F_{io} are the internal and IO cells frequency, respectively. The left-hand side term corresponds to the execution time of a block, while the right-hand side corresponds to its transfer time. Secondly, when mode_0 is used, which means that an input block is used multiple times for different sets of weight kernels without being updated, it can remain stored in the block buffer. In the first iteration (tile of Loop-K) it will be read from FMEM and copied to the block buffer, so that during the following iterations it can be read directly from there, bypassing FMEM which can be turned off. Since this smaller, latch-based memory consumes less power than the larger SRAM, this scheme decreases power consumption.

E. Compression/Decompression mechanism

In any DNN processing system, it is important to minimize the overhead of the data transfers associated with the off-chip memory, by increasing reuse of network parameters and intermediate results (feature-maps) and by more efficiently taking advantage of the available off-chip IO bandwidth. In the proposed architecture, this is primarily facilitated with the introduction of the block buffer and of the two modes of operation, as described in Section IV-D, but also with the addition of a compression mechanism which decreases the volume of the transferred data.

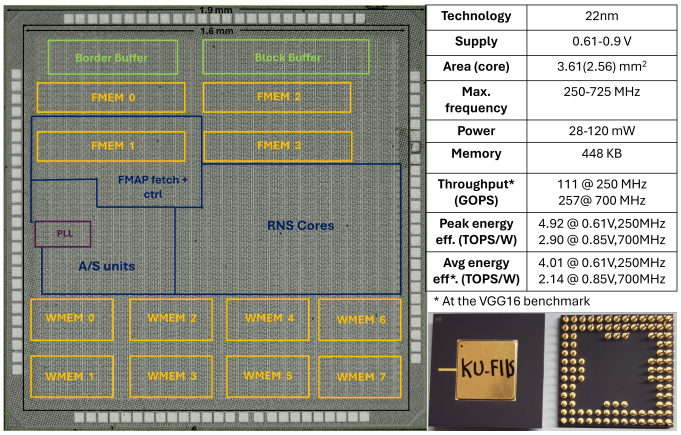
The compression unit aims to exploit sparsity of feature-map vectors, particularly due to the ReLU activation function, which maps negative values to zero (no compression is applied to the weights). The compression mechanism employs a variation of run-length encoding where data is grouped into 4×4 feature-map blocks. For each block, only the non-zero values are transferred, along with an index corresponding to the distance to the previous non-zero value. Encoded data are stored in a FIFO queue to be transferred to the off-chip memory according to the available bandwidth. A decoder applies the decompression process to the received feature-map data and provides the decompressed values (4×4 original block) to be stored inside the on-chip memory. The compression ratio depends on the sparsity of data and can be calculated as $c_r = \frac{16b_v - 16(b_v+b_i)sp}{16b_v}$, where sp denotes sparsity (ratio of non-zero values to the total number of feature-map values), b_i, b_v denote the number of bits used to represent the index and value corresponding to a non-zero feature-map entry, respectively. Since $b_v = 12$ bits are used for representing feature-map values and $b_i = 4$ bits for representing the index, and assuming 0.5 sparsity (typical after ReLU), the compression ratio is 0.33, considerably decreasing off-chip data transfer cost.

V. Results and Discussion

This section presents a comprehensive evaluation of the RNS-based DNN accelerator, both from a hardware as well as a model accuracy perspective. Comparisons with the conventional binary system implementation as well as with sate-of-the-art DNN processing systems are reported.

A. Prototype chips and testing setup

A prototype CMOS chip, which encompasses the architectural and RNS-specific contributions described above, was fabricated using the Global Foundries 22 FDSOI process. Silicon measurements confirm simulation results and substantiate the benefits of using RNS in a physically implemented hardware accelerator. The prototype chips feature 256 PEs organized in 16 cores and 8 A/S units. A description of the testing setup and hardware performance metrics obtained from silicon measurements are reported in the following paragraphs.



Technology	22nm
Supply	0.61-0.9 V
Area (core)	3.61(2.56) mm^2
Max. frequency	250-725 MHz
Power	28-120 mW
Memory	448 KB
Throughput* (GOPS)	111 @ 250 MHz 257@ 700 MHz
Peak energy eff. (TOPS/W)	4.92 @ 0.61V,250MHz 2.90 @ 0.85V,700MHz
Avg energy eff*. (TOPS/W)	4.01 @ 0.61V,250MHz 2.14 @ 0.85V,700MHz

* At the VGG16 benchmark

Fig. 8. Die photo, chip specifications and packaged chips

1) Chip specifications:

a) Area: The core area of the chip is 2.56 mm^2 (1.6 mm$\times 1.6$ mm), while the IO ring together with the wirebond pads and crackstop add another 0.15 mm (to all sides of the chip), resulting in a 3.61 mm^2 total area (1.9 mm$\times 1.9$ mm). A photo of the die is shown in Fig. 8.

b) Timing: The chip can achieve a maximum frequency of 725 MHz with a 0.9 V power supply for the standard cells. Timing is also verified at lower voltage supplies. For low-power operation, the chip can operate at 250 MHz with a 0.61 V supply voltage (0.7 V for the SRAM macros). Detailed power measurements for different voltage-frequency points are reported in Section V-B.

c) Memory: The prototype system has a total of 448 KB on-chip SRAM and 78 KB of register file memory. Each FMEM bank consists of four 2048×48 SRAM arrays, while each WMEM bank consists of two 4096×64 macros. The border buffer consists of six 1024×48 register file macros, while the block buffer consists of $14\ 512 \times 48$ register file macros. At any time, at most four of the FMEM SRAM macros and two of the weight macros (16 weights for each core = 128 bits) are activated.

d) Package and IO: The chip was fabricated by Global Foundries on a 22-nm FDSOI process, with a 10-layer metal stack. A wide-range PLL from Analog Bits was utilized for generating the internal clock. A 14 mm$\times 14$ mm ceramic pin grid array (CPGA) package with 100 pins was used. The chip features a 32-bit input bus, a 32-bit output bus and 9 control pads (clocks, resets, and handshake signals), and 27 power and ground pads. Four power supplies are used: (1) core power supply (V_{ddc}), SRAM memory array power supply (V_{ddm}), (3) IO power supply (V_{ddio}) and (4) PLL power supply (V_{pll}). Nominal value for V_{ddio} and V_{pll} is 1.8 V. The packaged chips are shown in Fig. 8. An FPGA board is used to interface the prototype chips. An FMC VITA 57 connector connects the chip, through a custom PCB board, to an FPGA, which is used for initialization, data storage and debugging. This setup allows for maximum flexibility and can support the execution of complex workloads by enhancing the chip capabilities with the available FPGA resources.

B. Silicon power measurements

The major goal of this work is to translate RNS efficiency in the implementation of the MAC operation into end-to-end system performance gains and illustrate these performance gains on a fabricated chip. Since the application of the developed system will be on power-constrained devices, the focus has been primarily on reducing power consumption. More specifically, the following techniques have been employed :

1) Amortizing the usage of complex RNS units (activations, scaling) over a large number of MACs and exploiting the periodic nature of its use through clock gating.
2) Reducing operation voltage in order to trade off the increased speed of the RNS arithmetic circuits for power reduction. By taking advantage of the quadratic scaling of power consumption with the voltage, versus the linear impact that it has on speed, larger energy savings have been achieved.
3) Reducing memory access cost by introducing multiple levels of memory hierarchy and optimizing memory access patterns.

The effectiveness of the above techniques is verified on the actual chips by performing comprehensive power measurements for (a) various workloads (layer geometries) to illustrate the effect of amortizing the non-trivial RNS activation and scaling (A/S) units and optimizing memory access patterns and hierarchy, and (b) various voltage-frequency operation points in order to illustrate the low-power capabilities. The following results refer to the average power consumption during the execution of a layer (they hence correspond to peak power efficiency) and are obtained using a current probe connected to the relevant power supply and a high-speed oscilloscope. Digital trigger signals coming from the chip are used to initiate capturing of the current readings.

1) Workload Characterization: Power consumption depends on the percentage of the total execution time during which the activation-scaling (A/S) units are utilized, which depends on the number of input channels, as well as on the *mode* of operation which refers to the order of execution of outer convolution loops. Deeper layers result in greater power efficiency due the infrequent usage of the A/S unit. For example, a layer with 128 input channels results in a $\approx 3\times$ reduction of the A/S units power consumption, compared to a shallow layer with only three input channels. Also, utilizing *mode*_0 (possible only if weights can fit inside the on-chip memory) means that an input block is used multiple times for different sets of weight kernels without being updated, and thus can remain stored in the block buffer. After the initial iteration, it can be read directly from there, bypassing FMEM, considerably decreasing power consumption.

Experiments were conducted with three different layers, representing worst, (shallow layer with 3 input channels and *mode*_1), average (32 input channels and *mode*_1) and best case (deeper layer with 128 input channels, 128 output channels and *mode*_0) scenarios. These layer sizes are selected according to the VGG network structure, where an initial shallow layer is used at the input, corresponding to the worst-case scenario, while the best-case scenario corresponds to the

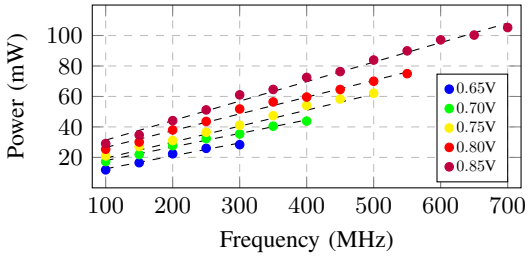


Fig. 9. Core power consumption (V_{ddc}) *vs.* frequency for different core supply voltages.

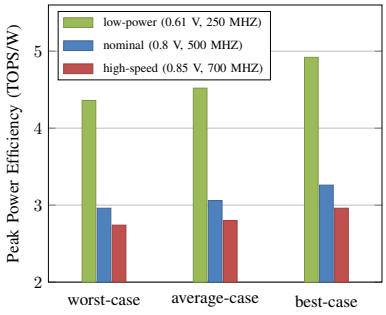


Fig. 10. Peak power efficiency for different workloads and operation points.

largest layer whose weight kernels can fit entirely in the on-chip memory. In most layers, weight parameters do not fit on-chip, while a conservative option for 32 input channels (corresponding to C_{split}) suffices to avoid overflow, hence the choice for the average-case scenario. We run these experiments at three different operation-point scenarios: (1) low-power at 0.61 V and 250 MHz, (2) nominal at 0.8 V and 500 MHz, and (3) high-speed at 0.85 V and 700 MHz. Power consumption of the core and memory as well as peak power efficiency (when all parameters are loaded in the on-chip memory and we have a maximum utilization of the 256 PEs) are reported in Table I and visualized in Fig. 10. At the nominal operation point and during the average-case workload the total power consumption of the chip is 83.3 mW, while during worst and best case workloads the power consumption is 86.2 mW and 78.2 mW, respectively (12% difference). The maximum peak power efficiency for the average case, which is achieved at the low-power operation point (250 MHz at 0.61V) is 4.52 TOPS/W. The overall maximum peak power efficiency 4.92 TOPS/W, at the best case workload and low-power operation point. The average power consumption during the execution of a NN models depends on the size of its layers. It also depends on the available external memory bandwidth. Effective power consumption, taking into account the data transfer time and power consumption during this idle PE time, is reported in Section V-D for a benchmark CNN model.

2) Frequency-Voltage characterization: For a comprehensive characterization of the chip performance, we conduct measurements for various supply voltages (from 0.61 V to 0.9 V) and frequencies (from 100 to 725 MHz). The maximum attainable frequency is 725 MHz, at 0.9 V. Results are presented in Fig 9. For each voltage level, we plot power consumption for the average-case scenario up until the maximum achievable frequency, with 50 MHz increment steps.

C. Area/power breakdown and comparison with BNS

To estimate what percentage of the total power is consumed by each component and compare the RNS architecture to an equivalent BNS we use post-layout power estimation reports after switching activity annotation, at the nominal operation point (@ 0.8 V, 500 MHz). The total power consumption, using the Synopsys PrimeTime tool is 79 mW, versus 83.3 mW measured on silicon. The absolute area and power consumption of the various RNS functional components as well as their binary counterparts' are reported in Table II, while the relative

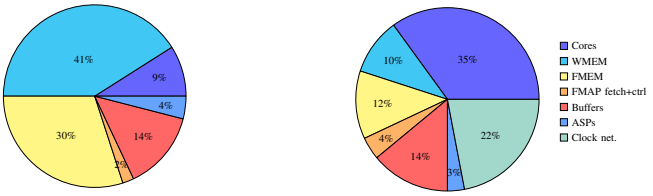


Fig. 11. Area (a) and power consumption (b) breakdown of the RNS accelerator's functional components.

contribution of each of them is visualized in Fig. 11. The RNS processing core consumes 1.49× less power compared to the binary counterpart, however there is a an overhead due to the more complex A/S and pooling units of about $8 \times (0.25 - 0.06) = 1.52$ mW, assuming an average-case workload, and an $\approx (3.12 - 1.83) = 1.3$ mW, overhead from the base extension units of the FMAP fetch/routing unit (the BNS system has a trivial activation and unit and the FMAP fetch unit only includes the shift registers and no base extension units). RNS and BNS are equivalent in terms of memory requirements and data transfer cost from memory to processing cores, since the same number of bits is stored and fetched for feature-maps and weights for both implementations. According to the contribution of each component (Fig. 11) this results in approximately 1.33× power reduction, considering only the processing logic or 1.14× end-to-end power reduction considering the memory system and clock network as well, compared to a BNS implementation.

D. Benchmarking

1) Hardware performance: Effective (average) power consumption depends on the volume of data that need to be transferred to and from the off-chip memory (FPGA board), which in turn depends on the network structure (size of layers), and also on the available external memory bandwidth, which is bottlenecked by the GPIO frequency. Using the VGG16 network as a benchmark application, the system delivers an average throughput of 111 GOPS at the low operation point, with 0.002 external memory accesses/MAC. With an average power consumption of 27.7 mW, this translates to an power efficiency of 4.01 TOPS/W, using a 0.61 V supply at 250 MHz. At the high-speed operation point, the system

TABLE I
WORKLOAD CHARACTERIZATION

Operating Point	V_{ddc} (V)	V_{ddm} (V)	Frequency (MHz)	**Worst-case** ($C_i = 3$, mode = 1) P_{core} (mW)	P_{mem} (mW)	P_{tot} (mW)	Peak (TOPS/W)	**Average-case** ($C_i = 32$, mode=1) P_{core} (mW)	P_{mem} (mW)	P_{tot} (mW)	Peak (TOPS/W)	**Best-case** ($C_i = C_o = 128$, mode=0) P_{core} (mW)	P_{mem} (mW)	P_{tot} (mW)	Peak (TOPS/W)
Low-power	0.61	0.7	250	24.3	4.9	29.2	4.36	23.4	4.8	28.2	4.52	22.2	3.8	26.0	**4.92**
Nominal	0.8	0.8	500	72.8	13.4	86.2	2.96	70	13.3	83.3	3.06	68.0	10.2	78.2	3.26
High-speed	0.85	0.9	700	109.6	20.2	129.9	2.74	105.2	22.3	127.5	2.80	103.4	17.3	120.8	2.96

TABLE II
POWER AND AREA BREAKDOWN (@0.80 V, 500 MHZ)

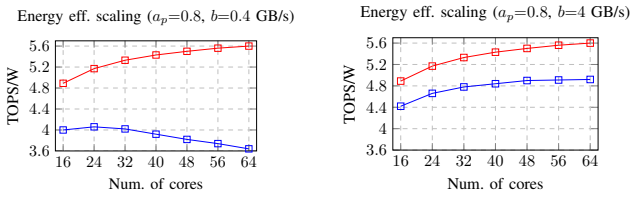
Units	**Components**	**Power (mW)** RNS	BNS	**Area (mm^2)** RNS	BNS
Core	16×PE + 1 BE	1.71	2.55	5137	5446
FMAP Fetch	16 BE Units + Shift Regs.	3.12	1.83	5668	2470
A/S	Scaling + 2 Sign Units BE + pool + regs.	0.25	0.06	4584	1420
FMEM	8 SRAM macros	9.6		292×10^3	
WMEM	16 SRAM macros	8.2		397×10^3	
Border Buf.	6 Reg. File macros	6.6		54×10^3	
Block Buf.	14 Reg. File macros	5.5		88×10^3	

TABLE III
MODEL ACCURACY (%)

	Original (FP32)	Quantized (INT8)	RNS $\mathcal{B}_c$	RNS $\mathcal{B}_e$, split conv.
VGG19	70.75	70.22	68.40	70.42
ResNet50	75.20	74.36	74.85	74.85

delivers 257 GOPS with 120 mW power consumption, which translates in an power efficiency of 2.14 TOPS/W.

2) Network accuracy: The architecture is also evaluated in terms of accuracy on the application level, using popular CNN benchmarks. More specifically, the accuracy of the RNS accelerator on ImageNet is investigated, using the ResNet50 [21] and VGG19 [23] models. An 8-bit quantized version of these networks is used, obtained from Xilinx Vitis-AI model zoo [30]. We also evaluate the impact of the limited dynamic range ($\mathcal{B}_c$ provides $\approx$ 20.1 bits equivalent range) and the potential benefit of using the 6-channel periodic base extension method presented in IV-C3. Weights are represented using the (approximately) 8-bit equivalent RNS base $\mathcal{B}_w$, while the 12-bit equivalent base $\mathcal{B}_f$ is used for feature-maps. Results are reported in Table III. For VGG19, there is a 2.35% accuracy loss, when no base extension is performed, which drops to 0.33% when the periodic base extension which splits the convolution computations along the input channel dimension is used. For ResNet50, the problem of overflow during partial product accumulation does not affect overall accuracy (only 0.40% accuracy drop), thus no base extension to six channels is needed. However, for both networks the periodic base extension is used in the final fully connected layers. In both cases, the RNS system performs better than the 8-bit quantized model (since 12 bits are used for feature-maps instead of 8).



(a) (b)

Fig. 12. Peak (red) and average (blue) power efficiency scaling with the number of cores for the VGG16 benchmark.

E. Scalability projections

The most efficient way of scaling up the proposed architecture is by increasing the number of processing cores. This would linearly increase throughput, when data has been fetched to the processing elements, at the cost of the increased power consumption, which however is not expected to increase linearly. This is because certain components of the architecture, such as FMEM and the FMAP-fetch unit are shared between cores. We thus make the following assumption: a_p is the factor of the total processing power (excluding memory) that increases linearly with the number of cores, a_m is the memory accessing power factor that increases linearly, and b is the available external memory bandwidth. Based on the power consumption breakdown (Fig. 11), we set $a_p = 0.8$ and $a_m = 0.5$. Plots of Fig. 12 show the estimated peak (red) and effective (blue) power efficiency as a function of the number of cores, for the VGG16 benchmark. While the peak power efficiency increases as the number of cores increases, reaching a value of 5.6 TOPS/W at 64 cores, the effective power efficiency receives a maximum value of 4.1 TOPS/W at a smaller number of cores (24), which is quite close to the actual number of cores utilized in the prototype chips. This is because, given the limited available bandwidth, the loading time (while the chip waits for data and performs no operation) to run time ratio increases, thus the average utilization of the PEs drops. However, if a higher bandwidth was available, the effective efficiency would more closely follow the peak efficiency (higher PE utilization), reaching 4.92 TOPS/W, thus we could more significantly benefit from an increased number of cores. We note that the optimal number of cores for the available memory budget depends on the benchmark been executed. For example, in a smaller network than VGG16, where most parameters fit in the on-chip memory, the average power efficiency would be closer to the peak performance.

TABLE IV
COMPARISONS TO STATE-OF-THE-ART IMPLEMENTATIONS

	ISSCC'20 [31]	ISSCC'22 [32]	ISSCC'21 [33]	ISSCC'23-a [34]	ISSCC'23-b [35]	RNSDNN [11]	This work RNS
Process	7 nm	65 nm	28 nm	28 nm	28 nm	45 nm	**22 nm**
Supply voltage (V)	0.575–0.825	1	0.6–0.9	0.66–1.33	0.65–0.9	1	**0.61–0.85**
Max. Frequency (MHz)	290–880	400	100–470	100–500	55–285	1200	**250 – 725**
On-chip Memory (KB)	2176	150	206	1120	537	NG	**448**
Bit Precision (act,wgt)	8	8	8	8	8	16,8	**12,8**
Network	MobileNet-v1	VGG16	VGG	ResNet50	Random workload	VGG16	**VGG16**
Performance (GOPS†)	3604	NG	1590	NG	84–437	134	**111–257**
Area (mm^2)	3.04	4.47	1.9	7.81	2.18	NG	**2.56**
Power (mW)	174–1053	126	19–140	17-174	NG	183	**28 – 120**
Power Eff. (TOPS†/W)	6.83	0.66-1.8	11.67	10.7	3–8.09	0.446	**4.01(4.92)@(0.61V,250MHz)*** **2.14(2.90)@(0.85V,700MHz)***

†1 MAC = 2 OPS * average (peak) efficiency NG: not given

F. Comparison to state of the art

The major advantages of the proposed architecture compared to state-of-the-art RNS based DNN systems are discerned within the following points: (a) The implemented RNS accelerator utilizes a balanced moduli set, with a maximum channel size of 5 bits. This is in contrast to Res-DNN [11], where the extension of a single channel (up to 12 bits), while reducing the complexity of the base extension circuits, creates an unbalanced representation where the largest channel becomes the bottleneck, thus partially canceling of the benefits of using RNS. (b) While the expensive base extension operation in [11] takes place before and after each multiplication, leading to the need for a larger number of such units compared to the number of PEs, the proposed architecture amortizes the complex RNS operations (activation functions, division) over 32 PEs. This is facilitated by the temporal reuse of network parameters (broadcasting), which reduces the number of base extension units, as well as the periodic usage of the A/S units, which are shared between two cores and are only used once the accumulation of multiple partial sums is completed. This dataflow, paired with the clock-gating scheme, diminishes the overhead of the non-trivial operations and allows taking full advantage of the RNS potential for low-power DNN inference.

Quantitative comparisons to state-of-the-art 8-bit digital AI accelerators is reported in Table IV. The implemented RNS accelerator offers a slightly higher dynamic range of 12 bits for activations than these systems. We report peak and avarage power efficiency (TOPS/W) on the VGG16 benchmark at the best operation point for our experiments (0.61V, 250 MHz) as well as at the high-speed operation point, where the chip delivers the maximum throughput (257 GOPS). With a 4.92 peak and 4.01 average power efficiency, the developed system is $9\times$ more energy-efficient than Res-DNN (this architecture uses a less advanced 45-nm process but results only refer to synthesis reports, since there is no silicon implementation). The proposed RNS accelerator is also more power efficient than the neural processor presented in [32], while it achieves comparable performance in terms of power efficiency with a cutting-edge 7-nm Samsung AI chip [31] for unpruned networks. The DNN processor proposed in [33] achieves a very high power efficiency of 10.7 TOPS/W, but utilizes high-level optimizations, such as effective weight convolution, to reduce the actual number of operations that are performed. Hence, the reported TOPS/W corresponds to effective power efficiency. In a more relevant comparison, where the benefits come from a hardware arithmetic innovation, the system in [35] uses signed magnitude multipliers to reduce switching activity and reports $3-8$ peak TOPS/W, depending on the layer parameter distributions, and after employing a bit-sparsification technique. In contrast, the performance of the proposed RNS-based accelerator does not dependent on the distribution of the network parameters or any pre-processing step. Apart from RNS-DNN [11], which has not been implemented on silicon, and the Samsung accelerator [31], which is fabricated on a far more advanced process, our chip achieves the maximum attainable clock frequency both at high and low supplies, indicating the high-speed capabilities of the Residue Numbering System. These results suggest that RNS, coupled with architectural and NN model-level optimizations that can be applied orthogonally, such as the dynamic dataflow proposed in [34] or the effective convolution proposed in [33], can push the performance limits of digital DNN processing systems.

VI. CONCLUSION

This paper introduced a RNS-based DNN accelerator. The proposed architecture achieves end-to-end RNS domain processing through innovative usage of activation function, scaling and overflow control techniques, maintaining a small maximum word-length among the residue channels. The non-trivial RNS operation overhead is minimized by amortizing the usage of the A/S units and exploiting their periodic usage to decrease power consumption. Post-layout simulation of the design reveal a $1.33\times$ processing power reduction compared to the binary counterpart, while silicon measurements on the prototype chips confirm simulation results and show considerably more power efficient processing on vision benchmarks compared to state-of-the-art RNS-based DNN accelerator. These findings prove that RNS can markedly enhance the low-power capabilities of modern AI accelerators.

ACKNOWLEDGMENTS

This research was conducted at Khalifa University's SoC Center and supported by the Semiconductor Research Corporation (SRC) project 2020-AH-2983.

REFERENCES

[1] S. Calo, M. Touna, D. Verma, and A. Cullen, "Edge computing architecture for applying AI to IoT," 12 2017, pp. 3012–3016.

[2] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, Dec. 2017.

[3] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. in 2016 IEEE international solid-state circuits conference (ISSCC)," *San Francisco, CA, USA*, pp. 262–263, 2016.

[4] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 267–278.

[5] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 246–247.

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

[7] E. B. Olsen, "RNS Hardware Matrix Multiplier for High Precision Neural Network Acceleration: "RNS TPU"," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[8] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–6.

[9] M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev, and N. Chervyakov, "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation," *Mathematics and Computers in Simulation*, vol. 177, pp. 232–243, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378475420301580

[10] M. Abdelhamid and S. Koppula, "Applying the residue number system to network inference," *arXiv preprint arXiv:1712.04614*, 2017.

[11] N. Samimi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Res-DNN: A Residue Number System-Based DNN Accelerator Unit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 2, pp. 658–671, 2020.

[12] Z. Torabi and G. Jaberipur, "Low-Power/Cost RNS Comparison via Partitioning the Dynamic Range," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 5, pp. 1849–1857, 2016.

[13] V. Sakellariou, V. Paliouras, I. Kouretas, H. Saleh, and T. Stouraitis, "On reducing the number of multiplications in RNS-based CNN accelerators," in *2021 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2021, pp. 1–6.

[14] ——, "A multiplier-Free RNS-Based CNN accelerator exploiting bit-Level sparsity," *IEEE Transactions on Emerging Topics in Computing*, no. 01, pp. 1–16, 2023.

[15] A. Roohi, M. Taheri, S. Angizi, and D. Fan, "Rnsim: Efficient deep neural network accelerator using residue number systems," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[16] S. Salamat, M. Imani, S. Gupta, and T. Rosing, "RNSnet: In-Memory Neural Network Acceleration Using Residue Number System," 11 2018, pp. 1–12.

[17] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[18] C. Efstathiou, H. Vergos, G. Dimitrakopoulos, and D. Nikolos, "Efficient diminished1 modulo $2^n + 1$ multipliers," *IEEE Transactions on Computers - TC*, vol. 54, pp. 491–496, 04 2005.

[19] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Improving post training neural quantization: Layer-wise calibration and integer programming," *ArXiv*, vol. abs/2006.10518, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219792681

[20] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[24] N. S. Szabó and R. I. Tanaka, *Residue arithmetic and its applications to computer technology*. McGraw-Hill, 1967.

[25] K. Gbolagade and S. Cotofana, "An $O(n)$ Residue Number System to Mixed Radix Conversion Technique." in *Proceedings - IEEE International Symposium on Circuits and Systems*, 05 2009, pp. 521–524.

[26] H. Xiao, Y. Ye, G. Xiao, and Q. Kang, "Algorithms for comparison in residue number systems," in *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*, 2016, pp. 1–6.

[27] M. Xu, Z. Bian, and R. Yao, "Fast Sign Detection Algorithm for the RNS Moduli Set $\{2^{n+1}-1, 2^n-1, 2^n\}$," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 2, pp. 379–383, 2015.

[28] V. Sakellariou, V. Paliouras, I. Kouretas, H. Saleh, and T. Stouraitis, "A high-performance RNS LSTM block," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 1264–1268.

[29] Y. Kong and B. Phillips, "Fast Scaling in the Residue Number System," *IEEE Transactions on VLSI Systems*, vol. 17, pp. 443–447, 03 2009.

[30] Xilinx. Vitis-AI. [Online]. Available: https://github.com/Xilinx/Vitis-AI

[31] C.-H. Lin, C.-C. Cheng, Y.-M. Tsai, S.-J. Hung, Y.-T. Kuo, P. H. Wang, P.-K. Tsung, J.-Y. Hsu, W.-C. Lai, C.-H. Liu, S.-Y. Wang, C.-H. Kuo, C.-Y. Chang, M.-H. Lee, T.-Y. Lin, and C.-C. Chen, "A 3.4-to-13.3TOPS/W 3.6TOPS Dual-Core Deep-Learning Accelerator for Versatile AI Applications in 7nm 5G Smartphone SoC," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 134–136.

[32] Y. Ju and J. Gu, "A 65nm Systolic Neural CPU Processor for Combined Deep Learning and General-Purpose Computing with 95% PE Utilization, High Data Locality and Enhanced End-to-End Performance," in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

[33] H. Mo, W. Zhu, W. Hu, G. Wang, Q. Li, A. Li, S. Yin, S. Wei, and L. Liu, "A 28nm 12.1TOPS/W Dual-Mode CNN Processor Using Effective-Weight-Based Convolution and Error-Compensation-Based Prediction," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 146–148.

[34] C.-Y. Du, C.-F. Tsai, W.-C. Chen, L.-Y. Lin, N.-S. Chang, C.-P. Lin, C.-S. Chen, and C.-H. Yang, "A 28nm 11.2TOPS/W Hardware-Utilization-Aware Neural-Network Accelerator with Dynamic Dataflow," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 1–3.

[35] H. An, Y. Chen, Z. Fan, Q. Zhang, P. Abillama, H.-S. Kim, D. Blaauw, and D. Sylvester, "29.3 An 8.09 TOPS/W Neural Engine Leveraging Bit-Sparsified Sign-Magnitude Multiplications and Dual Adder Trees," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 422–424.