Residue-Net: Multiplication-free Neural Network by In-situ No-loss Migration to Residue Number Systems

Sahand Salamat, Sumiran Shubhi, Behnam Khaleghi, Tajana Rosing
Computer Science and Engineering Department, UC San Diego, La Jolla, CA 92093, USA
{sasalama, sshubhi, bkhaleghi, tajana}@ucsd.edu

ABSTRACT
Deep neural networks are widely deployed on embedded devices to solve a wide range of problems from edge-sensing to autonomous driving. The accuracy of these networks is usually proportional to their complexity. Quantization of model parameters (i.e., weights) and/or activations to alleviate the complexity of these networks while preserving accuracy is a popular powerful technique. Nonetheless, previous studies have shown that quantization level is limited as the accuracy of the network decreases afterward. We propose Residue-Net, a multiplication-free accelerator for neural networks that uses Residue Number System (RNS) to achieve substantial energy reduction. RNS breaks down the operations to several smaller operations that are simpler to implement. Moreover, Residue-Net replaces the copious of costly multiplications with non-complex, energy-efficient shift and add operations to further simplify the computational complexity of neural networks. To evaluate the efficiency of our proposed accelerator, we compared the performance of Residue-Net with a baseline FPGA implementation of four widely-used networks, viz., LeNet, AlexNet, VGG16, and ResNet-50. When delivering the same performance as the baseline, Residue-Net reduces the area and power (hence energy) respectively by 36% and 23%, on average with no accuracy loss. Leveraging the saved area to accelerate the quantized RNS network through parallelism, Residue-Net improves its throughput by 2.8x and energy by 2.7x.

CCS CONCEPTS
• Hardware → Hardware accelerators; • Computing methodologies → Neural networks.

1 INTRODUCTION
Many of the machine learning applications including robotics, autonomous driving, and virtual/augmented reality require real-time processing where the computation platforms are strictly constrained by the size of the computation and communication [1–4]. Deep Neural Networks (DNNs) are widely exploited thanks to their ever-increasing accuracy. The accuracy of DNNs is highly dependent on the complexity of the model. Certain applications can afford to sacrifice the classification accuracy to achieve higher performance or reduce energy consumption whereas plenty of applications, e.g., autonomous driving workloads, require the highest accuracy to avoid fatal consequences of misprediction, while energy usage and performance (to process hundreds of inputs per second) is of utmost importance.

Quantizing the floating-point weights and activations (i.e., outputs of network layers) to fixed-point numbers with lower precision is a widely applied technique to shrink the memory footprint and computational complexity in DNN inference [1]. Previous works have shown that DNNs can be quantized down to six bits without affecting the accuracy of the network [5]. Quantizing the network not only reduces the size of the required memory to store the weights but also simplifies the computation of the multiplication and accumulation (MAC) operations, which contribute to more than 99% of operations in DNN inference [6]. FPGAs have been widely used to accelerate DNN operations [7, 8]. Figure 1 shows the efficacy of quantization in reducing the complexity of DNN inference (particularly MAC operations) on FPGA. Quantizing 32-bit fixed-point weights to six bits reduces the number of Look-up Tables (LUTs) by 97%, without adverse impact on the accuracy of the network after re-training (because of scale issue, LUT count for 32-bit operands is not shown, though the ∝ n² trend can be discerned from the figure). Although reducing the bit-width below six bits saves area, its significantly lower accuracy is not affordable for the majority of real-world applications.

In this paper, instead of trying to squeeze networks below six bits, we leverage a modified number representation to improve resource usage by breaking down the computation to smaller operations. Residue Number System (RNS) is an unorthodox number representation developed based on the Chinese Remainder Theorem (CRT). RNS has used in compute-intensive applications such as DNNs and digital signal processing applications [9–11] RNS is defined by a set of k integer numbers that are pairwise co-prime, called moduli set. To represent a binary number in RNS format, the number is divided by each modulus where the residues (remainders of divisions) represent the number in RNS. Each residue (remainder) is always smaller than the corresponding modulus. Therefore, for representing each residue fewer bits are needed. RNS simplifies the operations by breaking them down to the same operations on the residues with smaller bit-width. Consequently, it needs simpler arithmetic for implementation. Figure 1 shows the area of a MAC

Figure 1: Resource utilization of MAC operation in conventional binary, RNS, and Residue-Net on FPGA.
unit in RNS format for different bit-widths. The area is the total area of the residue MAC units. For instance, for six-bit binary numbers that are converted to RNS with \{3, 4, 5\} moduli set, the area of the MAC unit is shown as six-bit RNS MAC. Six-bit RNS MAC reduces the area by 2.8× as compared to the binary MAC unit (18 LUTs versus 44).

In this paper, we propose Residue-Net that uses six-bit RNS with \{3, 4, 5\} moduli set. Residue-Net exploits the benefits provided by RNS to represent six-bit network weights with two 2-bit and one 3-bit numbers. Considering the \{3, 4, 5\} moduli set, in Residue-Net, the residues can only be within the set of \{0, 1, 2, 3, 4\}. Therefore, multiplications over the mentioned set can be implemented with add and shift operations (as elaborated in section 3.1) which are more energy-efficient than the primary fixed-point multiplications. As it is shown in Figure 1, the six-bit MAC unit of Residue-Net, by eliminating the multiplication operations, requires 50.0% fewer LUTs than the baseline RNS MAC unit, and 79.5% less than the original binary unit.

Residue-Net trains and quantizes the DNN model using the approach proposed in [12], and transforms the weights and activations into RNS to carry out the exact computations on operands with fewer bits. Since Residue-Net performs all the computations in exact mode, the accuracy of the network remains the same. Residue-Net, not only breaks down the computations to operations with fewer bits, but it is also able to replace the costly multiplications with simpler add operations to further reduce the computation complexity. The main contribution of this paper is as follows.

(i) We propose Residue-Net, a novel FPGA-based accelerator based on RNS which maps all activations and weights of a neural network to RNS, which breaks down the six-bit operations to two 2-bit and a 3-bit operation.

(ii) We further simplify the multiplications to add and shift operations, which further enhances the computation complexity of the neural network inference.

(iii) The results of implementing four popular neural networks, LeNet, AlexNet, VGG16, and ResNet-50 on FPGA leveraging the aforementioned innovations reveal 2.8× average speedup without impacting the accuracy.

2 BACKGROUND AND RELATED WORK

Neural networks have been extensively used in many applications and have been accelerated on various platforms [7, 13]. Quantization has been explored to compress and accelerate the neural network in literature [3, 5, 14, 15]. The work in [16] developed a tool that enables both software simulation and hardware realization of DNNs using different data representations and approximate computing blocks. The work in [5] proposes an automated DNN inference accelerator that first quantizes the network and then re-trains it to retrieve the accuracy drop. The work [3] uses reinforcement learning for hardware-aware layer-wise weight quantization. The above works use static quantization so that the weight bit-width is set fixed either for the entire network or on a layer-by-layer basis. However, the studies in [14, 17] quantize the weights based on the complexity of the input such that difficult-to-predict inputs run through a more precise network. For over-simplifying the execution of the neural network, studies [18] quantize the network weights down to binary, \{-1,+1\}, which replaces the multiplications with XOR operations. Although binarized neural networks are significantly faster than fixed-point alternatives, they are practically unappealing for real-world applications due to their low accuracy [19, 20].

On an orthogonal research path, unorthodox number representations have been studied to achieve better accuracy or performance than the conventional fixed-point or floating-point networks [11, 21–23]. The study in [21] proposes narrow-precision floating-point representations instead of 32-bit floating-point numbers to simplify the operations. The posit number system is introduced in [22] as an alternative representation of IEEE floating-point number to represent real numbers. Posit has a larger dynamic range and higher accuracy as compared to floating-point numbers which make it suitable for DNNs to get the same accuracy as 32-bit floating-point weights with less than eight-bit posit weights [24]. While these studies convert a trained network to posit representation, [25] uses posit numbers to train the networks. Posit operations, however, are more complicated than the floating-point ones [26].

Residue Number System: RNS has been used in compute-intensive applications such as digital signal processing [9] and neural networks [10, 11, 27] to accelerate them by reducing the bit-width of operands. A number in RNS is represented by the remainders (residues) of dividing it by a set of numbers, called moduli. Moduli set is a set of numbers \(M_i \in \text{Moduli Set\{M}_1, \ldots, \ldots, M_k\}\) where any pair \(M_i\) and \(M_j\) are co-prime. The process of dividing the RNS numbers back to the binary system is called backward conversion. Several studies such as [28] investigate the impact of different moduli sets on the computation complexity of the forward conversion step. Multiplication and add operations, the two most common operation in DNNs, can be directly performed in the RNS domain [11]. Being the remainder of the division by a modulus, each number in RNS is smaller than the modulus, thereby RNS represents a binary number with multiple smaller numbers with fewer bits. Several recent studies have focused on optimizing the architecture of RNS operations [29]. By decreasing the bit width of the operands at the cost of increasing the number of operations, RNS simplifies the hardware implementation that can be translated to a higher parallelism.

The work in [11] uses RNS to implement the DNNs with resistive processing-in-memory (PIM) technology. They utilize RNS to simplify the operation such that all the DNN operations can be run in RRAM crossbar memory. The work in [30] focuses on using RNS to improve multipliers for neural network applications on FPGA. Works in [27] and [31] use nested RNS to reduce the bit width of numbers so that it maps all the NN operations in FPGA LUTs to increase the efficiency of computations. In our work, we take the advantage of RNS to breakdown the weights and activations so that the costly multiplications can be performed with energy-efficient shift and add operations whilst keeping the accuracy intact.

3 PROPOSED RESIDUE-NET

DNNs consist of convolutional (CNV), activation function (AF), fully connected (FC), and pooling layers. Residue-Net first converts the incoming inputs to RNS. The rest of the intermediate computations are carried out in RNS format. The moduli set for Residue-Net is \(\{2^t - 1, 2^t, 2^t + 1\}\); for \(t\) equal to 2, the moduli set becomes \{3, 4, 5\}. Considering the RNS with the selected moduli set, valid values for \(R_3\) (residue of dividing by 3) are \{0, 1, 2\}, and valid values for \(R_2\) and \(R_5\) are \{0, 1, 2, 3\} and \{0, 1, 2, 3, 4\}, respectively. To represent \(R_3\) and \(R_5\), Residue-Net requires two bits, while to represent \(R_2\) it requires three bits. Therefore, Residue-Net converts six-bit numbers and operations into two two-bit and one three-bit numbers and operations. In the following, we first introduce the required
operations involved in neural networks, and then we propose the Residue-Net architecture to accelerate DNNs.

### 3.1 RNS Operations

RNS can uniquely represent binary numbers in the range of $D = k \prod M_i$ (as each residue $R_i$ can take $i$ distinct values). Therefore, any $i + 1$ binary number $e \in [0, D)$ can be represented uniquely as $\{R_M, R_{M_1}, \ldots, R_{M_k}\}$, where $R_M$ is $|x|_{M_i}$, i.e., $x \mod M_i$. A six-bit binary number $x[5 : 0]$ can be expanded as $x = x[5 : 4] \times 2^4 + x[3 : 2] \times 2^2 + x[1 : 0] \times 2^0$.

We therefore can obtain residues of $R_3$ following Equation (1).

\[
R_3 = |x[5 : 4]| \times 16 + |x[3 : 2]| \times 4 + |x[1 : 0]| \times 1
\]

\[
= x[5 : 4] \times 16|x[3 : 2]| \times 4|x[1 : 0]| \times 1
\]

(1)

Analogously, for $R_4$ and $R_5$ we have:

\[
R_4 = |x[4]| = x[1 : 0]
\]

(2a)

\[
\]

(2b)

Note that to calculate $R_3 = x[5 : 4] + x[3 : 2] + x[1 : 0]$, if the result of the sum is greater than the modulus 3, we subtract the modulus from the result of the addition. $R_4$ is equal to the two least significant bits $x[1 : 0]$, and $R_5$ is calculated in a similar fashion of calculating the $R_3$, though the middle addition is replaced by subtraction. Therefore, Residue-Net requires two three-port adders for the forward conversion. To convert the RNS numbers back to the binary system, we use a look-up table that maps every RNS number to its corresponding binary number.

**Addition and Multiplication:** Upon converting binary numbers to RNS, Residue-Net executes the neural network operations in RNS, where the majority of operations in convolution and fully-connected layers are addition and multiplication. Multiplication and addition in RNS are similar to the binary operations. To multiply (add) two RNS numbers, their corresponding residues are multiplied (added); if the result is greater than the modulus, the final residue is computed by subtracting the modulus, as represented by Equation (3) and (4).

\[
(A + B)_{RNS} = \{ |A + B|_3, |A + B|_4, |A + B|_5 \}
\]

\[
= \{ |A|_3 + |B|_3, |A|_4 + |B|_4, |A|_5 + |B|_5 \}
\]

\[
(3)
\]

\[
(A \times B)_{RNS} = \{ |A \times B|_3, |A \times B|_4, |A \times B|_5 \}
\]

\[
= \{ |A|_3 \times |B|_3, |A|_4 \times |B|_4, |A|_5 \times |B|_5 \}
\]

\[
(4)
\]

As alluded before, considering the selected moduli set, RNS numbers will be in the set \{0, 1, 2, 3\}. Thus, Residue-Net simplifies the multiplication to shift and/or add operation. Multiplication by 0 results in constant 0 output. In multiplying by 1, the output simply gets the input value. Multiplication by 2 and 4 are realized by shift operations. Residue-Net computes multiplication by 3 by adding the input with to its left-shifted (i.e., $3x = 2x + x$).

Figure 2 demonstrates how a Residue-Net multiplication module works. First, if the inputs are not in RNS, Residue-Net converts them to RNS using the forward conversion (FWC) formulated earlier. Each multiplier covers all possible scenarios of the weight parameter (which is also in RNS format) and inputs/activation values. As shown in the figure, Residue-Net only utilizes a two-bit and a three-bit adder plus three multiplexers to implement a six-bit multiplication. The output of multiplications in RNS format can also be larger than the moduli set, which needs to be converted back to the right range. For this, at the end of computations, Residue-Net uses a Ranging Block (RB) to calculate the residue in case the output of multiplications are greater than the corresponding modulus.

The example of Figure 2 computes $7 \times 4$ in RNS. Residue-Net first transforms the input 7 to $R_3 = 3, R_4 = 3, R_5 = 2$ (if the input is an activation, i.e., outputs of intermediate layers, it will be already in RNS format). All weight parameters of the model are converted to RNS format offline for once. Here, the $w = 4$ is represented as $R_1 = 0, R_4 = 0, R_5 = 4$. Residues of the weights are connected to the select port of the multiplexers. The output of the multiplexers are, respectively, $\{R_3 = 1, R_4 = 3, R_5 = 2\}$, which matches with representing $7 \times 4 = 28$ in RNS format: $[28]_3 = 1, [28]_4 = 0$, and $[28]_5 = 3$. We note that the ranging blocks in Figure 2 are just for demonstration purposes. As explained in the following subsection, we share a single RB block for the entire adder tree (that sums up the output of several multiplications) by deferring RB operations after the summations.

**Comparison:** Comparison is another operation required to execute DNNs, mainly used in pooling and activation layers. Comparison cannot be directly performed in RNS format by simply comparing the residues [32]. Instead of converting the RNS numbers back to the binary number system to perform the comparison, Residue-Net uses mathematical attributes of RNS for this purpose.

For a number in our RNS representation with residues $\{R_3, R_4, R_5\}$, we define $\alpha$ as the smallest number that has the same residue for $R_3$ and $R_5$. For instance, for an arbitrary number with $R_3 = 0$ and $R_5 = 1$, we see $\alpha = 6$ as $|6|_3 = 0$ and $|6|_5 = 1 = R_5$. Any number smaller than $\alpha = 6$ does not hold this property. Such a number $\alpha$ will have an arbitrary $R_4$ of $\beta$, i.e., $\alpha|_4 = \beta$. That being said, if we multiply $x = \{R_3, R_4, R_5\}$ RNS by $M_3 \times M_5 = 3 \times 5 = 15$, we observe $|15x|_3 = 0$, $|15x|_4 = |-x|_4 = -R_4$, and $|15x|_5 = 0$ as well. With these insights, we can represent $x = \{R_3, R_4, R_5\}$ in RNS as follows.

\[
x \equiv |\beta - R_4|_{15} + \alpha
\]

(5a)

\[
\rightarrow |x|_3 = ||\beta - R_4||_{15} + |\alpha|_3 = 0 + R_3 = R_3
\]

(5b)

\[
\rightarrow |x|_4 = |\beta - R_4|_{15} + |\alpha|_4 = (R_4 - \beta) = R_4
\]

(5c)

\[
\rightarrow |x|_5 = |\beta - R_4|_{15} + |\alpha|_5 = 0 + R_5 = R_5
\]

(5d)

We use this property to compare two binary numbers $x' = |\beta_x - R_4x|_{15} + \alpha_x$ and $y' = |\beta_y - R_4y|_{15} + \alpha_y$ that have the same...
RNS representation of \( x \) and \( y \): we can simply compare \(|\beta_x - R_{4,x}|\) and \(|\beta_y - R_{4,y}|\) to compare the actual values \( x \) and \( y \). If these two terms are equal, we then use the comparison of \( \alpha_x \) and \( \alpha_y \). To avoid computational complexity, for every possible value of \( x \), Residue-Net stores \( \alpha_x \) and \( \beta_x \) in a memory. Note that binary \( x \) has six bits, so the memory footprint is trivial. The \( R_{4,x} \) (i.e., \(|x|_4\)) is ready beforehand as the operands passed through the layers are all in RNS format. We showed that Residue-Net can execute the neural network operations entirely in RNS representation. After performing all the computations, Residue-Net converts the final results back to the binary system using the backward conversion module explained above for the final soft-max operations. Since the binary number has a limited bit-width of six, Residue-Net transforms each RNS number to the corresponding binary number using a small lookup table.

### 3.2 Residue-Net Architecture

The architecture and dataflow of Residue-Net is illustrated in Figure 3(a). In Residue-Net, the network is trained and weights are quantized to 6-bit binary values offline. The weights are converted to RNS and transferred to FPGA DRAM. Since the computation in RNS is performed on each residue independently, Residue-Net stores the weights residues in three separate memory blocks to simplify the memory access pattern of different residues. During the execution of the network, Residue-Net transfers the weights to FPGA local BRAMs which have considerably faster access latency than the off-chip memory (DRAM). Residue-Net uses a weight stationary dataflow to load the weights. Weights of each kernel remain stationary in the FPGA BRAM to maximize the kernel reuse. Once a convolutional kernel weight is fetched into BRAMs, all the computations that use those weights are executed.

Residue-Net transfers the primary inputs to the Input Buffer which is connected to forward conversion (FCW) modules. FW modules calculate the residues as explained in Equation (1) and (2) for multiple input features in parallel. \( R_1 \), \( R_2 \), and \( R_3 \) of the converted inputs are stored in three separate memory blocks, one for each residue. Residue-Net comprises the required modules for DNN inference, convolutional, fully connected, ReLu activation function, pooling layers, memory blocks, scheduler, and controller. Multiplication and add operations can be carried out independently for each of the residues, however, an RNS comparison demands all the three representing residues. Thus, Residue-Net executes the computations of CNV and FC layers for each residue in parallel while ensuring that results of all residues become ready simultaneously to pass to the next layer. Residue-Net makes three instances of both convolution and fully connected layer, whereby each instance performs computations on the corresponding residue. As executing activation function and pooling layers require all the three residues, only one instance of these layers is present in the architecture. Note that, in Figure 3 (b)/(d), only the convolutional kernel and fully connected neuron of the residue \( R_4 \) is illustrated. Computations on the other two residues are performed in a module with the same architecture but slightly different in the primary arithmetic modules (see Figure 2).

The scheduler of Residue-Net moves primary input data to the input buffer as well as fetches the required weights into the weight memory blocks. Thereafter, based on the network architecture, it initiates different layers sequentially. Convolutional layers might consist of different numbers of convolutional kernels with different shapes. Thus, the scheduler applies the layer configurations to the CNV layers in the runtime. Since each layer may not be able to accomplish the computations in a single cycle, the controller generates the memory access addresses for the layer’s inputs and weights. For the first layer, the controller reads the RNS inputs from the first array of residue memories and writes the intermediate results into the second array of the residue memories. The controller also issues hand-shaking signals when the computations of a layer are accomplished. To execute the next layer, instead of reading the inputs from the first array, the controller reads the layer’s inputs from the second memory array and writes the results into the first array. That is, the controller switches the memories alternatively for every other layer; a layer reads from one memory and writes to the other memory.

Convolutional layers consist of single or multiple convolutional kernels that multiply the inputs by the kernel weights and accumulate the results. The convolutional kernels move along the inputs to generate outputs. Figure 3(b) depicts the convolutional kernel of Residue-Net that operates on the \( R_4 \) residues. The convolutional kernel weights and the inputs are carried out in Residue-Net multiplier (see Figure 2). Since the values of \( R_4 \) are smaller than 4, Residue-Net simplifies the multiplication to selecting among the pre-calculated multiplied values. The number of multiplications in a \( K \times K \) convolutional kernel is equal to the size of the kernel, i.e., \( K \times K \). Residue-Net aggregates the multiplication outputs in a pipelined adder-tree. The result of the accumulation of the multiplied inputs and weights may be greater than the modulus. Therefore, as mentioned earlier, Residue-Net uses a ranging block to convert the output back to valid RNS representation. In the RNS multiplier shown in Figure 2, a ranging block is used immediately after the multiplication. However, in intermediate computations, temporary variables do not need to be in a valid RNS representation. Therefore,
Table 1: Accuracy of the Residue-Net as compared to the full precision and quantized to 6 bits networks.

<table>
<thead>
<tr>
<th></th>
<th>LeNet (MNIST)</th>
<th>AlexNet (ImageNet)</th>
<th>VGG16 (CIFAR-10)</th>
<th>ResNet50 (CIFAR-10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit Fixed-point</td>
<td>99.3%</td>
<td>97.95%</td>
<td>92.44%</td>
<td>93.62%</td>
</tr>
<tr>
<td>6-bit Fixed-point</td>
<td>99.3%</td>
<td>97.95%</td>
<td>92.41%</td>
<td>93.58%</td>
</tr>
<tr>
<td>Residue-Net</td>
<td>99.3%</td>
<td>97.94%</td>
<td>92.41%</td>
<td>93.59%</td>
</tr>
</tbody>
</table>

Residue-Net utilizes only a single ranging block (here, mod 4) after computing the final value of convolutional kernel rather than using a ranging block after every operation. To fully utilize the available FPGA resources, Residue-Net instantiates multiple convolutional kernels within the convolutional layer.

In DNNs, the output of the layers passes through a non-linear activation function. Rectified Linear Unit (ReLU) activation function is the most widely used activation function layer. To implement the ReLU activation function, Residue-Net needs to perform the comparison using the procedure explained in Section 3.1. Figure 3(c) shows the Residue-Net comparator. To compare two RNS numbers, \( R_a \) and \( R_b \) of both inputs are connected to a lookup table to the values of \( \alpha \) and \( \beta \). Residue-Net comparator calculates the difference of \( R_i \) of each input and their corresponding \( \beta \); the one with the larger difference is greater. For equal differences, the number with the greater \( a \) is larger. To implement the ReLU activation function, Residue-Net uses the implemented comparator with one input fixed. Residue-Net also uses the RNS comparator to implement max/min pooling layers.

In fully connected layers, similar to convolutional layers, multiple neurons are executing simultaneously. The architecture of each neuron is illustrated in Figure 3(d). Since the number of inputs in different fully connected layers varies, Residue-Net uses a generic architecture, which parallelizes the computation by instantiating multiple neurons, where each neuron multiplies and accumulates multiple inputs. Each neuron multiplies the outputs of the previous layer by the corresponding weights and accumulates the results. A series of Residue-Net multipliers multiply the weights by the inputs. The results are aggregated in a pipelined adder-tree and an accumulator adds up the intermediate results of consecutive iterations. The controller issues the memory access signals and reads all the inputs of the layer (\( P \) inputs per cycle), and at the end, converts the accumulated RNS result to valid RNS representation.

4 EXPERIMENTAL RESULT

4.1 Experimental Setup

Residue-Net migrates the neural network execution to the RNS domain to reduce the complexity of DNNs without losing the accuracy of the quantized network. To evaluate the efficiency of Residue-Net, we implemented four common network architectures, LeNet, AlexNet, VGG16, and ResNet50 on FPGA. Table 1 compares the Top-1 accuracy of Residue-Net with that of the 32-bit fixed-point, and networks quantized to 6-bit over four popular DNNs classifying various popular datasets (LeNet on MNIST, AlexNet on ImageNet, VGG16 and ResNet-50 on CIFAR-10). To train and quantize the networks we used the approach proposed in [12]. As presented in the table, Residue-Net provides almost the same accuracy as the full-precision network.

Residue-Net host code is written in OpenCL to convert the quantized weights to RNS. The host code uses the Xilinx Vitis software platform to transfer the primary inputs and RNS weights to FPGA DRAM and invoke the Residue-Net kernel. Residue-Net FPGA kernel is implemented in SystemVerilog HDL and synthesized using Xilinx Vivado Design Suite. The timing and the functionality of the Residue-Net are also verified using Vivado Design Suite. We implemented Residue-Net on Xilinx-7 FPGA KC705 Evaluation kit. To estimate the device power we used the built-in Xilinx Power Estimator tool in Vivado. To evaluate the efficiency, we compared Residue-Net with the baseline FPGA implementation. Since the dynamic range of RNS used in Residue-Net can only uniquely represent 6-bit binary numbers, we implemented the network quantized to 6-bit weights and activations in the binary system on FPGA with the same architecture. We used the same architecture for the baseline and Residue-Net to minimize the impact of the accelerator architecture on the evaluations and emphasize the effectiveness of migrating to RNS from the binary system and multiplication-free execution of DNNs. We also compared the efficiency of the building blocks of DNNs to represent the efficiencies and overheads of Residue-Net.

4.2 System Evaluation

The baseline FPGA implementation is designed to utilize \( \sim 90\% \) of the FPGA available LUTs which are the bottleneck of increasing the parallelism. We took two approaches in selecting the parallelism level of Residue-Net: i) Residue-Net (area-opt) targets minimizing the area and power consumption of the network. It provides the same performance as the baseline while reducing the power and area of the network. ii) Residue-Net (performance-opt) targets to fully utilize \( \sim 90\% \) the FPGA LUTs to improve the performance and energy consumption of the network. Figure 4 shows the throughput improvement and energy reduction values of all the four DNNs as compared to the baseline for both area-opt and performance-opt implementations. Residue-Net (area-opt) provides the same performance as the baseline with less area and consequently lower power and energy consumption. Residue-Net (area-opt) shows 24% energy reduction on average as compared to the baseline. Residue-Net (performance-opt), by increasing the parallelism provides 2.8x higher throughput, with a close area and power consumption to the baseline; thus increasing the energy efficiency by 2.7x. The main efficiency of the Residue-Net comes from the optimized MAC unit, therefore, Residue-Net shows higher performance improvement in networks with a higher number of MAC operations. As the number of MAC operations in LeNet is significantly less than those of the ResNet-50 the performance improvement in LeNet is less than that in ResNet-50 (2.1x as compared to 3.1x). Although Residue-Net requires more memory to store the weights (7 bits as compared to 6 bits in the baseline), since all of Residue-Net implementations are compute-bound, the memory and communication overhead is negligible.

Figure 5 shows the LUT, and BRAM utilization of the FPGA (divided by the total available resources), as well as the power, throughput, and energy consumption of both Residue-Net (area-opt) and Residue-Net (performance-opt) as compared to the baseline in AlexNet. Residue-Net (area-opt) shows 33% reduction in the number of required LUTs as compared to the baseline. Although Residue-Net increases the BRAM utilization because of using multiple memory instances to store the RNS weights and activations, this overhead does not affect the performance as BRAMs are not the bottleneck of the design. Residue-Net (area-opt) in AlexNet, is able to reduce the power consumption and consequently the energy consumption of classifying an input by 17%, while delivering the same performance as the baseline. Residue-Net (area-opt) on average, reduces the resource utilization by 36% and power consumption by 21%. Residue-Net (performance-opt), by fully utilizing the FPGA resources, is able to increase the throughput and
energy efficiency by 2.9x as compared to the AlexNet quantized to 6 bits. Since CNV and FC layers in DNNs contribute to the majority of execution time, Residue-Net (performance-opt) employs the saved resources to increase the parallelism in CNV and FC layers, thereby increasing the performance. Residue-Net (performance-opt) provides higher speed-up and energy reduction due to the efficiency of Residue-Net in executing MAC operations. In the following subsection, we evaluate the efficiency of DNN sub-modules to justify where the efficiency of the system comes from.

### 4.3 Operation Evaluation

Residue-Net tackles the computation complexity of MAC operation and particularly the complexity of the multiplication operation. It has been shown that convolutional layers consume 80% of the execution time [33]. To show the effectiveness of Residue-Net in breaking down the computation complexity of the multiplication operation we evaluate the efficiency of Residue-Net in executing the building blocks of convolutional, fully connected, activation function and pooling layer separately.

**Efficiency:** The efficiency of DNN building modules is illustrated in Figure 6. Figure 6(a) shows the LUT utilization, \(\text{LUTs} / \text{Total LUTs}\), of each module in FPGA. Utilization values not only demonstrate the reduction in the area achieved by Residue-Net but also show the relative size of each module as compared to the available FPGA resources. These building modules can be integrated into any DNN accelerator to reduce the area of the accelerators. Residue-Net shows 79.5% area reduction in implementing MAC operations which comprise more than 99% of DNN operations [6]. Residue-Net MAC module also shows 80% power reduction as compared to 6-bit binary MAC operation. In figure 6 power and performance are calculated for 100 MAC modules since due to the small size of the MAC module, the power consumption of a single MAC module would be close to 0 and hard to represent. We also evaluate the efficiency of four common convolutional kernels (CNV 3 × 3, CNV 4 × 4, CNV 5 × 5, CNV 11 × 11). Residue-Net convolutional kernels, on average, show 54.5% area reduction and 59% power reduction as compared to the baseline modules. Consuming most of the execution time of the network, convolutional layers can be considerably accelerated in Residue-Net.

In fully connected layers, a neuron multiplies its inputs by weights and accumulate the results. In Figure 6, 16 FC, 64 FC and 128 FC represent a neuron that has 16, 64, and 128 inputs respectively. For instance, 64 FC multiplies 64 inputs by weights and accumulate the results in an adder-tree with 64 inputs. Note that if the inputs of the layer are more than 64, the module calculates the result in multiple iterations. In fully connected neurons, Residue-Net reduces the area by 53% and power consumption by 57% as compared to the baseline.

Residue-Net shows a significant reduction in computation complexity of both convolutional and fully connected layers which are the most time-consuming parts of DNN execution. However, due to higher complexity in comparison operations in RNS, Residue-Net activation function and pooling layers require more resources than the baseline layers. Residue-Net requires 75% more resources to implement the activation function for 25 inputs than the baseline activation function, 215% more resources for implementing the pooling layer. However, these two sub-modules require significantly fewer resources than the FC and CNV modules, and the Residue-Net area overhead in these two layers will not considerably impact the effectiveness of Residue-Net.

**Overhead:** Table 2 shows the LUT overhead of Residue-Net as compared to the baseline. First, Residue-Net needs to convert inputs to RNS using forward conversion modules, each requires 5 LUTs. Also, after executing the network, Residue-Net converts the results back to the binary system using backward conversion modules. The backward conversion module is a table with all the possible binary numbers and maps RNS numbers to a unique binary number, which requires 12 LUTs to implement. After performing MAC operations, Residue-Net should calculate the residues using a ranging block that needs 9 LUTs. AF and pooling layers require comparison operators, each of which needs 15 more LUTs in Residue-Net as compared to the same operation in the binary system. Considering the limited number of each of these modules in
the entire system, the overhead of Residue-Net will be negligible in the efficiency of the accelerator.

### 5 CONCLUSION

In this paper, we proposed Residue-Net, a multiplication-free DNN accelerator using the residue number system. Residue-Net migrates a trained DNN quantized to 6-bit to RNS representation using [3, 4, 5] as the moduli set. Residue-Net breaks down the 6-bit operations to smaller operations. It also replaces complex multiplication operations with energy-efficient shift and addition operations. Residue-Net on average, shows 36% and 21% reduction in area and power respectively as compared to the baseline FPGA implementation with the same throughput when executing widely-used DNNs. Residue-Net, with the same area as the baseline implementation, shows 2.8x speedup on average (up to 3.1x in ResNet-50), while delivering the same accuracy as the network quantized to six bits.

### ACKNOWLEDGEMENTS

This work was supported in part by CRISP, one of six centers in JUMP, an SRC program sponsored by DARPA, in part by SRC Global Research Collaboration (GRC) grant, DARPA HyDDENN grant, and also NSF grants #2028040, #1730158, #1911095, #2003279 and #1826967.

### REFERENCES


