A Deep Convolutional Neural Network Based on Nested Residue Number System

Hiroki Nakahara
Ehime University, Japan

Tsutomu Sasao
Meiji University, Japan

***Abstract*—A pre-trained deep convolutional neural network (DCNN) is the feed-forward computation perspective which is widely used for the embedded vision systems. In the DCNN, the 2D convolutional operation occupies more than 90% of the computation time. Since the 2D convolutional operation performs massive multiply-accumulation (MAC) operations, conventional realizations could not implement a fully parallel DCNN. The RNS decomposes an integer into a tuple of L integers by residues of moduli set. Since no pair of modulus have a common factor with any other, the conventional RNS decomposes the MAC unit into circuits with different sizes. It means that the RNS could not utilize resources of an FPGA with uniform size. In this paper, we propose the nested RNS (NRNS), which recursively decompose the RNS. It can decompose the MAC unit into circuits with small sizes. In the DCNN using the NRNS, a 48-bit MAC unit is decomposed into 4-bit ones realized by look-up tables of the FPGA. In the system, we also use binary to NRNS converters and NRNS to binary converters. The binary to NRNS converter is realized by on-chip BRAMs, while the NRNS to binary one is realized by DSP blocks and BRAMs. Thus, a balanced usage of FPGA resources leads to a high clock frequency with less hardware. The ImageNet DCNN using the NRNS is implemented on a Xilinx Virtex VC707 evaluation board. As for the performance per area GOPS (Giga operations per second) per a slice, the proposed one is 5.86 times better than the existing best realization.**

I. Introduction

A. Deep Convolutional Neural Network (DCNN)

A deep neural network (DNN) consists of multi-layer neuron model. A deep convolutional neural network (DCNN) is a combination of the 2D convolutional layers and the DNN. Since the DCNN emulates the human vision, it has a high accuracy for an image recognition. The DCNN is widely used for embedded vision systems including a hand-written recognition [14], a face detector [19], a scene determination [18], and an object recognition [11].

In the embedded vision system, since the learning is done by off-line, we can only consider the execution of pre-trained DCNN for the run-time environment. With the increase of the number of layers, the DCNN can increase recognition accuracy. Thus, a large scale DCNN is desired. Since the existing system using a CPU is too slow, the acceleration of the DCNN is necessary for the real-time requirement of the embedded vision systems [14]. Most of the software-based DCNNs use the GPUs [3], [8], [23], [24]. Unfortunately, since the GPU consumes high power, they are unsuitable for the embedded system [9]. Thus, the FPGA-based DCNN is required for low-power and real-time embedded vision system. As for the recognition accuracy, the DCNN using a fixed point representation is almost the same as one using a floating point representation [11]. The FPGA can use an appropriate low precision representation which reduces the hardware resources and increases the clock frequency, while the GPU cannot do it. Another merit of the FPGA is a lower power consumption than the GPU. A previous work [9] showed that, as for the performance per power, the FPGA-based DCNN is about 10 times more efficient than the GPU-based one. Recently, the Microsoft implemented a high performance per power DCNN by using the FPGA cluster (Catapult) [6].

In the DCNN, the 2D convolution occupies more than 90% of computation time [7]. Thus, in this paper, we consider the acceleration of the 2D convolution, which can be realized by massive MAC (multiply-accumulation) operations. Although the modern FPGA has DSP blocks (DSP48E for the Xilinx FPGA) for the MAC operations, a large scale DCNN requires $O(n \cdot 2^{2n})$ DSP48Es, where n is the precision of the fixed point representation. Thus, a DCNN with a high performance per area is desired.

B. Related Work

The learning and predict by an artificial neural network was implemented on the FPGA [15]. The basic 2D convolutional circuits were reported in [10], [19]. The efficient utilization of both on-chip and off-chip memories was discussed in [12]. The general classifier including the DCNN was considered in [2]. Various optimization techniques for the DCNN on the FPGA were proposed: Reduction of multipliers by dynamic reconfiguration [4]; optimization of the memory bandwidth [18], and selection of optimal parameters for the high-level synthesis design [26].

C. Proposed Method

In this paper, we propose an area-performance efficient DCNN by reducing MAC units. First, we use a residue number system (RNS) [21], [25] to decompose n-input MAC units into smaller ones. Then, we realize the small MAC units by LUTs on an FPGA. The RNS decomposes an integer into a tuple of L integers by residues of moduli set. In the conventional RNS, since no pair of modulus have a common factor with any other, the resulting MAC unit have different sizes. This means that even if we used the RNS, we cannot utilize the FPGA resources of uniform size. In this paper, we propose *the nested RNS (NRNS)*, which recursively decompose the numbers in RNS. By using the NRNS, we can decompose the MAC unit into smaller ones with uniform sizes. In the DCNN, since the NRNS decomposes the 48-bit fixed point MAC units into 4-bit ones, they can be implemented by 8-input 4-output LUTs.

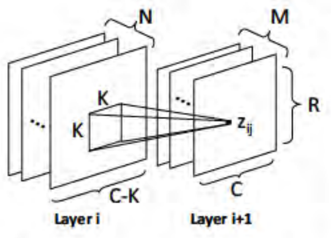


Fig. 1. 2D convolutional layer.

TABLE I. PARAMETER FOR IMAGENET.

Layer	M	N	R	C	K
1	3	48	55	55	11
2	48	128	27	27	5
3	256	192	13	13	3
4	192	192	13	13	3
5	192	128	13	13	3

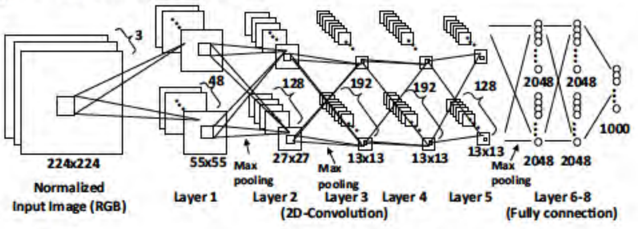


Fig. 2. ImageNet [13].

As an overhead, the NRNS system requires additional radix converters, which are realized by DSP48Es and BRAMs. Thus, the overhead of radix conversions can be neglected.

Contributions of the paper are as follows:

1. We proposed the NRNS which decomposes the MAC units into small LUTs. The NRNS can be applied to other digital signal systems such as FFT [22] and FIR [17] that use conventional RNS systems.
2. We realized converters between binary and RNS by BRAMs and DSP48Es, while the MAC units by LUTs. Thus, the proposed DCNN uses FPGA resources efficiently.
3. We implemented the ImageNet DCNN using 48-bit fixed-point representation on the Xilinx Virtex 7 VC707 evaluation board. As for the performance per area, the proposed one is 5.81 times higher than the existing best implementation.

D. Organization of the Paper

The rest of the paper is organized as follows: Chapter 2 introduces the deep convolutional neural network (DCNN); Chapter 3 introduces the residue number system (RNS); Chapter 4 proposes the DCNN using the NRNS; Chapter 5 shows the experimental results; and Chapter 6 concludes the paper.

II. DEEP CONVOLUTIONAL NEURAL NETWORK (DCNN)

A. 2D Convolutional Operation

In this section, we introduce the deep convolutional neural network (DCNN), and show the 2D convolutional operation which occupies a major computation power of the DCNN. The previous work [7] proved that 2D convolutional operations occupy more than 90% of the computation time. Thus, in this paper, we accelerate them in layers 1-5.

The DCNN consists of multiple feature maps and a classifier. To recognize the input image, first, the feature map reacts corresponding subdivided teacher data. Then, the classifier selects the appropriate reactions from feature maps. Usually,

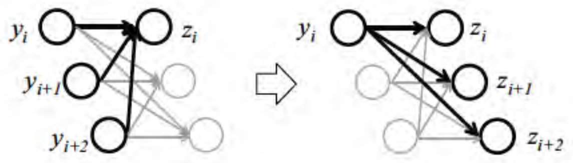


Fig. 3. Exchange of computation order [10].

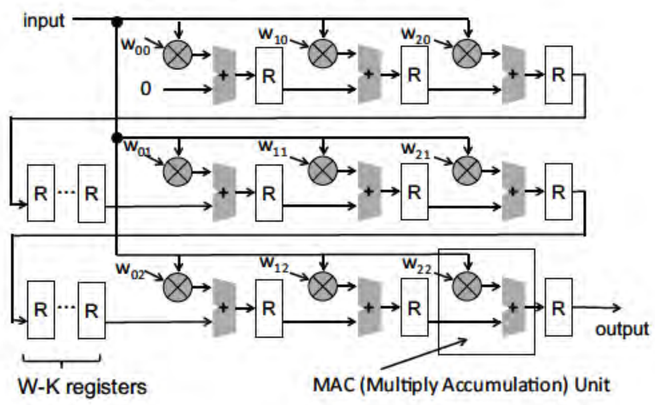


Fig. 4. Circuit for 2D convolution [10].

the classifier is realized by the neural network or the support vector machine (SVM). The typical feature map consists of a 2D convolutional layer, an activation function, and a sub-sampling (pooling) layer [1].

Fig. 1 shows the 2D convolutional layer, where N denotes the number of input layers and M denotes the number of output layers. The 2D convolutional layer computes the output by shifting a $K \times K$ size kernel. For M feature maps with $R \times C$ size, the following MAC (multiply-accumulation) operation is performed:

$$z_{ij} = \sum_{p=0}^{K-1}\sum_{q=0}^{K-1} y_{i+p,j+q}w_{pq}, \qquad (1)$$

where y_{ij} denotes an input value, w_{pq} denotes a weight coefficient, and z_{ij} denotes an output value. Fig. 2 shows the ImageNet [13] which is a real-life DCNN application. It consists of eight layers. The front five layers consist of 2D convolutional layers, while the rear three layers consist of fully connected neural networks. Note that, the ImageNet has dual layers 1-7. First, the ImageNet receives normalized three 224×224 images, those consist of an RGB image. Then, it categorizes the image into 1,000 classes. As shown in Fig. 2, the layer 1 receives 3 input images, and outputs a pair of 48 feature maps with 55×55 size. The kernel size of the layer 1 is 11×11 and the sliding window shifts across input image in a stride of 4 pixels. The following feature map has a similar structure. The stride for layer 2-5 is 1 pixel. Table I shows parameters of the ImageNet. From Table I, since the maximum kernel size is 11×11, the maximum number of MAC operations for a layer is 121. To reduce the bandwidth of the input for Expr. (1), we exchange the computation order as shown in Fig. 3. Fig. 4 shows the 2D convolutional circuit [10] based on Fig. 3. Consider the 48-bit fixed point representation, that is the highest precision of conventional

[1]Most of DCNN use the maximum pooling layer for 2×2 kernels.

implementation [4]. In this case, the 2D convolution works with $48 + 48 + \lceil log_2 121 \rceil = 103$ bits representation. After convolution, the output is rounded to 48 bits. Since it requires excessive MAC units (DSP48Es), implementation of a fully parallel 2D convolutional circuit is hard. In this paper, we decompose the n-input MAC unit by the nested residue number system (NRNS).

III. Residue Number System (RNS)

A. Definition

A **residue number system (RNS)** [21], [25], is defined by a set of L integer constants $\langle m_1, m_2, \ldots, m_L \rangle$, where no pair of modulus have a common factor with any other. An arbitrary integer Z can be uniquely represented by the RNS as a tuple of L integers $(Z_1, Z_2, \ldots, Z_L)$, where $Z_i \equiv Z \ (mod\ m_i)$.

$M = \prod_{i=1}^{L} m_i$ is a **dynamic range** of the RNS. In the RNS, the addition, the subtraction, and the multiplication can be performed in digit-wise. Let X and Y be integers, x_i and y_i be integers in the RNS defined by m_i $(1 \leq i \leq L)$, $\circ$ includes $+$ (addition), $-$ (subtraction), and $*$ (multiplication). Then $Z = X \circ Y$ satisfies $Z = (Z_1, Z_2, \ldots, Z_L)$, where $Z_i = (X_i \circ Y_i)\ mod\ m_i$. Note that, the division is not included in the operations.

Example 3.1: Let $\langle m_1, m_2, m_3 \rangle = \langle 3, 4, 5 \rangle$ be the moduli set. Consider the multiplication $X \times Y$, where $X = 8$ and $Y = 2$. Since $X \times Y = 16$, it is represented by $(1, 0, 1)$ in the RNS. X and Y is represented by $(2, 0, 3)$ and $(2, 2, 2)$ in the RNS, respectively. Thus, $X \times Y$ in the RNS is computed as follows:

$$\begin{aligned} X \times Y &= (4\ mod\ 3, 0\ mod\ 4, 6\ mod\ 5) \\ &= (1, 0, 1). \end{aligned}$$

■

In the RNS, the arithmetic operation is performed in digit-wise. This means that a large multiplier can be decomposed into smaller ones. In this paper, we realize them by LUTs on an FPGA. The multiplier on the RNS requires additional resources: A binary to RNS (*Bin2RNS*) converter and an RNS to binary (*RNS2Bin*) one. The modern FPGA has BRAMs and DSP48Es in addition to Slices. The usage of both BRAMs and DSP48Es for the converters hides the area overhead. In this paper, we implement the Bin2RNS converter by a cascade of BRAMs, while the RNS2Bin converter by the DSP48Es and BRAMs [1].

B. Realization of Bin2RNS Converter

Let $X = (x_1, x_2, \ldots, x_n)$, where $x_i = \{0, 1\}$. Consider the RNS $\langle m_1, m_2, \ldots, m_L \rangle$ for X. The single ROM realization of the Bin2RNS converter requires $2^n \sum_{i=1}^{L} \lceil log_2 m_i \rceil$ bits, which is too large to realize for a large n. By applying functional decompositions [5], we can reduce the total amount of memory.

Consider a function $F(X) : B^n \rightarrow \{0, 1, \ldots m-1\}$, where $B = \{0, 1\}$ and $X = (x_1, x_2, \ldots, x_n)$. Let (X_L, X_H) be a partition of X into two parts. A *decomposition chart* of F is the two-dimensional matrix, where each column label has distinct assignment of elements in X_L, and each row label has

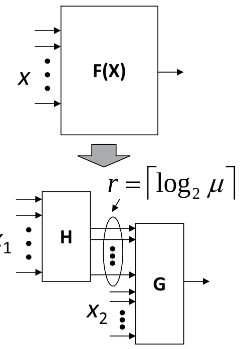
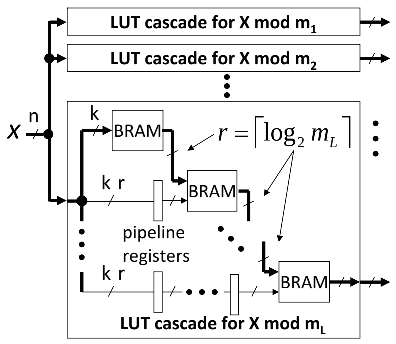


Fig. 5. Functional decomposition.



LUT cascade for X mod m_1
LUT cascade for X mod m_2
X
n
k
BRAM
$r = \lceil \log_2 m_L \rceil$
k r
BRAM
pipeline registers
k r
BRAM
LUT cascade for X mod m_L

Fig. 6. Bin2RNS converter by parallel LUT cascades.

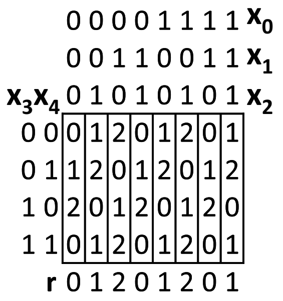


Fig. 7. Example of decomposition chart (X mod 3).

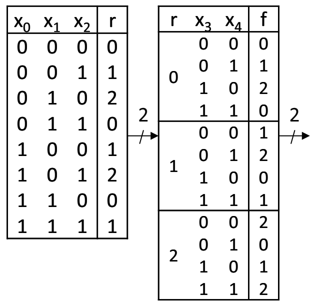


Fig. 8. An LUT cascade for X mod 3.

distinct assignment of elements in X_H, and the corresponding matrix value is $F(X_L, X_H)$. The number of different column patterns in the decomposition chart is the *column multiplicity* μ. X_L denotes the *bound variables*, and X_H denotes the *free variables*. Fig. 5 shows the functional decomposition. Connections with adjacent LUTs are called *rails*, where $r = \lceil log_2 \mu \rceil$. Let $|X_L| = n_1$ and $|X_H| = n_2$. In this case, the total amount of memory for the functional decomposition is $2^{n_1} \times r_1 + 2^{r_1 + n_2}$ bits. By applying the functional decomposition recursively, we have *an LUT cascade* [20]. Fig. 6 shows the Bin2RNS converter implemented by parallel LUT cascades. From the property of modulo operations, the column multiplicity for each modulo m_i is at most m_i. Let s be the number of 18Kb BRAMs with k inputs. As for the LUT cascade for modulo m_i, from Fig. 6, we have the following equation [20]:

$$k + (s-1)(k-r) \quad \geq \quad n,$$

where $r = \lceil log_2 m_i \rceil$ and $k > \lceil log_2 m_i \rceil$. Then, we have

$$s \quad = \quad \left\lceil \frac{n - \lceil log_2 m_i \rceil}{k - \lceil log_2 m_i \rceil} \right\rceil .$$

Example 3.2: Assume that $X = (x_0, x_1, x_2, x_3, x_4)$, $X_L = (x_0, x_1, x_2)$ and $X_H = (x_3, x_4)$. Fig. 7 shows the decomposition chart for $X\ mod\ 3$, and Fig. 8 shows its LUT cascade. Fig. 7 shows that the column multiplicity μ is 3, and the number of rails r is $\lceil log_2 3 \rceil = 2$. The single ROM realization for the Bin2RNS converter requires $2^5 \times 2 = 64$ bits, while the LUT cascade realization requires $2^3 \times 2 + 2^{2+2} \times 2 = 48$ bits. ■

C. Realization of RNS2Bin Converter

To realize the RNS2Bin converter, we use a formula $X_i = x_i |M_i^{-1}|_{m_i} \cdot M_i$, where $M_i = M/m_i$, $M_i^{-1} \cdot M_i =$

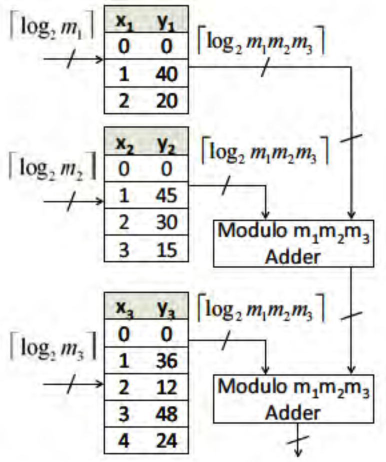


Fig. 9. RNS2Bin converter for RNS $\langle 3, 4, 5\rangle$.

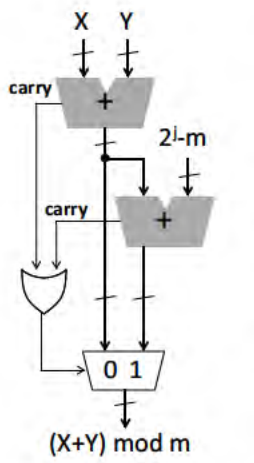


Fig. 10. Realization of modulo m adder.

1 [16]. Fig. 9 shows the RNS2Bin converter which converts the RNS $\langle 3, 4, 5\rangle$ to the binary. Assume that an integer X is represented by $(X_1, X_2, \ldots, X_L)$ in the RNS. As for X_i, the ROM generates M_i, the product of all the moduli except for m_i. The tree structured modulo adders produces X_i with a high clock frequency. As shown in Fig. 10, the modulo adder can be realized by two binary adders, an OR gate, and a multiplexer [1], where j denotes an integer such that $2^{j-1} < m \leq 2^j$. Since the RNS2Bin converter can be implemented with the $O(L)$ DSP48Es and BRAMs, they are not a bottleneck.

IV. DCNN Using Nested RNS (NRNS)

A. Nested RNS (NRNS)

Since moduli set of RNS consists of pairwise relatively prime numbers, we cannot realize them by LUTs of uniform sizes. Assume that an integer Z is represented by $Z = (Z_1, Z_2, \ldots, Z_L)$ in the RNS, and Z_i is represented by the RNS recursively. We call this **the nested RNS (NRNS)** which is defined by a set of j moduli $\langle m_{i1}, m_{i2}, \ldots, m_{ij}\rangle$, where no pair of modulus have a common factor with any other. In the NRNS, Z_i can be uniquely represented as a tuple of j integers as follows:

$$Z = (Z_1, Z_2, \ldots, (Z_{i1}, Z_{i2}, \ldots, Z_{ij})_i, \ldots, Z_L).$$

In this paper, we describe the original modulus m_i in the subscript of the right of the parenthesis. Since the NRNS can be applied to each element in the RNS recursively, moduli can be decomposed into uniform size. In this case, we must consider the dynamic range. Let A and B be integers represented by $A = (A_1, A_2, \ldots, A_L)$ and $B = (B_1, B_2, \ldots, B_L)$ in the RNS. By applying the NRNS to A_i and B_i, we have $A = \left(A_1, A_2, \ldots, (A_{i1}, A_{i2}, \ldots, A_{ij})_{m_i}, \ldots, A_L\right)$ and $B = \left(B_1, B_2, \ldots, (B_{i1}, B_{i2}, \ldots, B_{ij})_{m_i}, \ldots, B_L\right)$. Since the dynamic range for the NRNS is $M_{m_i} = \prod_{l=1}^{j} m_{il}$, the relation $M_{m_i} \geq A_i \circ B_i$ must be satisfied. In some applications, since $A_i \circ B_i$ is always less than the dynamic range, we can select a smaller moduli set. However, due to the page limitation, in this paper, we only consider the maximum dynamic range.

Example 4.3: Consider that the RNS $\langle 3, 4, 5\rangle$ is represented by the moduli set $\langle 3, 4, \langle 3, 4\rangle_5\rangle$ by the NRNS. Let $X = 17$ and $Y = 3$. In the NRNS, X and Y are represented by $X = (2, 1, (2, 2)_5)$ and $Y = (0, 3, (0, 3)_5)$. Thus, $X + Y$ in the NRNS is computed by

$$\begin{aligned} X + Y &= (2 \; mod \; 3, 4 \; mod \; 4, (2 \; mod \; 3, 5 \; mod \; 4)_5) \\ &= (2, 0, (2, 1)_5). \end{aligned}$$

By converting $(2, 1)_5$ in the RNS $\langle 3, 4\rangle_5$ to the binary number, we have 5. Note that, $5 \; mod \; 5 = 0$. Then, we have

$$(2, 0, (2, 1)_5) = (2, 0, 5) = (2, 0, 0).$$

By converting $(2, 0, 0)$ in the RNS $\langle 3, 4, 5\rangle$ into the binary number, we have 20, which is equal to the result of $17 + 3$. ■

B. Realization of NRNS Converters

By rewriting the output of the Bin2RNS converter shown in Fig. 6, we can realize the Bin2NRNS converter. Thus, we have following:

Corollary 4.1: Let an integer Z be represented by $(Z_1, Z_2, \ldots, Z_L)$ in the RNS, and m_i be the i-th modulus. When Z_i is represented by $(Z_{i1}, Z_{i2}, \ldots, Z_{ij})_i$ in the NRNS, the LUT cascade realizing Bin2NRNS converter has at most $\lceil log_2 m_i\rceil$ rails.

Corollary 4.1 shows that the amount of memory for the last BRAM only increases to store the NRNS integers. It is proportional to the number of moduli L. On the other hand, as shown in Fig. 9, the NRNS2Bin converter is realized by the tree of $O(L)$ RNS2Bin converters. We realize them by the BRAMs and the DSP48Es.

C. Moduli Set for NRNS

In this paper, we consider 48-bit fixed point representation, which has the highest precision of the conventional implementation. The circuit shown in Fig. 2 performs 2D convolution up to 11×11 kernel size. This means that it performs 121 MAC operations. Therefore, the dynamic range is at most $48 + 48 + \lceil log_2 121\rceil = 103$ bits. We choose the moduli set of the RNS from small integers so that they cover the dynamic range 2^{103}. In this case, we have the RNS moduli set as follows:

$$\langle 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83\rangle.$$

To reduce the MAC units, we apply the NRNS to moduli [2] that are greater than 15. Then, we have the NRNS moduli set as follows:

$$\begin{aligned} &\langle\, 3, 5, 7, 11, 13, \\ &\langle 3, 4, 5, 7, 11, 13\rangle_{17}, \\ &\langle 3, 4, 5, 7, 11, 13\rangle_{19}, \\ &\langle 3, 4, 5, 7, 11, 13, \langle 3, 4, 5, 7, 11, 13\rangle_{17}\rangle_{23}, \\ &\langle 3, 4, 5, 7, 11, 13, \langle 3, 4, 5, 7, 11, 13\rangle_{17}\rangle_{29}, \\ \ldots \; &, \langle 3, 4, 5, 7, 11, 13, \langle 3, 4, 5, 7, 11, 13\rangle_{17}\rangle_{83} \,\rangle . \end{aligned}$$

By applying the NRNS, we can decompose the 48-bit MAC unit into parallel 4-bit MACs. Therefore, we can realize them by 8-input 4-output LUTs.

[2]We can reduce the number of LUTs for the MAC operation which has greater than four bit modulus.

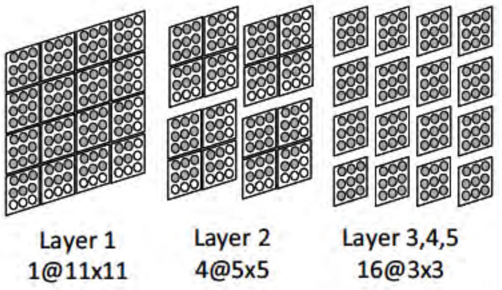


Fig. 11. Reconfiguration of the kernel by 3×3 2D convolutional circuits.

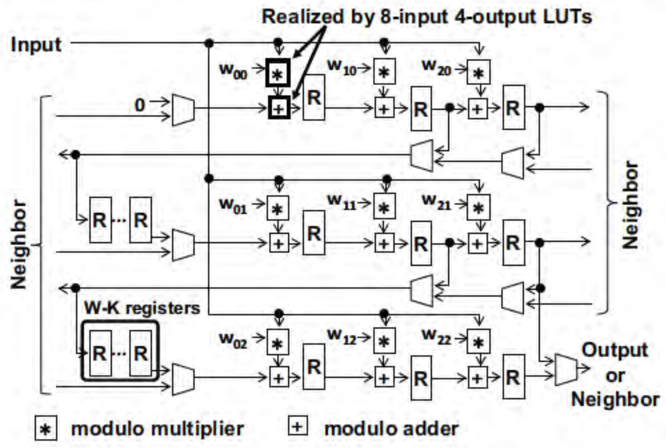


Fig. 12. Reconfigurable 3×3 2D convolutional circuit.

D. Proposed Architecture

As shown in Table I, different layers have different kernel size K. In this paper, we insert multiplexers into the 3×3 2D convolutional circuit shown in Fig. 4. The configuration bits are used to switch the kernel size as 11×11, four parallel 5×5, or 16 parallel 3×3 (Fig. 11).

Fig. 12 shows a reconfigurable 3×3 2D convolutional circuit using the NRNS. The proposed circuit decomposes 48-bit MAC unit into parallel 4-bit ones, and realize them by 8-input 4-output LUTs. Since it generates no carry signals, it operates with a high clock frequency. Let t be the number of neurons on the DCNN. The number of weights w_{ij} increases by $O(2^t)$. Thus, a large size DCNN cannot store weights into the BRAMs [12]. In this paper, we store them in the off-chip DDR3SODIMM. Fortunately, since weights are updated not so frequently, it has enough time to read from the off-chip DDR3SODIMM.

Fig. 13 shows the DCNN architecture using the NRNS. First, it reads the input images, the weights, and the configuration bits from the DDR3SODIMM. Second, it converts the binary signals into NRNS ones by the Bin2NRNS converter. Third, it performs the 2D convolution which has appropriate kernel size. Fourth, it converts the NRNS signals into 103 bits binary ones by the NRNS2Bin converter. Finally, the output is rounded to 48-bits signal. The on-chip memory is used to store the temporary data for different layers. The sequencer controls above operations.

V. Experimental Results

A. Implementation Results

We implemented the ImageNet using the NRNS on the Xilinx Inc. Virtex 7 VC707 evaluation board, which has the Xilinx Virtex 7 FPGA (XC7VX485T-2FFG1761C, 75,900 Slices, 2,060 18KbBRAMs, 2,160 DSP48Es) and the 1 Giga Byte DDR3SODIMM (Micron MT8JTF12864HZ-1G6G1, bus clock: 800 MHz, memory controller clock on the FPGA: 200 MHz). We designed the ImageNet using Convnet2 [8], then generated the trained DCNN with 48-bit fixed precision. We used the Xilinx Inc. Vivado 2014.1 with timing constrain 400 MHz. Our implementation used 52,380 Slices (69.0%), 1,235 18Kb BRAMs (59.9%), and 398 DSP48Es (18.4%). Also, it satisfied the timing constraint for real-time applications. Table II shows the computation time (ms) for each layer and the total one. The number of operations for the implemented DCNN was 132.2 GOPS (Giga Operations Per Seconds).

B. Comparison with Other Implementations

Since different implementations used different FPGAs and DCNNs, we used the performance per area efficiency (GOPS/Slice) to do fair comparison. Since the implemented DCNN operated 132.2 GOPS with 52,380 Slices, the performance per area is 25.2×10^{-4} (GOPS/Slice). Table III compares with other implementations. From Table III, as for the 48-bit fixed realization, the implemented one is 5.86 times higher than ISCA2010's implementation. Thus, our DCNN has the highest performance per area.

VI. Conclusion

In this paper, we proposed the NRNS which is a variant of the RNS. In the DCNN, the NRNS decomposed 48-bit MAC units into 4-bit parallel ones. Since they are realized by 8-input 4-output LUTs on the FPGA, it works with a high clock frequency. Also, we realized the binary to NRNS converter by on-chip BRAMs, while the NRNS to binary one by DSP48Es and BRAMs. Since our architecture utilizes most of the FPGA resources, the resource utilization efficiency is very high. The ImageNet DCNN using the NRNS was implemented on the Virtex VC707 evaluation board. As for the performance per area (GOPS per a slice), the proposed one is 5.86 times higher than the existing best realization.

Since the NRNS can decompose the MAC units into small LUTs, it can be applied to various digital signal processing systems (e.g., FIR filter and FFT). The future project is realized such applications by the NRNS.

VII. Acknowledgments

This research is supported in part by the Grants in Aid for Scientistic Research of JSPS.

References

[1] J. L. Beuchat, "Some modular adders and multipliers for field programmable gate arrays," *17th Int'l Symp. on Parallel and Distributed Processing (IPDPS)*, 2003, pp.190.2.

[2] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar and H. P. Graf, "A programmable parallel accelerator for learning and classification," *19th Int'l Conf. on Parallel architectures and compilation techniques (PACT2010)*, 2010, pp.273-284.

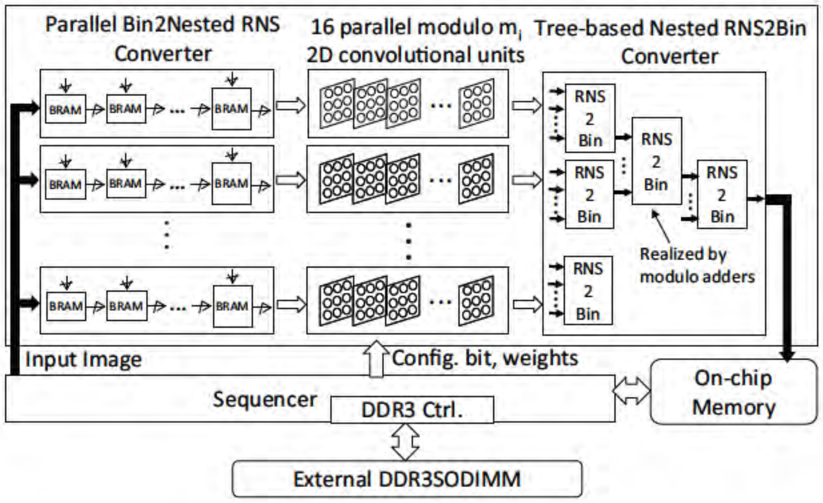

Parallel Bin2Nested RNS Converter
16 parallel modulo m_i 2D convolutional units
Tree-based Nested RNS2Bin Converter
BRAM
RNS 2 Bin
Realized by modulo adders
Input Image
Config. bit, weights
Sequencer
DDR3 Ctrl.
On-chip Memory
External DDR3SODIMM

Fig. 13. DCNN architecture using the NRNS.

TABLE II. COMPUTATION TIME [MS].

Layer	Time [ms]
1	4.36
2	2.80
3	1.30
4	0.98
5	0.64
Total	10.08

TABLE III. COMPARISON WITH OTHER IMPLEMENTATIONS.

	Precision	Clock Freq. [MHz]	FPGA	Performance [GOPS]	Normalized Performance [GOPS/Slice$\times 10^{-4}$]
ASAP2009 [19]	16bit fixed	115	Virtex5 LX330T	6.7	1.3
PACT2010 [2]	fixed	125	Virtex5 SX240T	7.0	1.9
FPL2009 [10]	48bit fixed	125	Spartan3A DSP3400	5.3	2.2
ISCA2010 [4]	48bit fixed	200	Virtex5 SX240T	16.0	4.3
ICCD2013 [18]	fixed	150	Virtex6 LVX240T	17.0	4.5
FPGA2015 [26]	32bit float	100	Virtex7 VX485T	61.6	8.1
Proposed	48bit fixed	400	Virtex7 VX485T	**132.2**	**25.2**

[3] Caffe: Deep learning framework, http://caffe.berkeleyvision.org/

[4] S. Chakradhar, M. Sankaradas, V. Jakkula and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *37th Annual Int'l Symp. on Computer architecture (ISCA2010)*, 2010, pp.247-257.

[5] H. A. Curtis, "A new approach to the design of switching circuits," *D. Van Nostrand Co.*, Princeton, NJ, 1962.

[6] T. Chilimbi, Y. Suzue, J. Apacible and K. Kalyanaraman, "Project Adam: Building an efficient and scalable deep learning training system," *11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[7] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," *Artificial Neural Networks and Machine Learning (ICANN2014)*, 2014, pp. 281-290.

[8] CUDA-Convnet2: Fast convolutional neural network in C++/CUDA, https://code.google.com/p/cuda-convnet2/

[9] A. Dundar, J. Jin, V. Gokhale, B. Martini and E. Culurciello, "Memory access optimized routing scheme for deep networks on a mobile coprocessor," *High Performance Extreme Computing Conference (HPEC2014)*, 2014, pp. 1-6.

[10] C. Farabet, C. Poulet, J. Y. Han and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," *FPL2009*, 2009, pp.32-37.

[11] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," *Int'l Symp. on Circuits and Systems (ISCAS2010)*, 2010, pp.257-260.

[12] S. K. Kim, L. C. McAfee, P. L. McMahon and K. Olukotun, "A highly scalable restricted Boltzmann machine FPGA implementation," *Int'l Conf. on Field Programmable Logic and Applications (FPL2009)*, 2009, pp.367-372.

[13] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet: Classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, 2012.

[14] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, Vol. 86, No. 11, 1998, pp.2278-2324.

[15] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*, *Springer*, 2006.

[16] A. R. Omondi and B. Premkumar, *Residue Number Systems*, *Imperial College Press*, 2007.

[17] P. Patronik, K. Berezowski, S. J. Piestrak, J. Biernat and A. Shrivastava, "Fast and energy-efficient constant-coefficient FIR filters using residue number system," *17th IEEE/ACM Int'l Symp. on Low-power Electronics and Design (ISLPED2011)*, 2011, pp.385-390.

[18] M. Peemen, A. A. A. Setio, B. Mesman and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," *31st International Conference on Computer Design (ICCD2013)*, 2013, pp.13-19.

[19] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," *20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP2009)*, 2009, pp.53-60.

[20] T. Sasao, *Memory-based Logic Synthesis*, *Springer*, 2011.

[21] F. J. Taylor, "Residue arithmetic: A tutorial with examples," *IEEE Trans. on Compt.*, Vol. 17, No. 5, pp.50-63, May, 1984.

[22] B. D. Tseng, G. A. Jullien, and W. C. Miller, "Implementation of FFT structures using the residue number system," *IEEE Trans. on Compt.*, Vol.100, No. 11, pp.831-845, 1979.

[23] Theano, http://deeplearning.net/software/theano/

[24] Torch: A scientific computing framework for LUTJIT, http://torch.ch/

[25] H. M. Yassine, "Fast arithmetic based on residue number system architectures," *ISCAS'91*, 1991, pp.2947-2950.

[26] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA2015)*, 2015, pp.161-170.